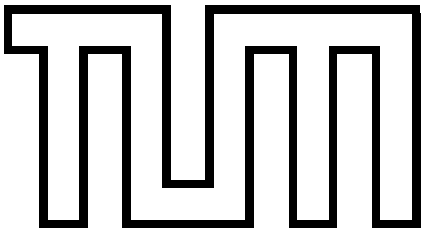
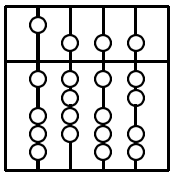
Technische Universität München



Fakultät für Informatik



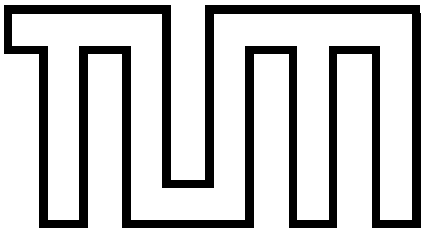
Diplomarbeit

Konstruktion guter unterer Schranken für Probleme aus APX

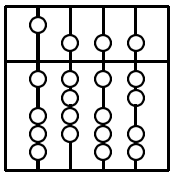
Ingo Rohloff

14. August 2001

Technische Universität München



Fakultät für Informatik



Diplomarbeit

Konstruktion guter unterer Schranken für Probleme aus APX

Themensteller : Prof. Dr. Angelika Steger

Betreuer : Prof. Dr. Angelika Steger
Bearbeiter : Ingo Rohloff

Abgabedatum : 15. August 2000

Ehrenwörtliche Erklärung

Ich versichere, daß ich diese Diplomarbeit selbständig verfaßt und
und nur die angegebenen Quellen verwendet habe.

München, den 14. August 2001 (Unterschrift des Kandidaten)

Kurzfassung

Für die Behandlung von NP-schweren Optimierungsproblemen ist man in der Praxis meist auf die Verwendung von Heuristiken bzw. Approximationsalgorithmen angewiesen. Als Maß für die Güte eines Approximationsalgorithmus nimmt man den maximalen Faktor, um den der vom Algorithmus ausgegebene Wert sich vom optimalen Wert unterscheiden kann. Verschiedene NP-schwere Probleme können sich bezüglich der Existenz guter Approximationsalgorithmen stark unterscheiden. Für manche Probleme (z.B. Knapsack) gibt es für jedes noch so kleine $\epsilon > 0$ einen polynomialen Approximationsalgorithmus mit Güte kleiner gleich $1 + \epsilon$. Für andere Probleme (z.B. TSP) ist bekannt, daß es, falls nicht $P = NP$, für kein $c \in N$ einen polynomialen Approximationsalgorithmus mit Güte kleiner gleich c geben kann.

Im Rahmen dieser Diplomarbeit sollen vor allem Probleme betrachtet werden, die in gewisser Weise „dazwischen“ liegen. D.h., für die es eine Konstante C gibt, so daß es einen polynomialen Approximationsalgorithmus mit Güte kleiner gleich C gibt, ein Approximationsalgorithmus mit Güte kleiner gleich $C - \epsilon$ aber nur im Falle $P = NP$ existiert.

Die Menge der so approximierbaren Optimierungsprobleme bezeichnet man mit APX. Bis auf wenige Ausnahmen kennt man für die Probleme aus APX zwar obere und untere Schranken für den Wert C, nicht aber den wahren Wert. Untere Schranken für die C-Werte lassen sich von einem Problem auf andere Probleme in APX übertragen, in dem man geeignete Reduktionen oder Ketten von Reduktionen angibt. Dabei hängt es von der Qualität der Reduktionen ab, wie gut die dabei erzielbaren Schranken sind. In der Diplomarbeit sollen die in der Literatur bekannten Reduktionstechniken zusammengetragen werden und dargestellt werden, welche unteren Schranken sich damit erzielen lassen.

Danksagung

Ich möchte mich zuerst bei allen bedanken, die mich bei dieser Diplomarbeit unterstützt haben:

Besonderer Dank geht an Frau Prof. Dr. Angelika Steger für die intensive Betreuung; ohne die zahlreichen und anregenden Diskussionen hätte die Diplomarbeit nur halb so viel Spaß gemacht.

Weiterhin möchte ich mich bei meinen Eltern und meiner Freundin für die moralische Unterstützung bedanken. Sie haben mir das Arbeiten ungemein erleichtert.

Als letztes möchte ich mich noch bei Marcel May für eine schöne und arbeitsame Woche am Bodensee bedanken, in der ein Großteil des Textes entstanden ist.

Inhaltsverzeichnis

1 Einleitung

Jeder Informatiker wird schon mal auf ein Problem gestoßen sein, für dessen (optimale) Lösung ihm einfach kein effizienter Algorithmus eingefallen ist. Wenn sich für ein Problem auch nach langem Suchen einfach kein effizienter Algorithmus finden läßt, dann stellt sich die Frage, ob man einfach nur nicht dir richtige Idee hat, oder ob es generell unmöglich ist, effizient eine (optimale) Lösung zu finden.

Mit genau dieser Frage beschäftigt sich die Theorie der NP-vollständigen Probleme. Mit dieser Theorie ist es möglich zu zeigen, daß ein neues Problem mindestens so schwer ist, wie eine ganze Klasse von alten schon bekannten Problemen, die sich ebenfalls standhaft einer effizienten Lösung entziehen. Eine ganze Reihe von solchen Problemen und die Transformation dieser Probleme aufeinander sind in dem Standardwerk von Garey und Johnson [22] beschrieben, inklusive der dazu nötigen Theorie.

Auch wenn ein endgültiger Beweis noch erbracht werden muß, daß es sich bei diesen Problemen wirklich um Probleme handelt, die nicht effizient lösbar sind, so weist doch die Tatsache, daß man inzwischen buchstäblich hunderte solcher Probleme kennt und für keines davon einen effizienten Algorithmus gefunden hat, darauf hin, daß diese Probleme tatsächlich „schwierig“ sind.

Wenn es sich bei einem Problem um ein Optimierungsproblem handelt, bei dem man zwar eine Lösung finden kann, aber eben nur sehr schwer die optimale, dann reicht es in der Praxis oft, wenn man einen Algorithmus angeben kann, der garantiert, daß die produzierte Lösung nicht zu weit vom Optimum entfernt ist.

Solche Algorithmen nennt man Approximations-Algorithmen. Im Lauf der Zeit hat man immer bessere Methoden entwickelt, wie man NP-schwere Optimierungsprobleme approximiert, aber es fehlten Beweise, die zeigen, daß sich bestimmte Probleme nicht beliebig genau approximieren lassen. Ein Haupt-Grund dafür war, daß es einige Zeit brauchte, bis man eine vernünftige Definition für die Transformation zwischen Optimierungsproblemen gefunden hatte. Eine weitere Schwierigkeit bestand darin, daß sich Nicht-Approximierbarkeit bis zur Entdeckung des PCP-Theorems generell nur bei wenigen Problemen beweisen ließ.

Ziel der Diplomarbeit war es, sich mit Problemen zu beschäftigen, für die man Algorithmen kennt, die bis auf einen konstanten Faktor an das Optimum herankommen. Die Klasse, zu der diese Probleme gehören, nennt man APX . Innerhalb dieser Klasse gibt es Probleme, von denen sich beweisen läßt, daß sie tatsächlich nicht besser als bis auf einen konstanten Faktor approximiert werden können, wenn NP-vollständige Probleme tatsächlich so schwer sind wie vermutet.

Die Frage ist, wie man solche unteren Schranken für die Approximierbarkeit eines Problems beweist und wie man sie verbessern kann.

Im Weiteren sollen zunächst die Grundbegriffe und Definitionen erklärt werden, mit denen man bei Problemen aus APX arbeitet. In den folgenden Kapiteln wird erläutert, wie man

mit Hilfe von speziellen Graphen eine untere Schranke für ein Problem beweisen kann und wie man versuchen kann, diese untere Schranke in einem speziellen Fall zu verbessern.

1.1 Probleme aus NP

Da die meisten Beweise auf der Annahme P $\neq$ NP beruhen, soll zunächst erklärt werden, was mit diesen beiden Begriffen gemeint ist.

Als Erstes werden nur Entscheidungsprobleme betrachtet, d.h. Probleme, bei denen es als Lösung nur „Ja“ oder „Nein“ gibt.

Jede Instanz eines Problems kann als binäre Zeichenkette aufgefaßt werden, d.h. als Zeichenvorrat nehmen wir $\Sigma = \{0, 1\}$ und als Zeichenkette $x \in \Sigma^*$. Ein Problem wird durch alle möglichen Instanzen des Problems $\mathcal{I} \subseteq \Sigma^*$ charakterisiert.

Wir gehen davon aus, daß man in linearer Zeit feststellen kann, ob $x \in \mathcal{I}$, d.h. ob es sich bei einer Zeichenkette x wirklich um eine Instanz eines Problems handelt oder ob es eine sinnlose Eingabe ist.

Definition: Entscheidungsproblem
Ein Entscheidungsproblem $\mathcal{P}$ ist ein Tupel $\langle \mathcal{I}, SOL \rangle$ so daß gilt:

- *Es ist $\mathcal{I} \subseteq \Sigma^*$ und es kann in linearer Zeit entschieden werden, ob für ein $x \in \Sigma^*$ auch $x \in \mathcal{I}$ gilt.*
- *Für jede Instanz $x \in \mathcal{I}$ ist $SOL(x) \subseteq \Sigma^*$ die Menge der Lösungen für x.*

Man sagt, ein Algorithmus $\mathcal{A}$ löst ein Entscheidungsproblem $\langle \mathcal{I}, SOL \rangle$, wenn der Algorithmus für jedes $x \in \mathcal{I}$ entweder JA ausgibt, wenn $SOL(x) \neq \emptyset$, oder NEIN ausgibt, wenn $SOL(x) = \emptyset$. Dabei muß der Algorithmus keine Lösung finden; es genügt wenn er feststellen kann, ob $SOL(x)$ leer ist oder nicht.

Definition: Klasse P
Die Klasse P *enthält alle Entscheidungsprobleme, die von einem Algorithmus in polynomieller Zeit gelöst werden können.*

Für viele Entscheidungsprobleme kennt man keinen Algorithmus, der einem in polynomieller Zeit eine Antwort liefert. Allerdings ist es oft nicht schwer zu überprüfen, ob eine Lösung $s \in SOL$ wirklich eine Lösung ist oder nicht. Eine solche Lösung könnte man auch als Beweis auffassen, daß die korrekte Antwort des Entscheidungsproblems JA lautet.

Entscheidungsprobleme, bei denen man in polynomieller Zeit überprüfen kann, ob ein Beweis im oben beschriebenen Sinne korrekt ist, gehören zur Klasse NP; die Abkürzung steht für *Nicht-deterministisch Polynomiell*:

Definition: Klasse NP
Ein Entscheidungsproblem $\mathcal{P} = \langle \mathcal{I}, SOL \rangle$ gehört zur Klasse NP, wenn gilt:

- *Die Größen der Lösungen einer Instanz $x \in \mathcal{I}$ sind durch ein Polynom in der Länge der Instanz beschränkt. D.h. es gibt ein Polynom p, so daß gilt:*

$$|s| \leq p(|x|) \qquad \forall x \in \mathcal{I} \wedge s \in SOL(x)$$

- *Es gibt einen Algorithmus $\mathcal{A}(x, s)$ und ein Polynom q, so daß $\forall x \in \mathcal{I}$ gilt: Wenn $s \in SOL(x)$ gibt $\mathcal{A}(x, s)$ innerhalb der Zeit $q(|x|)$ JA aus; an sonsten gibt er innerhalb der Zeit $q(|x|)$ NEIN aus.*

Wenn man zwei Probleme aus NP hat, dann stellt sich die Frage, ob sie gleich schwer sind oder nicht. Dazu definiert man folgende Transformation:

Definition: Karp-reduzierbar [30]
Ein Entscheidungsproblem $\mathcal{P}_{alt} = \langle \mathcal{I}_{alt}, SOL_{alt} \rangle$ ist Karp-reduzierbar auf ein Problem $\mathcal{P}_{neu} = \langle \mathcal{I}_{neu}, SOL_{neu} \rangle$, wenn es eine Funktion $f : \mathcal{I}_{alt} \rightarrow \mathcal{I}_{neu}$ gibt, die in polynomieller Zeit berechenbar ist und wenn gilt:

$$SOL_{alt}(x) \neq 0 \Longleftrightarrow SOL_{neu}(f(x)) \neq 0 \qquad \forall\ x \in \mathcal{I}_{alt}$$

Man schreibt : $\mathcal{P}_{alt} \leq_p \mathcal{P}_{neu}$

Mit einer solchen Transformation kann man $\mathcal{P}_{alt}$ lösen, wenn man weiß wie man $\mathcal{P}_{neu}$ löst; deshalb muß $\mathcal{P}_{neu}$ mindestens so schwer sein wie $\mathcal{P}_{alt}$.

Ein Problem, auf das sich jedes andere Problem aus NP reduzieren läßt, gehört demnach zu den schwersten Problemen aus NP ; Solche Probleme nennt man *NP-vollständig.*

Definition: NP-vollständig
Ein Entscheidungsproblem $\mathcal{P}$ wird NP-vollständig genannt,wenn $\mathcal{P} \in NP$, und für jedes Problem $\mathcal{P}' \in NP$ gilt: $\mathcal{P}' \leq_p \mathcal{P}$.

Das erste NP-vollständige Problem und der Begriff der NP-Vollständigkeit wurde unabhängig von Cook [17] und Levin [31] entdeckt. Cook bewies, daß das Problem

SAT: Gegeben sei ein boolesche Formel in Konjunktiver Normalform. Gibt es eine Belegung die die Formel erfüllt ?

NP-vollständig ist.

Kann man beweisen, daß sich SAT auf irgendein anderes Problem $\mathcal{P}$ Karp-reduzieren läßt, dann ist $\mathcal{P}$ ebenfalls NP-vollständig, denn man kann jedes andere Problem erst auf SAT und dann auf $\mathcal{P}$ reduzieren. Um zu zeigen, daß ein neues Problem NP-vollständig ist, reicht es deshalb zu zeigen, daß sich irgendein beliebiges NP-vollständiges Problem auf das neue Problem reduzieren läßt.

Mit dieser Methode hat man inzwischen von hunderten von Problemen bewiesen, daß sie NP-vollständig sind. Könnte man auch nur für ein einziges dieser Entscheidungsprobleme einen Algorithmus finden, der das Problem in polynomieller Zeit löst, könnte man alle Probleme in NP in polynomieller Zeit lösen.

1.2 Probleme aus NPO und APX

Oft hat man nicht ein Entscheidungsproblem als Grundlage, sondern ein Optimierungsproblem. Damit meint man ein Problem, von dem es normalerweise leicht ist *irgendeine* Lösung zu finden; die Frage, ob es eine Lösung gibt, kann deshalb praktisch immer mit JA beantwortet werden. Allerdings haben die Lösungen jetzt eine unterschiedlich gute Qualität. Die Aufgabe besteht deshalb darin, die beste Lösung – und nicht irgendeine – zu finden.

Im folgenden werden nur Optimierungsprobleme betrachtet, bei denen es immer eine Lösung gibt:

Definition: Optimierungsproblem
Ein Optimierungsproblem $\mathcal{P}$ ist ein Vier-Tupel $\langle \mathcal{I}, SOL, val, goal \rangle$:

- $\mathcal{I} \in \Sigma^*$ *ist die Menge aller Instanzen des Problems.*
- *Für jede Instanz $x \in \mathcal{I}$ wird mit $SOL(x) \subseteq \Sigma^*$ die Menge der Lösungen von x bezeichnet; diese Menge ist nie leer.*
- *Für jede Instanz x und für jede Lösung $s \in SOL(x)$ gibt der Integer-Wert $val(x, s)$ die Qualität der Lösung an.*
- $goal \in \{\min, \max\}$

Die Aufgabe bei einem Optimierungsproblem ist es, für eine vorgegebene Instanz x eine Lösung $s_{opt} \in SOL(x)$ zu finden, für die gilt:

$$opt(x) = val(x, s_{opt}) = \begin{cases} \min\{val(x,s) | s \in SOL(x)\} & \text{für } goal = \min \\ \max\{val(x,s) | s \in SOL(x)\} & \text{für } goal = \max \end{cases}$$

Definition: Klasse PO
Die Klasse PO enthält alle Optimierungsprobleme, für die man einen Algorithmus kennt, der das Optimum in polynomieller Laufzeit findet.

Als nächstes wird die Klasse NPO definiert, zu der alle Optimierungsprobleme gehören, die uns interessieren und bei denen man im allgemeinen keinen Algorithmus mit polynomieller Laufzeit zur Bestimmung der optimalen Lösung kennt:

Definition: Klasse NPO
Ein Optimierungsproblem $\mathcal{P} = \langle \mathcal{I}, SOL, val, goal \rangle$ gehört zu NPO wenn folgende Bedingungen erfüllt sind:

- *Für die Menge $\mathcal{I} \subseteq \Sigma^*$ kann in linearer Zeit überprüft werden, ob ein $x \in \Sigma^*$ Element von $\mathcal{I}$ ist oder nicht.*

- *Die Größen der Lösungen sind durch ein Polynom der Länge der Instanz beschränkt, d.h. es gibt ein Polynom p, so daß gilt:*

$$|s| \leq p(|x|) \qquad \forall x \in \mathcal{I} \wedge s \in SOL(x)$$

- *Man kann in polynomieller Zeit entscheiden, ob $s \in SOL(x)$ gilt.*
- *Die Funktion $val(x, s)$ ist in polynomieller Zeit berechenbar.*

Oft existieren von NP-vollständigen Entscheidungsproblemen Optimierungsvarianten, die offensichtlich nach der gleichen Lösung suchen und ebenso schwer sind.

Ein einfaches Beispiel ist SAT: Anstatt danach zu fragen, ob eine boolesche Formel in CNF erfüllbar ist, kann man auch versuchen, die Anzahl der erfüllten Klauseln zu maximieren. Hat man das optimale Ergebnis, kann man auch sofort entscheiden, ob es sich um eine erfüllbare Formel handelt oder nicht.

Kann man ein NP-vollständiges Entscheidungsproblem auf diese Weise auf ein Optimierungsproblem reduzieren, dann nennt man das Optimierungsproblem *NP-schwer*. Da die optimale Lösung mindestens so schwer zu finden ist, wie die Lösung des Entscheidungsproblems, untersucht man Algorithmen, die nicht unbedingt das Optimum finden.

Ein Approximations-Algorithmus ist ein Algorithmus, der für ein Optimierungsproblem eine beliebige Lösung findet. D.h. für ein $x \in \mathcal{I}$ liefert ein Algorithmus $\mathcal{A}$ ein $s \in SOL(x)$ zurück, oder als Formel: $\mathcal{A}(x) \in SOL(x)$.

Definition: Approximations-Algorithmus
Sei $\mathcal{P} = \langle \mathcal{I}, SOL, val, goal \rangle$ ein Optimierungsproblem. Ein Approximations-Algorithmus $\mathcal{A}$ mit Performance p ist ein Algorithmus für den gilt:

$$\frac{1}{p} \leq \frac{val(x, \mathcal{A}(x))}{opt(x)} \leq p \qquad \forall x \in \mathcal{I}$$

Kann man für ein Problem einen Algorithmus mit polynomieller Laufzeit angeben, der für jedes $\epsilon > 0$ die Performance $r = 1 + p$ hat, dann spricht man von einem *polynomial time approximation scheme* oder PTAS.

Definition: Klasse PTAS
Die Klasse PTAS enthält alle Optimierungsprobleme, für die man ein PTAS angeben kann.

Ist die Laufzeit des Algorithmus auch noch durch ein Polynom in $|x|$ *und* $\frac{1}{e}$ beschränkt, dann spricht man von einem *fully polynomial time approximation scheme* oder FPTAS.

Definition: Klasse FPTAS
Die Klasse FPTAS enthält alle Optimierungsprobleme, für die man ein FPTAS angeben kann.

Die Klasse der Optimierungsprobleme, mit denen wir uns beschäftigen ist folgendermaßen definiert:

Definition: Klasse APX
APX ist die Klasse der Optimierungsprobleme, für die es einen polynomiellen Approximations Algorithmus mit einer Performance $p \geq 1$ gibt.

Die Klasse APX enthält offensichtlich PO, PTAS und FPTAS. Die Probleme, denen in dieser Diplomarbeit besondere Aufmerksamkeit gewidmet wird, sind Probleme, die zu APX gehören und für die man gleichzeitig beweisen kann, daß es kein PTAS gibt.

1.3 Problemliste

In diesem Kapitel sollen die Probleme vorgestellt werden, die im weiteren Text erwähnt werden. Gleichzeitig dient die Liste als Namenskonvention für diese Arbeit.

Zuerst werden die verschiedenen Varianten von Satisfiability (d.h. Erfüllbarkeit) erklärt. Eine boolesche Formel über eine Menge von Variablen $X = \{x_1, x_2, \ldots, x_k\}$ ist rekursiv wie folgt definiert:

- Jede Variable x_i ist eine boolesche Formel.
- Die Negierung einer Formel F ist wieder eine boolesche Formel; in Zeichen: $\overline{F}$.
- Die Verknüpfung zweier boolescher Formeln F_1 und F_2 durch „und“ (Konjunktion) ist eine boolesche Formel; in Zeichen: $F_1 \wedge F_2$.
- Die Verknüpfung zweier boolescher Formeln F_1 und F_2 durch „oder“ (Disjunktion) ist eine boolesche Formel; in Zeichen: $F_1 \vee F_2$.

Eine negierte Variable $\overline{x_i}$ oder nicht negierte Variable x_i nennt man Literal. Eine boolesche Formel ist in konjunktiver Normalform (CNF), wenn sie als $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ geschrieben ist und jede Formel aus einer Disjunktion von Literalen besteht; das heißt z.B. : $F_1 = (x_1 \vee x_2 \vee x_3)$. Die Teilformeln $F_1, \ldots, F_n$ werden in diesem Fall „Klauseln“ genannt.

Eine Belegung weist jeder Variablen x_i einen Wert aus $\{wahr, falsch\}$ zu, d.h. sie kann als Funktion $f : X \to \{wahr, falsch\}$ verstanden werden.

Eine Belegung erfüllt eine boolesche Formel, wenn die Formel, sobald man sie mit der Belegung auswertet, „wahr“ ergibt. In Englisch heißt die Belegung deshalb *satisfying assignment*.

Folgende Problemvariationen werden unterschieden:

SAT	Das Entscheidungsproblem: Gegeben ist eine boolesche Formel in CNF. Gibt es eine Belegung, die die Formel erfüllt ?

3-SAT Das Entscheidungsproblem SAT, mit der Einschränkung, daß jede Klausel *höchstens* 3 Literale enthält.

E3-SAT Das Entscheidungsproblem SAT, mit der Einschränkung, daß jede Klausel *genau* 3 Literale enthält.

Da SAT das erste Problem war, von dem bewiesen wurde, daß es NP-vollständig ist und damit die Basis für alle weiteren NP-Vollständigkeits-Beweise ist, ist es natürlich besonders wichtig. Die Einschränkungen sind ebenfalls NP-vollständig und lassen sich leichter analysieren, besonders E3-SAT. Die Problemgröße $|x|$ einer Instanz wird oft als Anzahl der Klauseln angegeben.

Man kann noch die zusätzliche Einschränkung treffen, daß jede Variable nicht mehr als k-mal in der Formel auftaucht. In diesem Fall hängt man an die Bezeichnung ein kOCC an, z.B. E3-SAT-5OCC, wenn jede Variable nur 5mal vorkommen darf.

Neben den Entscheidungsproblemen gibt es natürlich auch die entsprechenden Maximierungsprobleme. Die Aufgabe lautet jetzt, mit einer Belegung möglichst viele Klauseln zu erfüllen. Die Optimierungsvarianten von SAT werden durch ein vorangestelltes MAX gekennzeichnet, z.B. MAX-E3-SAT.

Nahe Verwandte zu MAX-SAT sind die Optimierungsprobleme, bei denen es darum geht, ein System linearer Gleichungen über dem Körper F_2 zu lösen (d.h. eine Variable darf nur die Werte 1 und 0 annehmen). Hier gibt es:

MAX-LIN2 Gegeben sei ein (überbestimmtes) lineares Gleichungssystem über dem F_2. Bestimme eine Belegung für die vorkommenden Variablen, die die Anzahl der erfüllten Gleichungen maximiert.

MAX-Ek-LIN2 Das Problem MAX-LIN2 unter der Einschränkung, daß jede Gleichung genau k Variablen enthält.

Als Problemgröße wird oft die Anzahl der Gleichungen verwendet. Das Nicht-Approximierbarkeits-Ergebnis für MAX-E3-LIN2 von Håstad [26] ist Ausgangsbasis für eine ganze Reihe von weiteren Nicht-Approximierbarkeits-Beweisen. Es sei noch erwähnt, daß es *kein* entsprechendes Entscheidungsproblem gibt. Kann man ein Gleichungssystem lösen, d.h. alle Gleichungen erfüllen, dann findet man die Lösung mit Hilfe des Gaußschen Algorithmus in polynomieller Zeit.

Man kann wieder die Anzahl der Vorkommen einer Variablen einschränken, indem man ein kOCC anhängt. Z.B. MAX-E2-LIN2-3OCC.

Die beiden beschriebenen Problemklassen haben eine ähnliche Struktur. Es gibt einen Satz von Variablen $X = \{x_1, \ldots, x_n\}$ und eine Reihe von Bedingungen (in Englisch *Constraints*), die erfüllt werden sollen. Man kann diese Idee generalisieren und erhält dann die so genannten *Constraint Satisfaction Problems* (CSPs):

Definition: Constraint-Funktion
Eine Constraint-Funktion mit k Parametern ist eine Funktion der Form:
$f : \{0,1\}^k \to \{0,1\}$

Definition: Constraint
Gegeben sei eine Menge von Variablen $X = \{x_1, \ldots, x_n\}$. *Ein* Constraint *ist ein Paar* $\mathcal{C} = (f, (i_1, \ldots, i_k))$, *wobei* f *eine Constraint-Funktion mit* k *Parametern ist und* $1 \leq i_1, \ldots, i_k \leq n$ *gilt.*

Ein Constraint ist durch eine Belegung $\vec{x}$ für $x_1, \ldots, x_n$ erfüllt, wenn gilt:
$f(\vec{x}) = 1$

Eine *Constraint Familie* ist eine endliche Menge $\mathcal{F}$, deren Elemente Constraint Funktionen sind.

Ein Constraint $(f, (i_1, \ldots, i_k))$ gehört zu $\mathcal{F}$ wenn $f \in \mathcal{F}$.

Ein CSP wird jetzt definiert als:

MAX $\mathcal{F}$ Gegeben ist eine Menge an Variablen $X = \{x_1, \ldots, x_n\}$ und eine Menge an Constraints $\{C_1, \ldots, C_m\}$ für X aus $\mathcal{F}$. Finde eine Belegung, die die Anzahl der erfüllten Constraints maximiert.

Als nächstes werden ein paar Graphenprobleme definiert. Ein Graphenproblem, das auch zu den CSPs gehört ist:

MAX CUT Gegeben ist ein Graph $G(V, E)$. Finde eine Teilmenge $X \subset V$, so daß die Anzahl der Kanten $\{x, y\} \in E$, die X mit V verbinden maximiert wird. Eine Kante verbindet X mit V wenn entweder $x \in X$ und $y \in V \setminus X$ gilt, oder $x \in V \setminus X$ und $y \in X$.

Zwei prominente Graphenprobleme, die nicht zu APX gehören (zumindest ohne Einschränkungen), und die auch keine CSPs darstellen sind:

TSP Die Abkürzung steht für Traveling Sales Person. Gegeben ist ein vollständiger Graph $G(V, E)$ und eine Gewichtsfunktion $g : E \to N$. Finde eine Tour durch G, die die Summe der Gewichte der benutzten Kanten minimiert. Eine Tour ist dabei eine Permutation aller Knoten $V = \{v_1, \ldots, v_n\}$; bei 5 Knoten z.B. v_3, v_1, v_4, v_2, v_5; bei dieser Tour werden die Kanten $\{v_3, v_1\}, \{v_1, v_4\}, \ldots, \{v_5, v_3\}$ benutzt.

MAX CLIQUE Gegeben sei ein Graph $G(V, E)$. Finde eine Teilmenge $X \subset V$, so daß X eine CLIQUE ist und die Anzahl der Knoten in X maximiert wird. X ist eine CLIQUE, wenn für jedes Knotenpaar $x_1, x_2 \in X$ gilt $\{x_1, x_2\} \in E$.

Der große Unterschied der beiden letzten Probleme zu CSPs ist, daß sie nicht aus einer Reihe von unabhängigen Constraints bestehen. In einem CSP kann man ein Constraint überprüfen, ohne die restlichen Constraints zu berücksichtigen.

Weder bei TSP noch bei MAX CLIQUE gibt es solche lokalen Constraints. Die beiden Probleme sind sehr viel globalerer Natur.

2 Nicht-Approximierbarkeit und das PCP-Theorem

Viele Probleme aus der Praxis sind Optimierungsprobleme aus NPO. Durch einen Beweis, daß es sich bei einem Optimierungsproblem um ein NP-schweres Problem handelt, weiß man zwar, daß es (wenn NP $\neq$ P) keinen Algorithmus gibt, der immer die optimale Lösung findet, aber man weiß nichts über die Approximierbarkeit eines Problems.

Da man in der Praxis zumindest eine einigermaßen gute Lösung benötigt, sind für viele Probleme Algorithmen mit einer garantierten Performance bekannt; die Techniken, um solche Algorithmen zu entwickeln, wurden ebenfalls immer weiter verfeinert.

Beweise für die Tatsache, daß sich ein Problem *nicht* beliebig gut approximieren läßt, gab es allerdings anfangs nur sehr wenige; die ersten dieser Beweise benutzen einige grundlegende Techniken, die im Folgenden näher erläutert werden.

2.1 Bootstrapping

Nehmen wir an, wir haben einen Algorithmus für ein Problem, der eine garantierte Performance besitzt. Ein offensichtlicher Ansatz, die Performance zu verbessern ist es, die Eingabe durch eine Vorverarbeitung so aufzubereiten, daß sich die Performance erhöht. Kann man die Vorverarbeitung wiederholt anwenden, läßt sich die Performance manchmal immer weiter verbessern.

Mit dieser Vorgehensweise man kann auch negative Ergebnisse beweisen.

Zum Beispiel läßt sich zeigen:
Wenn es einen Algorithmus gibt, der für MAX-CLIQUE eine Lösung mit einer Performance von mindestens r_{alt} findet, dann gibt es auch einen Algorithmus, der eine Lösung mit einer beliebigen Performance $r_{neu} > 1$ findet.

Beweis:
Nehmen wir an, es ist ein Graph $G = (V, E)$ gegeben. Aus dem Graphen wird ein neuer Graph $G' = (V', E')$ gebaut. Die Knotenmenge V' ist $V \times V$, wobei mit $\times$ das kartesische Produkt gemeint ist. Für die Kantenmenge E' gilt:

$$\{(x_1, y_1), (x_2, y_2)\} \in E' \iff (\{x_1, x_2\} \in E) \vee (x_1 = x_2 \wedge \{y_1, y_2\} \in E)$$

Es gilt: $|V'| = |V|^2$ und $|E'| = O(|V|^4)$.

Nehmen wir an, C ist eine Clique in G. Man kann leicht überprüfen, daß es dann in G' auch eine Clique C' gibt, die aus den Knoten

$$C' = \{(x, y) : x \in C \wedge y \in C\}$$

besteht; die Umkehrung gilt ebenfalls und es gilt $|C'| = |C|^2$. Daraus folgt natürlich auch $opt(G') = opt(G)^2$.

Nehmen wir an, es gäbe einen Algorithmus $\mathcal{A}$, der eine garantierte Performance von r_{alt} hat. Wendet man diesen Algorithmus auf G' an, erhält man eine Clique C', für die gilt:

$$|C'| \geq \frac{opt(G')}{r_{alt}}$$

Betrachtet man die zu C' korrespondierende Clique C in G, dann gilt deshalb offensichtlich:

$$|C| = \sqrt{|C'|} \geq \sqrt{\frac{opt(G')}{r_{alt}}} = \frac{opt(G)}{\sqrt{r_{alt}}}$$

Das bedeutet, man hat durch die Umwandlung von G in G' einen Algorithmus gefunden, der für G eine Performance von mindestens $r_{neu} = \sqrt{r_{alt}}$ hat.

Da man diese Umwandlung wiederholt anwenden kann, kann man die Performance beliebig nahe an 1 annähern (auch wenn die Laufzeit dabei natürlich immer länger wird.)

Tatsächlich war Håstad [25] in der Lage zu zeigen, daß sich MAX CLIQUE für einen Graphen mit n Knoten nicht besser als $n^{1-\epsilon}$ approximieren läßt. MAX CLIQUE gehört also nicht zu APX.

Eine Variante des Bootstrappings erlaubt es einem zu zeigen, daß ein Problem $\mathcal{P}$ nicht in FPTAS $\setminus$ PO sein kann, wenn das Maß der optimalen Lösung durch ein Polynom in der Größe der Eingabe beschränkt ist.

Beweis:
Nehmen wir an, $\mathcal{P} = \langle \mathcal{I}, SOL, val, goal \rangle$ gehört zu FPTAS; deshalb gibt es einen Algorithmus $\mathcal{A}(x, \epsilon) \in SOL(x)$, der eine Lösung für $x \in \mathcal{I}$ mit Performance $1+\epsilon$ findet und dessen Laufzeit durch ein Polynom in $|x|$ und $1/\epsilon$ beschränkt ist.

Nehmen wir an, daß gilt $opt(x) \leq p(|x|)$ und p ein Polynom ist. Wir wählen dann $\epsilon = 1/(p(|x|)+1)$; daraus folgt:

$$\frac{1}{1+\frac{1}{p(|x|)+1}} = \frac{1}{1+\epsilon} \leq \frac{val(x, \mathcal{A}(x,\epsilon))}{opt(x)} \leq 1+\epsilon = 1 + \frac{1}{p(|x|)+1}$$

Für ein Minimierungsproblem folgt:

$$\begin{aligned} val(x, \mathcal{A}(x,\epsilon)) &\leq opt(x) + \frac{opt(x)}{p(|x|)+1} \\ val(x, \mathcal{A}(x,\epsilon)) - opt(x) &\leq \frac{opt(x)}{p(|x|)+1} < 1 \end{aligned}$$

Für ein Maximierungsproblem folgt:

$$opt(x) \leq val(x, \mathcal{A}(x,\epsilon)) + \frac{val(x, \mathcal{A}(x,\epsilon))}{p(|x|)+1}$$

$$opt(x) - val(x, \mathcal{A}(x,\epsilon)) \quad \leq \quad \frac{val(x, \mathcal{A}(x,\epsilon))}{p(|x|)+1} \leq \frac{opt(x)}{p(|x|)+1} < 1$$

Da laut der Definition von Optimierungsproblemen *opt* und *val* Integerwerte zurückliefern, hat der Algorithmus die optimale Lösung gefunden. Die Laufzeit ist durch ein Polynom in $|x|$ und $1/\epsilon = 1 + p(|x|)$ beschränkt; d.h. sie ist insgesamt polynomiell in $|x|$. Also muß $\mathcal{P} \in$ PO gelten.

2.2 Die „Gap“ Technik

Mit einer anderen Technik läßt sich zeigen, daß das allgemeine Traveling Sales Person (TSP) Problem nicht zu APX gehören kann, d.h. das es keinen Algorithmus mit garantierter Performance $r > 1$ für dieses Problem geben kann. Wir folgen der Darstellung in [8].

Dazu zeigt man, daß man mit Hilfe eines solchen Algorithmus ein NP-vollständiges Problem in polynomieller Zeit lösen könnte; wenn gilt NP $\neq$ P , dann ist das ein Widerspruch, d.h. die Behauptung das es einen solchen Algorithmus gibt ist falsch.

Das Problem, das als Ausgangsbasis dient, ist das Hamiltonsche Kreis (HK) Entscheidungsproblem; d.h. gegeben sei ein Graph $G(V, E)$ mit $n = |V|$ und man soll entscheiden, ob dieser Graph einen Kreis enthält, der durch alle Knoten geht.

Dieses Problem ist schon in der Struktur ähnlich zu TSP; um HK zu lösen, bauen wir aus G wieder einen neuen vollständigen Graphen $G'(V', E')$, bei dem $V' = V$ ist und eine Kante $\{x, y\} \in E'$ folgendes Gewicht $g(x, y)$ erhält:

$$g(x,y) = \begin{cases} 1 & \text{wenn } \{x,y\} \in E, \\ 1+\delta n & \text{sonst.} \end{cases}$$

Wenn es in G einen Hamiltonschen Kreis gibt, dann hat die optimale Tour für das TSP Problem in G' eine Länge von genau n, weil dann genau n Kanten der Länge 1 benutzt werden. Eine Lösung, die keinen Hamiltonschen Kreis in G benutzt, hat als Länge mindestens $n + \delta n = n(1 + \delta)$.

Hätte man einen polynomiellen Algorithmus für TSP mit einer besseren Performance als $1 + \delta$, dann würde dieser Algorithmus – wenn es einen Hamiltonschen Kreis gibt – immer das Optimum finden. Damit könnte man das HK Entscheidungsproblem in polynomieller Zeit lösen; deshalb kann es so einen Algorithmus – unter der Annahme NP $\neq$ P – nicht geben.

Da δ nicht weiter eingeschränkt worden ist, kann TSP nicht zu APX gehören.

Ein weiteres Beispiel dieser Technik ist das 3-Farben-Problem: „Wenn man einen Graph hat, kann man dann mit 3 Farben jeden Knoten so einfärben, daß benachbarte Knoten unterschiedliche Farben haben? “. Dieses Entscheidungsproblem ist ebenfalls NP-vollständig [35].

Betrachtet man die Optimierungsvariante, bei der gefragt ist, wie viele Farben benötigt werden, dann kann es keinen Algorithmus geben, der eine bessere Performance als 4/3 hat. Ein solcher Algorithmus müßte nämlich für jeden Graphen der 3-färbbar ist, die entsprechende Färbung finden, und damit könnte man das Entscheidungsproblem in polynomieller Zeit lösen.

Formal kann man die benutzte Technik so formulieren (nach [8]):

Sei $\mathcal{D}$ ein NP-vollständiges Entscheidungsproblem und $\mathcal{P}$ ein Minimierungsproblem aus NPO. Nehmen wir an, daß es zwei Funktionen $f : \mathcal{I}_D \to \mathcal{I}_P$ und $c : \mathcal{I}_D \to N$ gibt und eine Konstante $\mathtt{gap} > 0$*, so daß für alle $x \in \mathcal{I}_D$ gilt:*

$$\begin{array}{lcll} opt(f(x)) & \leq & c(x) & wenn \quad SOL(x) \neq \emptyset \\ opt(f(x)) & \geq & c(x)(1+\mathtt{gap}) & wenn \quad SOL(x) = \emptyset \end{array}$$

Dann kann es keinen Approximations Algorithmus für $\mathcal{P}$ mit einer besseren Performance als $1+\mathtt{gap}$ *geben, falls NP $\neq$ P .*

Beweis:
Nehmen wir an, es gäbe einen Algorithmus $\mathcal{A}$, der eine bessere Performance als $(1+\mathtt{gap})$ hat. Wir können jetzt das Problem $\mathcal{D}$ in polynomieller Zeit lösen, indem wir den Algorithmus für $\mathcal{P}$ verwenden. Dazu benutzt man die Instanz $y = f(x)$ von $\mathcal{P}$:

- Nehmen wir an $SOL(x) = \emptyset$. In diesem Fall ist $opt(y) \geq c(x)(1+\mathtt{gap})$. Der Algorithmus liefert also eine Lösung $s = \mathcal{A}(y)$, bei der auf jeden Fall $val(y,s) \geq c(x)(1+\mathtt{gap})$ gilt.

- Wenn $SOL(x) \neq \emptyset$, dann ist $opt(y) \leq c(x)$. Weil der Algorithmus eine bessere Performance als $(1+\text{gap})$ hat, gilt für die ausgegebene Lösung $s = \mathcal{A}(y)$ auf jeden Fall $val(y,s) < c(x)(1+\mathtt{gap})$

Da man $c(x)$,$f(x)$,s und $val(y,s)$ zusammen in polynomieller Zeit berechnen kann, kann man an Hand von $val(y,s)$ das Entscheidungsproblem lösen. Das ist ein Widerspruch zur Annahme NP $\neq$ P .

Mit genau der gleichen Argumentation, kann man die Gap-Technik auch auf Maximierungsprobleme anwenden. Dazu muß das Optimum von $f(x)$ nur entweder größer als $c(x)$ oder kleiner als $c(x)/(1+\text{gap})$ sein.

Leider ist diese Technik nur schwer anwendbar, wenn bei einem Problem das Optimum mit wachsender Größe der Instanz gegen unendlich geht.

Nehmen wir zum Beispiel 3-SAT: Wenn man eine Instanz x von 3-SAT betrachtet, dann stellt die Frage der Erfüllbarkeit ein NP-vollständiges Entscheidungsproblem dar. Nimmt man als Optimierungsproblem MAX-3-SAT, dann kann man als $c(x)$ einfach die Anzahl

der Klauseln nehmen. Wenn diese Anzahl m ist, dann müßte für nicht erfüllbare Instanzen gelten:

$$\begin{aligned} opt(x) &\leq \frac{m}{(1+\texttt{gap})} \\ \texttt{gap} &\leq \frac{m}{opt(x)} - 1 \\ \texttt{gap} &\leq \frac{m - opt(x)}{opt(x)} \end{aligned}$$

Das bedeutet, das Verhältnis zwischen der Anzahl an erfüllbaren Klauseln ($opt(x)$) und der Anzahl an nicht-erfüllbaren Klauseln ($m - opt(x)$) müßte bei nicht erfüllbaren Instanzen immer größer als die (beliebige) Konstante `gap` sein. Leider ist nicht klar, warum bei einer beliebigen Instanz von 3-SAT die optimale Lösung nicht z.B. genau $m-1$ Klauseln erfüllen kann. Die `gap` geht bei solchen Instanzen mit wachsendem m gegen Null.

Nehmen wir an, eine Instanz von 3-SAT könnte auf eine andere Instanz von 3-SAT Karp-reduziert werden, bei der gilt: Ist die Instanz unerfüllbar, dann ist das Verhältnis zwischen unerfüllten und erfüllten Klauseln bei einer optimalen Belegung größer als eine (kleine) Konstante `gap`.

Gäbe es eine solche Reduktion, dann könnte man die Gap-Technik auf die reduzierte Instanz anwenden und würde sofort ein Nicht-Approximierbarkeits-Ergebnis für MAX-3-SAT erhalten.

Leider kennt man bis jetzt keine einfache Reduktion, die diese Eigenschaft besitzt. Die einzige (theoretische) Methode, eine solche Reduktion zu bauen, basiert auf einem Theorem, das erst vor weniger als 10 Jahren entdeckt worden ist.

2.3 Das PCP-Theorem

Die Frage nach unteren Schranken für Approximationsalgorithmen wurde bereits ausführlich bei Garey und Johnson in ihrem Buch über NP-vollständige Probleme diskutiert [22].

Allerdings wurden wirklich aussagekräftige Ergebnisse erst möglich, nachdem man die Verbindung zwischen „Interactive Proofs“ und der Approximierbarkeit von Problemen herstellte [19, 16]. Interactive Proofs wurden ursprünglich aus einer anderen Motivation heraus unabhängig von Goldwasser, Micali, Rackoff [24] und Babai [10, 9] eingeführt. Die Verallgemeinerung zu „multi-prover protocols“ stammt von Ben-Or, Goldwasser, Kilian, Widgerson [12].

Zum Verständnis der folgenden Erläuterungen muß man sich noch einmal die Definition der Klasse NP ins Gedächtnis rufen: Ein Entscheidungsproblem gehört zu NP, wenn man einen Beweis für eine positive Antwort in polynomieller Zeit überprüfen kann.

Nehmen wir als Beispiel wieder SAT. Ein Beweis für die Erfüllbarkeit einer booleschen Formel ist einfach eine Belegung der Variablen, bei der die Formel erfüllt ist. Angenommen

ein Prüfer würde nur einen Teil des Beweises, d.h. nur einen Teil der Belegung für die Variablen erfahren. Eventuell könnte er dann bereits entscheiden, ob der Beweis korrekt ist oder nicht; wenn zum Beispiel eine Klausel von dieser partiellen Belegung bereits nicht erfüllt wird, dann kann man den Beweis sicher ablehnen. Die Entscheidung, ob abgelehnt wird oder nicht, hängt also offensichtlich davon ab, welchen Teil des Beweises der Prüfer zu sehen bekommt.

Dieses Konzept des Prüfers soll jetzt formalisiert werden:

Definition: Verifier
Ein Verifier ist ein Algorithmus, der als Eingabe drei Dinge bekommt:

1. *Eine Instanz $x \in \mathcal{I}$ eines Entscheidungsproblems $\langle \mathcal{I}, SOL \rangle$.*
2. *Einen Bitstring r mit zufälligen Bits.*
3. *Einen Beweis p für $SOL(x) \neq \emptyset$, der wieder aus einem Bitstring besteht.*

Der Verifier $\mathcal{V}(x, r, p)$ akzeptiert (gibt JA aus) oder verwirft (gibt NEIN aus) den Beweis in polynomieller Zeit. Dabei berechnet der Verifier die Antwort in zwei Phasen: Erst wird durch r ausgewählt, welcher Teil von p gelesen wird. (Die Zufallsbits adressieren also die Bits, die von p ausgelesen werden.) Dann wird an Hand von x und des gelesenen Teils von p JA oder NEIN ausgegeben.

Man kann die Anzahl der zufälligen Bits durch eine Funktion $r(|x|)$ beschränken und die Anzahl der von p eingelesenen Bits durch eine Funktion $p(|x|)$. Einen solchen Verifier nennt man $(r(n), p(n))$ beschränkt, wobei $n = |x|$.

Es ist kaum vorstellbar, daß ein Verifier sehr oft eine vernünftige Antwort ausgibt. Das liegt vor allem daran, daß man bei einem normalen Beweis für z.B. SAT kaum redundante Informationen hat. Die Wahrscheinlichkeit, daß ein Verifier einen Beweis verwirft ohne ihn komplett zu lesen, erscheint für große Instanzen gering.

Allerdings ist in der Definition nicht gesagt, wie der Beweis kodiert ist. Man kann sich jetzt vorstellen, daß man den Beweis redundant kodiert, so daß eventuelle Fehler leichter bemerkt werden können. Solche Beweise nennt man *Probabilistically Checkable Proofs* oder kurz PCPs [21, 7].

Mit Hilfe von PCPs kann man jetzt eine neue Klassifizierung von Entscheidungsproblemen definieren:

Definition: Klasse $PCP(r(n), p(n))$
Ein Entscheidungsproblem $\mathcal{P} = \langle \mathcal{I}, SOL \rangle$ gehört zur Klasse $PCP(r(n), p(n))$ wenn es einen $(r(n), p(n))$ beschränkten Verifier gibt, so daß für jede Instanz $x \in \mathcal{I}$ gilt:

1. *Wenn $SOL(x) \neq \emptyset$ dann gibt es einen Beweis p, bei dem der Verifier für jede mögliche Kombination an Zufallsbits JA ausgibt, d.h.*

$$Pr(\mathcal{V}(x, r, p) = \text{„}JA\text{“}) = 1$$

2. *Wenn $SOL(x) = \emptyset$, dann gilt für jeden Beweis p:*

$$Pr(\mathcal{V}(x, r, p) = \text{„}JA\text{“}) \leq \frac{1}{2}$$

Die Wahrscheinlichkeit $Pr(\cdot)$ wird dabei unter der Voraussetzung berechnet, daß alle Kombinationen an Zufallsbits gleich wahrscheinlich sind.

Es ergeben sich sofort einige Folgerungen:

$$\mathrm{P} = PCP(0, 0)$$

Der Verifier kann einfach die Lösung der Instanz in polynomieller Zeit berechnen und dann danach entscheiden. Es werden also keine Zufallsbits und kein Beweis gebraucht.

$$\mathrm{NP} = PCP(0, poly(n)) := \bigcup_{k \geq} PCP(0, n^k)$$

Da ein Beweis für eine positive Antwort für ein Problem aus NP immer in polynomieller Laufzeit überprüft werden kann, folgt daraus auch, daß die Länge des Beweises durch ein Polynom beschränkt ist. Der Verifier kann daher den gesamten Beweis einlesen und ihn dann laut der Definition von NP in polynomieller Laufzeit überprüfen; zufällige Bits werden ebenfalls nicht verwendet.

Jetzt ist die Frage, was passiert, wenn man wirklich Zufallsbits zuläßt und nicht mehr den gesamten Beweis einliest. Das erstaunliche Ergebnis lautet [6, 8, 28, 27]:

$$NP = PCP(O(\log(n), O(1))$$

Das heißt, daß der Verifier bereits mit einer konstanten Anzahl an Bits probabilistisch sinnvolle Entscheidungen treffen kann; (die Anzahl ist also unabhängig von der Länge der zu überprüfenden Instanz des Problems; tatsächlich ist sie sogar unabhängig von der Art des Problems!)

Die Beschränkung der gelesenen Zufallsbits auf $O(\log(n))$ besagt, daß man alle Positionen des Beweises auslesen kann; wenn man $k \log(n)$ zufällige Bits hat, dann gibt es nämlich $2^{k \log(n)} = n^k$ verschiedene Möglichkeiten für die Belegung der Bits; da die Länge des Beweises durch ein Polynom in der Länge der Instanz beschränkt ist, kann man also alle Positionen adressieren. Der lineare Faktor k hängt von der Art des Problems ab.

2.4 Nicht-Approximierbarkeit

Man kann die Wahrscheinlichkeiten, mit der ein Verifier Beweise akzeptiert, verallgemeinern. Die Wahrscheinlichkeit, mit der ein Verifier einen richtigen Beweis akzeptiert, nennt man *Completeness*; die Wahrscheinlichkeit mit der er einen falschen Beweis akzeptiert

Soundness. Man schreibt dann $PCP_s^c(r(n), p(n))$, wobei mit c die Completeness und mit s die Soundness gemeint ist.

Die Frage, wie groß die genaue Anzahl der eingelesenen Bits des Beweises, die Completeness und Soundness sein müssen, damit $PCP_s^c(O(\log(n)), O(1))$ noch NP charakterisiert spielt eine entscheidende Rolle für die nicht Approximierbarkeit von Problemen. Deshalb wurde nach der Entdeckung des PCP-Theorems versucht diese Werte zu verbessern [4, 5].

Håstad [26] gelang es zum ersten Mal ein optimales Ergebnis zu beweisen. Er zeigte, daß es für MAX-E3-SAT keinen Algorithmus mit einer besseren Performance als 8/7 geben kann, wenn $P \neq NP$. Gleichzeitig konnten Karloff und Zwick [29] zeigen, daß es einen polynomiellen (deterministischen) Algorithmus gibt, der MAX-E3-SAT mit einer Performance von 8/7 löst. Die untere Schranke von 8/7 ist damit die beste, die man beweisen kann und ist in diesem Sinne optimal.

Um hervorzuheben, wie wichtig das PCP-Theorem für Nicht-Approximierbarkeits-Beweise ist, soll mit der Charakterisierung von NP durch $PCP_{0.76}^1(O(\log(n)), 3)$ [26] eine (nicht optimale) untere Schranke für die Approximierbarkeit von MAX-E3-SAT gezeigt werden.

Durch die Charakterisierung ist es möglich, eine Reduktion zu beschreiben, die eine beliebige E3-SAT Instanz auf eine neue E3-SAT Instanz reduziert, bei der in unerfüllbaren Fällen mindestens ein konstanter Anteil der Klauseln unerfüllt ist. Diese Reduktion führt, wie in dem Kapitel über die Gap-Technik beschrieben, zu einer unteren Schranke für Approximierbarkeit.

Nehmen wir an, wir haben den $(O(\log(n)), 3)$ beschränkten Verifier, der Beweise von Instanzen aus E3-SAT überprüft.

Ohne Beschränkung der Allgemeinheit, können wir annehmen, daß der Verifier immer genau 3 Bits des Beweises erfragt, auch wenn er nicht alle davon benutzt.

Es sei eine Instanz x von E3-SAT und der dazugehörige Beweis p gegeben. Nehmen wir an, daß wir für jedes Bit im Beweis eine Variable $p[i]$ einführen, die den Wert dieses Bits erhält. Da der Verifier nur $k \log(|x|)$ Zufallsbits einliest, kann es auch nicht mehr als $2^{k \log(|x|)} = |x|^k$ verschiedene Variablen geben.

Für jeden möglichen Zufallsstring r liest der Verifier drei verschiedene Variablen ein, die mit $p[r_1]$, $p[r_2]$ und $p[r_3]$ bezeichnet werden.

Für einige 3-Tupel $(p[r_1], p[r_2], p[r_3])$ akzeptiert der Verifier und für andere nicht. Nehmen wir z.B. an, er verwirft den Beweis für die folgenden 3-Tupel (bei einem festgelegten Zufallsstring): $\{(0,0,0), (0,1,1), (1,0,0), (1,1,1)\}$.

Für jedes dieser Tupel konstruieren wir jetzt eine Klausel mit 3 Literalen, die genau dann erfüllt ist, wenn $(p[r_1], p[r_2], p[r_3])$ nicht den Wert des Tupels annimmt. Für das

angegebene Beispiel erhält man die vier Klauseln:

$$(p[r_1] \vee p[r_2] \vee p[r_3])$$
$$\left(p[r_1] \vee \overline{p[r_2]} \vee \overline{p[r_3]}\right)$$
$$\left(\overline{p[r_1]} \vee p[r_2] \vee p[r_3]\right)$$
$$\left(\overline{p[r_1]} \vee \overline{p[r_2]} \vee \overline{p[r_3]}\right)$$

Da es insgesamt nur 8 mögliche 3-Tupel gibt und nicht mehr als n^k verschiedene Zufallsstrings, erhalten wir eine neue boolesche Formel, die höchstens $8n^k$ Klauseln enthält.

Gibt es einen Beweis, bei dem der Verifier immer akzeptiert, dann ist die erzeugte Formel erfüllbar; die Belegung der Variablen entspricht genau den Bits dieses Beweises. Gibt es diesen Beweis nicht, dann müssen *bei jedem Beweis* mindestens 24 Prozent der Zufallsstrings r den Verifier zum Ablehnen bringen, d.h. für einen ablehnenden Zufallsstring muß mindestens eine der maximal 8 möglichen Klauseln nicht erfüllt sein.

Von maximal $8n^k$ Klauseln sind also bei unerfüllbaren Instanzen mindestens $0,24n^k$ Klauseln unerfüllbar. Mit Hilfe der Gap-Technik folgt daraus, daß sich MAX-E3-SAT nicht besser als 8/7.76 approximieren läßt.

2.5 Håstads optimale Ergebnisse

Am Schluß seien noch einmal alle Ergebnisse von Håstad [26] tabellarisch aufgelistet. Ergebnisse bei denen obere und untere Schranke zusammen fallen sind in dem Sinn optimal, daß sich nichts besseres beweisen läßt.

	obere Schranke		untere Schranke
	Konstante	Quelle	Konstante
E3-LIN2	2	Folklore	$2-\epsilon$
E3-LINp	p	Folklore	$p-\epsilon$
E3-LINΓ	$\lvert\Gamma\rvert$	Folklore	$\lvert\Gamma\rvert-\epsilon$
E2-LIN2	1.1383	[23]	$\frac{12}{11}-\epsilon$
E3-SAT	$\frac{8}{7}$	[29]	$\frac{8}{7}-\epsilon$
E2-SAT	1.0741	[20]	$\frac{22}{21}-\epsilon$
E4-Set Splitting	$\frac{8}{7}$	Folklore	$\frac{8}{7}-\epsilon$
MAX-CUT	1.1383	[23]	$\frac{17}{16}-\epsilon$
MAX-di-CUT	1.164	[23]	$\frac{12}{11}-\epsilon$
Vertex Cover	2	[22]	$\frac{7}{6}-\epsilon$

3 AP-Reduktionen und Gadgets

Will man beweisen, daß ein neues Entscheidungsproblem P_{neu} NP-vollständig ist, sucht man sich ein altes Entscheidungsproblem P_{alt}, von dem schon bekannt ist, daß es NP-vollständig ist; dann versucht man eine Karp-Reduktion von P_{alt} auf P_{neu} zu finden. Gelingt es einem zu zeigen, daß es eine Karp-Reduktion gibt, dann hat man damit auch gezeigt, daß P_{neu} NP-vollständig ist.

Im Falle von NP-vollständigkeits Beweisen reicht eine Reduktion auf ein einziges altes Problem, um zu zeigen, das ein neues Problem genau so schwer ist wie *alle* anderen NP-vollständigen Probleme.

Bei Optimierungsproblemen ist auf den ersten Blick nicht einmal klar, welche Art von Reduktion gebraucht wird. Obwohl es offensichtlich erscheint, daß ein Problem aus PTAS leichter ist als ein Problem aus APX $\setminus$ PTAS ist nicht klar, was für eine Reduktion dafür benötigt wird. Noch schwerer vorstellbar ist die Idee von APX-vollständigen Problemen, d.h. Probleme, mit deren Hilfe man alle Probleme aus APX beschreiben kann.

3.1 AP-Reduktionen

Für Approximationsuntersuchungen braucht man eine Reduktion, die nicht nur die Instanzen und Lösungen von zwei Problemen aufeinander transformiert, sondern auch noch die Qualität der Lösungen erhält.

Eine solche Reduktion nennt sich AP-Reduktion und ist wie folgt definiert.

Definition: AP-Reduktion
Seien P_{alt} und P_{neu} zwei Optimierungsprobleme aus NPO. P_{alt} heißt AP-reduzierbar auf P_{neu} in Zeichen $P_{alt} \leq_{AP} P_{neu}$, wenn es zwei Funktionen f und g und eine Konstante $\alpha > 0$ gibt, so daß gilt:

1. *Für jede Instanz $x_{alt} \in \mathcal{I}_{P_{alt}}$ und für jede reelle Zahl $\delta > 0$ gilt:*
   $x_{neu} = f(x_{alt}, \delta) \in \mathcal{I}_{P_{neu}}$.

2. *Für jede Instanz x_{alt} und für jede reelle Zahl $\delta > 0$ gilt:*
   $s_{neu} \in SOL(x_{neu}) \Longrightarrow s_{alt} = g(x_{alt}, s_{neu}, \delta) \in SOL(x_{alt})$

3. *f und g lassen sich für jede feste reelle Zahl $\delta > 0$ in polynomieller Zeit berechnen.*

4. *Für jede Instanz x_{alt}, für jedes $\delta > 0$ und für jedes s_{neu} gilt:*

$$\frac{1}{1+\delta} \leq \frac{val(x_{neu}, s_{neu})}{opt(x_{neu})} \leq 1+\delta \Longrightarrow \frac{1}{1+\alpha\delta} \leq \frac{val(x_{alt}, s_{alt})}{opt(x_{alt})} \leq 1+\alpha\delta$$

Die Definition stammt von Creszenzi, Kann, Silvestri und Trevisan [18] und ist zur Zeit die allgemein akzeptierte Beschreibung für Reduktionen zwischen Optimierungsproblemen.

Die Abkürzung *AP* steht für *Approximation Preserving*, also approximationserhaltend. Die Abkürzung hebt Punkt vier der Definition hervor, der garantiert, daß die Qualität einer beliebigen Lösung von I_{neu} bei der Transformation bis auf einen Faktor α erhalten bleibt. Das bedeutet auch, daß das Optimum von I_{neu} durch $g(I_{alt}, S_{neu}, \delta)$ auf das Optimum von I_{alt} abgebildet wird.

Existiert eine AP-Reduktion von einem Problem P_{alt} auf ein Problem P_{neu} so folgt daraus, daß P_{neu} mindestens so schwer ist wie P_{alt} oder anders herum, daß P_{alt} nicht schwerer ist als P_{neu}.

Mit Hilfe von AP-Reduktionen lassen sich deshalb zwei Dinge zeigen: Ist $P_{neu} \in \mathcal{APX}$ und ist $P_{alt} \leq_{AP} P_{neu}$, so ist auch $P_{alt} \in \mathcal{APX}$, weil man durch die AP-Reduktion einen Approximationsalgorithmus für P_{alt} erhält.

Auf der anderen Seite kann P_{neu} nicht zu $\mathcal{PTAS}$ gehören, wenn P_{alt} nicht zu $\mathcal{PTAS}$ gehört und $P_{alt} \leq_{AP} P_{neu}$.

Nehmen wir zum Beispiel an, daß sich ein Problem P_{neu} mit einem Faktor α auf $P_{alt} =$ MAX-3SAT reduzieren läßt und daß es einen Algorithmus A_{neu} gibt, der eine Performance von $1 + \delta$ hat. Durch die AP-Reduktion gilt:

$$\frac{val(I_{alt}, S_{alt})}{opt(I_{alt})} \geq \frac{1}{1 + \alpha\delta}$$

Håstad hat, wie schon vorher erwähnt, gezeigt, daß es NP-schwer ist, Instanzen von MAX-3SAT, die erfüllbar sind, von Instanzen, bei denen höchstens $(\frac{7}{8} + \epsilon)$ Klauseln erfüllbar sind, zu unterscheiden [26]. Das bedeutet:

$$\begin{aligned}
\frac{7}{8} &> \frac{1}{1 + \alpha\delta} \\
7 + 7\alpha\delta &> 8 \\
\delta &> \frac{1}{7\alpha} \\
\frac{1}{1 + \frac{1}{7\alpha}} &> \frac{1}{1 + \delta} \\
\frac{7\alpha}{7\alpha + 1} &> \frac{1}{1 + \delta}
\end{aligned}$$

Eine Besonderheit der AP-Reduktion ist die Abhängigkeit der Funktionen f und g von δ. Durch diese Definition wird es möglich, verschiedene Transformationen zu benutzen, je nachdem, wie gut die Performance der betrachteten Lösungen sein soll.

Auch wenn bei vielen Reduktionen diese Abhängigkeit nicht gebraucht wird, scheint es bis jetzt schwierig, APX-Vollständigkeit zu beweisen, ohne die Abhängigkeit der Funktionen f und g von δ zu benutzen.

3.2 L-Reduktionen

Bereits vor der Einführung von AP-Reduktionen versuchte man, Klassen von Optimierungsproblemen und dazugehörige vollständige Probleme zu finden. Papadimitriou und Yannakakis führten dazu in [34] die Klassen *MAX NP* und *MAX SNP* ein, die über die strukturellen Eigenschaften der zugehörigen Probleme definiert sind. Als Reduktion zwischen den Problemen wurde die L-Reduktion definiert. Sie hat den Vorteil, daß sie anders als die AP-Reduktion unabhängig von der Qualität der Lösung ist.

Definition: L-Reduktion
Es seien zwei Optimierungsprobleme P_{alt} und P_{neu} aus NPO gegeben. P_{alt} ist auf P_{neu} L-reduzierbar, (in Symbolen $P_{alt} \leq_L P_{neu}$) wenn es zwei Funktionen f und g gibt und zwei Konstanten $\beta, \gamma > 0$, so daß gilt:

1. *Eine Instanz $x_{alt} \in \mathcal{I}_{P_{alt}}$ wird durch f in polynomieller Laufzeit auf eine Instanz $x_{neu} = f(x_{alt}) \in \mathcal{I}_{P_{neu}}$ abgebildet.*
2. *$opt(x_{neu}) \leq \beta \cdot opt(x_{alt})$.*
3. *Eine Lösung $s_{neu} \in SOL(x_{neu})$ wird durch g in polynomieller Laufzeit auf eine Lösung $s_{alt} = g(x_{alt}, s_{neu}) \in SOL(x_{alt})$ abgebildet.*
4. *$|opt(x_{alt}) - val(x_{alt}, s_{alt})| \leq \gamma \cdot |opt(x_{neu}) - val(x_{neu}, s_{neu})|$*

L-Reduktionen sind im allgemeinen nicht so mächtig wie AP-Reduktionen, aber sie sind einfacher zu konstruieren. Um zu zeigen, daß es für Nicht-Approximierbarkeits-Beweise ausreicht, eine L-Reduktion zu finden, wird folgendes gezeigt: Gegeben sind zwei Probleme P_{alt} und P_{neu}, die beide zu APX gehören, und eine L-Reduktion, so daß $P_{alt} \leq_L P_{neu}$ gilt. In diesem Fall gibt es auch eine AP-Reduktion $P_{alt} \leq_{AP} P_{neu}$.

Beweis:
Wir bauen aus der L-Reduktion eine AP-Reduktion. Dazu müssen, je nachdem ob P_{alt} und P_{neu} Maximierungs oder Minimierungsprobleme sind, vier Fälle unterschieden werden.

Zuerst der einfache Fall, daß P_{alt} und P_{neu} beide Minimierungsprobleme sind; d.h. wenn eine Lösung s_{neu} eine Performance von $1 + \delta$ hat, ergibt sich:

$$\begin{aligned} \frac{val(x_{neu}, s_{neu})}{opt(x_{neu})} &\leq 1 + \delta \\ val(x_{neu}, s_{neu}) - opt(x_{neu}) &\leq \delta \cdot opt(x_{neu}) \end{aligned}$$

Damit es eine AP-Reduktion gibt, muß es ein festes α geben, so daß daraus für eine Lösung s_{alt} folgt:

$$\begin{aligned} \frac{val(x_{alt}, s_{alt})}{opt(x_{alt})} &\leq 1 + \alpha\delta \\ val(x_{alt}, s_{alt}) - opt(x_{alt}) &\leq \alpha\delta\, opt(x_{alt}) \end{aligned}$$

Aus $P_{alt} \leq_L P_{neu}$ folgt aber:

$$|opt(x_{alt}) - val(x_{alt}, s_{alt})| \leq \gamma \cdot |opt(x_{neu}) - val(x_{neu}, s_{neu})|$$

Weil es sich um Minimierungsprobleme handelt und die Performance von s_{neu} mindestens $1 + \delta$ beträgt, folgt daraus:

$$\begin{aligned} val(x_{alt}, s_{alt}) - opt(x_{alt}) &\leq \gamma \cdot (\delta \cdot opt(x_{neu})) \\ val(x_{alt}, s_{alt}) - opt(x_{alt}) &\leq \delta\gamma\beta \cdot opt(x_{alt}) \end{aligned}$$

Wählt man für $\alpha = \gamma\beta$, dann kann man f und g einfach von der L-Reduktion übernehmen und erhält dadurch eine korrekte AP-Reduktion.

Nehmen wir jetzt an, daß P_{alt} ein Maximierungsproblem und P_{neu} ein Minimierungsproblem ist. Eine AP-Reduktion läßt sich nur dann konstruieren, wenn wir die Tatsache benutzen, daß P_{alt} auch in APX liegt und es deshalb einen Algorithmus $\mathcal{A}$ gibt, der eine Lösung $\mathcal{A}(x_{alt})$ für P_{alt} in polynomieller Laufzeit findet, die eine garantierte Performance von $p > 1$ hat.

Für die AP-Reduktion $\langle f', g', \alpha \rangle$wird festgelegt:

- $f'(x_{alt}, \delta) = f(x_{alt})$, d.h. wir übernehmen die Transformation der L-Reduktion.
- $$g'(x_{alt}, s_{neu}, \delta) = \begin{cases} g(x_{alt}, s_{neu}) & \text{für } \delta \leq 1/(2\beta\gamma) \\ \mathcal{A}(x_{alt}) & \text{sonst} \end{cases}$$

  D.h. wir übernehmen g nur, wenn wir uns für eine Performance interessieren, die besser als $1 + 1/(2\beta\gamma)$ ist.
- Die Konstante α wird auf $2\beta\gamma \cdot p$ gesetzt; p meint dabei die Performance des Algorithmus $\mathcal{A}$

Mit dieser Festlegung lassen sich zwei Fälle unterscheiden. Im ersten Fall muß gelten: $\delta \leq 1/(2\beta\gamma)$. Es folgt aus der L-reduzierbarkeit:

$$\begin{aligned} opt(x_{alt}) - val(x_{alt}, s_{alt}) &\leq \gamma \cdot (val(x_{neu}, s_{neu}) - opt(x_{neu})) \\ val(x_{alt}, s_{alt}) &\geq opt(x_{alt}) - \gamma\delta \cdot opt(x_{neu}) \\ val(x_{alt}, s_{alt}) &\geq opt(x_{alt}) - \gamma\delta\beta \cdot opt(x_{alt}) \\ \frac{val(x_{alt}, s_{alt})}{opt(x_{alt})} &\geq 1 - \gamma\delta\beta \geq \frac{1}{1 + 2\beta\gamma\delta} \qquad \text{wegen } \delta < 1/(2\beta\gamma) \\ \frac{val(x_{alt}, s_{alt})}{opt(x_{alt})} &\geq \frac{1}{1 + p2\beta\gamma\delta} = \frac{1}{1 + \alpha\delta} \qquad \text{wegen } p > 1 \end{aligned}$$

Für den Fall, daß gilt $\delta > 1/(2\beta\gamma)$, folgt:

$$val(x_{alt}, s_{alt}) = val(x_{alt}, \mathcal{A}(x_{alt})) \geq \frac{1}{p} opt(x_{alt}) \geq \frac{1}{\alpha\delta} opt(x_{alt}) \geq \frac{1}{1+\alpha\delta} opt(x_{alt})$$

Die beschriebene AP-Reduktion ist also korrekt.

Die Beweise für die anderen beiden Fälle funktionieren ähnlich.

Will man von einem Problem P_{neu} ein Nicht-Approximierbarkeits-Ergebnis zeigen und weiß man, daß es in APX enthalten ist, dann kann man also eine L-Reduktion dazu verwenden.

3.3 Gadgets

Nachdem erklärt wurde, was für Eigenschaften die Reduktionen haben müssen, wenn man Probleme aus APX aufeinander reduzieren will, stellt sich natürlich die Frage, wie man solche Reduktionen konstruiert.

Für eine bestimmte Art von Reduktionen, die auf lokalen Ersetzungen beruhen, hat man dieses Problem vollständig gelöst. Diese Reduktionen beziehen sich auf Probleme aus MAX $\mathcal{F}$.

Probleme dieser Klasse haben zuerst einmal den Vorteil, daß sie in APX liegen. Um das einzusehen, stellt man sich vor, daß man eine Familie $\mathcal{F}$ hat, die nur aus Constraint Funktionen mit k Parametern bestehen. Es gibt höchstens 2^k verschiedene Möglichkeiten, die Eingabeparameter mit Werten zu belegen. Wenn man ausschließt, daß $\mathcal{F}$ eine Funktion enthält, die äquivalent zu 0 ist, ergeben sich daraus zwei Konsequenzen:

1. Ein randomisierter Algorithmus, der einfach allen Variablen zufällig eine 0 oder 1 zuweist, hat eine erwartete Performance von mindestens 2^k.

   Wenn nämlich $\mathcal{F}$ keine Null-Funktion enthält, dann gibt es für eine Funktion aus $\mathcal{F}$ zumindest eine von 2^k möglichen Belegungen für die Parameter, bei der man eine 1 als Ergebnis erhält. Hat man n Constraints, die alle simultan erfüllt werden können, dann wird der randomisierte Algorithmus im Schnitt mindestens $n/2^k$ davon erfüllen.

   Daraus folgt, daß jedes Problem aus MAX $\mathcal{F}$ in APX liegt.

2. Wenn es n Constraints gibt, dann gibt es auf jeden Fall $n/2^k$ Constraints, die simultan erfüllt werden können, d.h. $n/2^k$ ist außerdem eine untere Schranke für das Optimum.

Der andere Vorteil an CSPs ist, daß man relativ leicht Reduktionen zwischen zwei Problemen bauen kann. Eine Reduktion, um ein CSP auf ein anderes zu reduzieren, läßt sich

am einfachsten dadurch konstruieren, daß man ein Constraint im Ausgangsproblem durch mehrere Constraints im Zielproblem ersetzt.

Nehmen wir an, wir wollen das MAX-E3-LIN2 Problem $\mathcal{P}_{alt} = \langle \mathcal{I}_{alt}, SOL_{alt} \rangle$ auf das MAX-E2-LIN2 Problem $\mathcal{P}_{neu} = \langle \mathcal{I}_{neu}, SOL_{neu} \rangle$ reduzieren. Im weiteren werden wieder die gleichen Variablen wie immer benutzt, d.h. $x_{alt,neu} \in \mathcal{I}_{alt,neu}$ und $s_{alt,neu} \in SOL_{alt,neu}(x_{alt,neu})$.

Die für eine L-Reduktion nötigen Ersetzungen sehen wie in Abbildung 1 aus.

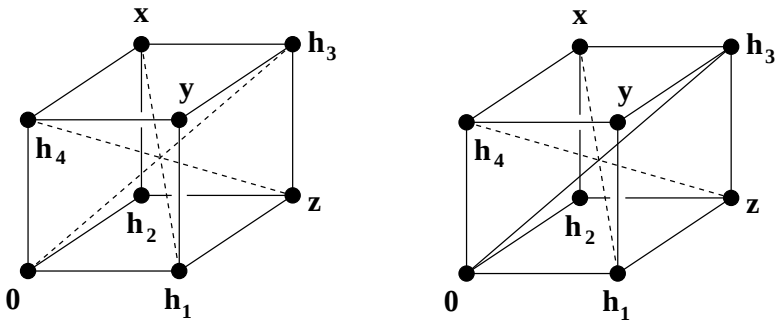


Abbildung 1: Die Ersetzungen für MAX-E3-LIN2 $\leq_L$ MAX-E2-LN2

Jede Gleichung der Form $x \oplus y \oplus z = 0$ oder $x \oplus y \oplus z = 1$ aus MAX-E3-LIN2 wird durch 16 Gleichungen im MAX-E2-LIN2 ersetzt. Jede durchgezogene Kante entspricht dabei einer Gleichung der Form $u \oplus v = 1$ und jede gestrichelte Kante einer Gleichung der Form $u \oplus v = 0$. Bei der Ersetzung werden für jede ursprüngliche Gleichung aus MAX-E3-LIN2 jeweils vier zusätzliche Hilfsvariablen $h_1, \ldots, h_4$ eingeführt.

Wenn eine Gleichung der Form $x \oplus y \oplus z = 0$ im E3-LIN2 durch eine Belegung erfüllt ist, dann kann man die Hilfsvariablen so belegen, daß von den 16 korrespondierenden Gleichungen im E2-LIN2 12 erfüllt sind.

Ist dagegen eine Gleichung der Form $x \oplus y \oplus z = 0$ nicht erfüllt, dann kann man von den 16 korrespondierenden Gleichungen nur 10 erfüllen.

Für Gleichungen der Form $x \oplus y \oplus z = 1$ gilt das Selbe.

Das bedeutet, daß für das Optimum $opt(x_{neu})$ des E2-LIN2 gilt:
$opt(x_{neu}) = 10 \cdot |x_{alt}| + 2 \cdot opt(x_{alt})$. Dabei ist mit $|x_{alt}|$ die Anzahl der Gleichungen im E3-LIN2 gemeint.

Um eine Belegung s_{neu} des E2-LIN2 in eine Belegung des E3-LIN2 s_{alt} zu transformieren, muß man sich bloß die Variablen der Belegung heraus nehmen, die auch im E3-LIN2 vorkommen. Dabei gilt:
$val(x_{neu}, s_{neu}) \leq 10 \cdot |x_{alt}| + 2 \cdot val(x_{alt}, s_{alt})$

Für eine L-Reduktion muß noch gezeigt werden:
$opt(x_{neu}) \leq \beta \cdot opt(x_{alt})$

Dazu berechnen wir die Konstante β, die verwendet werden muß, damit die Ungleichung erfüllt ist. In einem E3-LIN2 hat jedes Constraint genau 3 Parameter; deshalb gilt auf jeden Fall $opt(x_{alt}) \geq |x_{alt}|/2^3$. Daher muß gelten:

$$\begin{aligned} opt(x_{neu}) &\leq \beta \cdot opt(x_{alt}) \\ 10 \cdot |x_{alt}| + 2 \cdot opt(x_{alt}) &\leq \beta \cdot opt(x_{alt}) \\ \beta &\geq \frac{10 \cdot |x_{alt}|}{opt(x_{alt})} + 2 \\ \beta &\geq \frac{10 \cdot |x_{alt}|}{\frac{|x_{alt}|}{2^3}} + 2 \\ \beta &\geq 82 \end{aligned}$$

Wählen wir also $\beta = 82$, dann ist die Bedingung $opt(x_{neu}) \leq \beta \cdot opt(x_{alt})$ immer erfüllt.

Weiterhin gilt:

$$\begin{aligned} val(x_{neu}, s_{neu}) &\leq 10 \cdot |x_{alt}| + 2 \cdot val(x_{alt}, s_{alt}) \\ opt(x_{neu}) - 10 \cdot |x_{alt}| - 2 \cdot val(x_{alt}, s_{alt}) &\leq opt(x_{neu}) - val(x_{neu}, s_{neu}) \\ 2 \cdot opt(x_{alt}) - 2 \cdot val(x_{alt}, s_{alt}) &\leq opt(x_{neu}) - val(x_{neu}, s_{neu}) \\ opt(x_{alt}) - val(x_{alt}, s_{alt}) &\leq \frac{1}{2}\left(opt(x_{neu}) - val(x_{neu}, s_{neu})\right) \end{aligned}$$

Alle Bedingungen für eine L-Reduktion sind damit erfüllt und es folgt MAX-E3-LIN2 $\leq_L$ MAX-E2-LIN2.

Die Methode, einzelne Constraints in einer Instanz des Problems P_{alt} aus MAX $\mathcal{F}$ durch mehrere Constraints einer Instanz des Problems P_{neu} zu ersetzen, läßt sich formalisieren.

Dazu betrachtet man eine Ersetzung, die eine Constraint-Funktion f auf mehrere Constraints aus der Constraint-Familie $\mathcal{F}$ abbildet:

Definition: α-Gadget [11]
Es sei $\alpha \in R^+$, eine Constraint Funktion $f : \{0,1\}^k \to \{0,1\}$, und eine Constraint-Familie $\mathcal{F}$ gegeben. Ein α-Gadget, das f auf $\mathcal{F}$ reduziert, besteht aus einer Menge an Hilfsvariablen $y_1, \ldots, y_n$, einer endlichen Menge von reellen Gewichten $w_j \geq 0$ und den damit gewichteten Constraints C_j auf $\mathcal{F}$ über den Primärvariablen $x_1, \ldots, x_k$ und den Hilfsvariablen $y_1, \ldots, y_n$. Ein α-Gadget erfüllt für jede Belegung $\vec{x}$ für $x_1, \ldots, x_k$ und $\vec{y}$ für $y_1, \ldots, y_n$ folgende Eigenschaften:

$$(\forall \vec{x} : f(\vec{x}) = 1)(\forall \vec{y}) \quad : \quad \sum_j w_j C_j(\vec{x}, \vec{y}) \leq \alpha$$

$$(\forall \vec{x} : f(\vec{x}) = 1)(\exists \vec{y}) \quad : \quad \sum_j w_j C_j(\vec{x}, \vec{y}) = \alpha$$
$$(\forall \vec{x} : f(\vec{x}) = 0)(\forall \vec{y}) \quad : \quad \sum_j w_j C_j(\vec{x}, \vec{y}) \leq \alpha - 1$$

Das α-Gadget heißt strikt, wenn außerdem gilt:

$$(\forall \vec{x} : f(\vec{x}) = 0)(\exists \vec{y}) \quad : \quad \sum_j w_j C_j(\vec{x}, \vec{y}) = \alpha - 1$$

Wenn man bei einem Gadget alle Gewichte für die Constraints auf 1 setzt, dann erhält man ein ungewichtetes Gadget.

Nehmen wir an, es gibt ein striktes, ungewichtetes α-Gadget, das eine Funktion h mit k Parametern auf eine Constraint Familie $\mathcal{F}$ abbildet. In diesem Fall gilt:

$$\text{MAX}\{h\} \leq_L \text{MAX}\mathcal{F}$$

Beweis:
Nehmen wir an, daß x_{alt} eine Instanz von MAX $\{h\}$ ist. Es sei $x_{neu} = f(x_{alt})$ die Instanz aus MAX $\mathcal{F}$, die durch Anwendung des α-Gadgets auf alle Constraints aus x_{alt} entsteht. Sei $s_{neu} \in SOL(x_{neu})$; eine Lösung s_{alt} erhält man, indem man sich aus der Lösung s_{neu} alle Werte für die Variablen heraus nimmt, die auch in s_{alt} vorkommen. Es sei $|x_{alt}|$ die Anzahl der Constraints in x_{alt}. Aus der Definition eines strikten α-Gadgets folgt:

$$\begin{aligned} opt(x_{neu}) &= (\alpha - 1)|x_{alt}| + opt(x_{alt}) \\ val(x_{neu}, s_{neu}) &\leq (\alpha - 1)|x_{alt}| + val(x_{alt}, s_{alt}) \end{aligned}$$

Mit der gleichen Argumentation, wie bei dem Beweis für MAX-E3-LIN2 $\leq_L$ MAX-E2-LIN2, kann man aus diesen beiden Ungleichungen die obige Behauptung zeigen.

Mit Gadgets, kann man auch leicht Nicht-Approximierbarkeits Beweise führen. Nehmen wir an, ein MAX-E3-LIN2 hat n Gleichungen; Håstad hat bewiesen [26], daß sich MAX-E3-LIN2 Instanzen, bei denen $n - \epsilon$ Gleichungen erfüllbar sind, nicht von Instanzen zu unterscheiden sind, bei denen nur $n/2 + \epsilon$ Gleichungen erfüllbar sind. Daraus folgt:

Gibt es ein ungewichtetes α-Gadget, das MAX-E3-LIN2 auf ein neues Problem $\mathcal{P}$ aus MAX $\mathcal{F}$ reduziert, dann folgt: Es gibt keinen polynomiellen Approximationsalgorithmus für $\mathcal{P}$, der eine bessere Performance als $\frac{2\alpha}{2\alpha-1} - \epsilon$ erzielt.

Beweis:
Nehmen wir an, wir hätten einen polynomiellen Algorithmus für P, der eine bessere Performance als $\frac{2\alpha}{2\alpha-1}$ erzielt. Gegeben sei nun eine Instanz mit n Gleichungen aus MAX-E3-LIN2, von denen sich $n - \epsilon$ Gleichungen erfüllen lassen. Durch das Gadget gibt es in

der Instanz von $\mathcal{P}$ mindestens $\alpha(n-\epsilon)$ erfüllbare Constraints. Wenn der Algorithmus s_{neu} Constraints erfüllt, dann gilt für die Anzahl der erfüllten Gleichungen s_{alt} in MAX-E3-LIN2:

$$s_{neu} \leq \alpha s_{alt} + (n - s_{alt})(\alpha - 1) = \alpha n - n + s_{alt}$$

Aus der Annahme über die Performance des Algorithmus wird jetzt ein Widerspruch abgeleitet; weil es sich um ein Maximierungs Problem handelt, gilt:

$$\begin{aligned} \frac{s_{neu}}{\alpha(n-\epsilon)} &> \frac{2\alpha - 1}{2\alpha} \\ 2\alpha(\alpha n - n + s_{alt}) \geq 2\alpha s_{neu} &> 2\alpha^2 n - 2\alpha^2\epsilon - \alpha n + \alpha\epsilon \\ 2\alpha s_{alt} &> \alpha n - \alpha\epsilon(2\alpha - 1) \\ \frac{s_{alt}}{n} &> \frac{1}{2} - \epsilon\frac{2\alpha - 1}{n} \end{aligned}$$

Da ϵ beliebig klein werden kann, erhält man für MAX-E3-LIN2 eine Lösung, bei der mehr als $n/2$ Gleichungen erfüllt sind. Damit weiß man aber, daß es sich um keine Instanz handeln kann, bei der nur $n/2$ Gleichungen erfüllbar sind; damit kann man ein NP-schweres Problem lösen, und das ist ein Widerspruch zu NP $\neq$ P .

Wie man sieht, ist es entscheidend, Gadgets zu finden, die ein möglichst kleines α haben. Glücklicherweise wurde entdeckt [36], wie man (beweisbar) optimale Gadgets mit Hilfe von Linearer Programmierung konstruieren kann.

Leider funktioniert dieser Ansatz nur bei Problemen aus MAX $\mathcal{F}$, bei denen durch eine lokale Ersetzung – wie sie durch ein Gadget beschrieben wird – die Konstruktion einer Reduktion möglich ist.

Hat man Probleme, bei denen auch globale Bedingungen erfüllt sein müssen oder bei denen andere, nicht durch Constraints beschreibbare Einschränkungen gelten, dann gestaltet sich die Suche nach einer guten Reduktion immer noch schwierig.

Eine bestimmte Art von solchen Problemen wird im weiteren besprochen.

4 Expander und randomisierte Reduktionen

Besondere Schwierigkeiten entstehen, wenn man die Nicht-Approximierbarkeit von Problemen aus *Max F* beweisen will, bei denen die Anzahl der Vorkommen einer Variablen beschränkt sind.

Die gängigen Beweise benutzen eine Reduktion des unbeschränkten Problems auf die beschränkte Variante. Die Reduktion bildet eine Variable X im unbeschränkten Fall auf mehrere Variablen im beschränkten Fall ab; eine Variable, die n-mal vorkommt wird auf n Variablen X_1, ..., X_n abgebildet. Jede der X_i Variablen ist nur noch an einem Constraint im ursprünglichen Problem beteiligt.

Damit eine Belegung der X_i Variablen in eine Belegung für das ursprüngliche Problem transformiert werden kann, muß sichergestellt werden, daß alle X_i Variablen die gleiche Belegung haben.

Für den Beweis, daß der beschränkte Fall ebenso wie der unbeschränkte Fall NP-vollständig ist – wenn es um die Frage der Erfüllbarkeit geht – genügt es, $n-1$ zusätzliche Bedingungen einzuführen, nämlich $X_1 = X_2$, ..., $X_{n-1} = X_n$; dadurch wird sichergestellt, daß alle X_i den gleichen Wert haben. Als Graph kann man das wie in Abbildung 2 darstellen; jeder Knoten entspricht einer Variablen und jede Kante einer Gleichheitsbedingung.

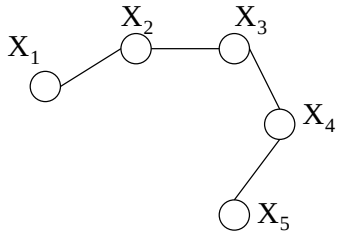


Abbildung 2: Vier Gleichheitsbedingungen bei 5 Variablen

Wenn man allerdings ein Maximierungs-Problem betrachtet, dann ist die Einführung dieser Bedingungen nicht ausreichend, denn eventuell kann man die Anzahl der erfüllten Bedingungen maximieren, indem man nur wenige der Gleichheitsbedingungen bricht und dadurch viele andere Bedingungen erfüllt. Das bedeutet gleichzeitig, daß eine Umbelegung der Variablen in eine gültige Belegung (d.h. alle X_i bekommen den gleichen Wert) die Anzahl der erfüllten Bedingungen verringert und deshalb durch Umbelegung keine gültige AP-Reduktion mehr möglich ist.

Um die gewünschte Gleichheit der X_i zu erzwingen, muß man zusätzliche Gleichheitsbedingungen einführen, die bewirken, daß eine optimale Belegung immer alle X_i auf den gleichen Wert setzt. Man könnte zum Beispiel als Bedingungen $X_i = X_j, i \neq j$ verwenden,

also einen vollständigen Graph. Leider ist dann die Anzahl der Vorkommen einer Variablen nicht mehr beschränkt, und zusätzlich ist auch noch die Anzahl der Bedingungen im beschränkten Fall nicht mehr linear abhängig von der Anzahl der Bedingungen im unbeschränkten Fall.

4.1 Expander Graphen

Was gebraucht wird ist ein d-regulärer Graph (V, E), der folgende Eigenschaft hat:

Definition: Expander Graph
Es sei $S \subset V$ *und* $\mathrm{Cut}(S)$ *die Anzahl der Kanten, die* S *mit* $V \setminus S$ *verbinden; eine dieser Kanten wird Cut-Kante genannt. Dann muß für jedes* S *mit* $|S| \leq |V|/2$ *gelten* $\mathrm{Cut}(S) > |S|$.

Graphen mit dieser Eigenschaft werden in der Literatur oft als *Expander* bezeichnet. Meistens wird allerdings nicht die Anzahl Cut-Kanten betrachtet, sondern die Größe der Nachbarschaft von S; d.h. die Knoten, die nicht zu S gehören, aber durch eine Kante mit S verbunden sind.

Benutzt man die Kanten eines solchen Expanders als Gleichheitsbedingungen, dann kann man jede Belegung der X_i in eine Belegung verwandeln, bei der alle X_i den gleichen Wert haben, ohne die Anzahl der erfüllten Bedingungen zu verringern; es werden einfach alle X_i auf den Wert gesetzt, den die Mehrzahl der X_i besitzt. Werden m Variablen umbelegt, dann werden durch die Expandereigenschaft auch mindestens m Gleichheitsbedingungen zusätzlich erfüllt. Da jedes X_i aber an nur einer Bedingung des ursprünglichen Problems beteiligt ist, werden nie mehr als m Bedingungen zusätzlich gebrochen, die Anzahl der erfüllten Bedingungen kann also nur steigen. Daraus folgt, daß bei jeder optimalen Belegung alle X_i den gleichen Wert haben. Durch die d-Regularität des Graphen ist außerdem sichergestellt, daß jedes X_i an genau $d + 1$ Bedingungen beteiligt ist und daß es im beschränkten Fall nicht mehr als $(1 + d/2)$ mal so viele Bedingungen gibt wie im unbeschränkten Fall.

4.2 Reduktionen mit randomisierten Expandern

Leider ist bis jetzt die deterministische Konstruktion von Expandern extrem schwer, besonders wenn d klein ist. Das Hauptproblem besteht darin zu beweisen, daß ein deterministisch erzeugter Graph wirklich ein Expander ist. Meistens wird dazu der zweite Eigenwert des Graphen betrachtet. Dieser Wert steht in enger Verbindung zu den Expandereigenschaften des Graphen [3, 2]. Die bisher besten Konstruktionen versuchen diesen Wert zu maximieren [32, 33]. Eine rekursive Konstruktion, die ohne den zweiten Eigenwert des Graphen auskommt, wird von Ajtai beschrieben [1].

Interessanterweise läßt sich zeigen, daß ein zufällig erzeugter d-regulärer Graph mit hoher Wahrscheinlichkeit Expandereigenschaften besitzt. Diese Tatsache läßt sich ausnutzen um randomisierte Reduktionen zu konstruieren.

Es ist noch die Frage zu klären, *wie* hoch die Wahrscheinlichkeit sein muß, daß ein zufällig erzeugter d-regulärer Graph ein Expander ist.

Es sei n die Anzahl der Expander die gebraucht wird (das entspricht der Anzahl der Variablen im ursprünglichen Problem). Es sei p die Wahrscheinlichkeit, daß ein zufällig erzeugter d-regulärer Graph *kein* Expander ist. Für Die Wahrscheinlichkeit, daß alle n Graphen Expander sind, ergibt sich (mit Hilfe der Bernoullischen Ungleichung):

$$(1-p)^n \geq 1 - np$$

Damit die Reduktion korrekt ist, muß diese Wahrscheinlichkeit größer $1/2$ sein. Daraus folgt:

$$\begin{aligned} 1 - np &\geq \frac{1}{2} \\ \frac{1}{2n} &\geq p \end{aligned}$$

Das heißt, die Wahrscheinlichkeit p, daß ein Graph *kein* Expander ist, muß mindestens mit $O(1/n)$ gegen Null gehen, damit die Reduktion korrekt ist.

4.3 Konstruktion von randomisierten Expander

Als nächstes soll untersucht werden, wie groß d bei einem zufällig erzeugten d-regulären Graphen sein muß, damit er mit der gewünschten Wahrscheinlichkeit ein Expander ist.

Bollobas hat diese Fragestellung untersucht [15]. Dabei liegt folgende Idee zugrunde: Um einen d-regulären Graphen mit n Knoten zu erzeugen, wird ein Graph mit dn Knoten benutzt, d.h. jeder der n Knoten wird auf d Knoten aufgeblasen. Auf die dn Knoten wird zufällig ein Matching gelegt, das bei Bollobas als *Konfiguration* bezeichnet wird. Danach wird der Graph wieder kontraktiert, so daß man die gewünschten n Knoten hat. Bei der Kontraktion kann natürlich ein Graph entstehen, der Schleifen oder doppelte Kanten enthält. Bollobas [14] war allerdings in der Lage zu zeigen, daß die Wahrscheinlichkeit, daß es sich nach der Kontraktion um einen gültigen Graphen handelt, größer als Null ist, selbst wenn n gegen unendlich geht.

Wegen dieser Tatsache genügt es, die Wahrscheinlichkeit zu berechnen, mit der eine *Konfiguration* die gewünschte Expandereigenschaft *nicht* besitzt. Für die Abschätzung dieser Wahrscheinlichkeit werden ein paar mathematische Hilfsmittel benötigt, die im folgenden näher erläutert werden.

4.4 Mathematische Hilfsmittel

Als erstes braucht man die Anzahl der perfekten Matchings $M(n)$, die auf n Knoten geworfen werden können. Für gerade n ergibt sich:

$$M(n) = \frac{n!}{\left(\frac{n}{2}\right)!2^{\frac{n}{2}}}$$

Beweis durch Induktion:

- Für $n = 2$ gilt die Formel, wie man leicht nachrechnen kann.
- Nehmen wir an, es gibt n Knoten und man wählt sich einen Knoten aus.
- Diesen Knoten kann man mit $n - 1$ verschiedenen Knoten verbinden.
- Auf die übrigen $n - 2$ Knoten kann man ein Matching auf $M(n - 2)$ verschiedene Arten werfen.
- Daraus folgt:

$$(n-1)M(n-2) = \frac{(n-1)(n-2)!}{\left(\frac{n-2}{2}\right)!2^{\frac{n-2}{2}}} = \frac{(n-1)!}{\left(\frac{n}{2}-1\right)!2^{\frac{n}{2}-1}} = \frac{\frac{n}{2}n!2}{n\left(\frac{n}{2}\right)!2^{\frac{n}{2}}} = M(n)$$

Für die kombinatorischen Berechnungen braucht man außerdem Abschätzungen, die nur die exponentiellen Anteile berücksichtigt. Die Abschätzungen beruhen auf der Stirlingschen Formel:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n O\left(1 + \frac{12}{n}\right)$$

Wenn man die Formel in eine e-Funktion umwandelt, bekommt man:

$$n = e^{n\ln n - n + O(\ln n)}$$

Der Fehlerterm $O(\ln n)$ kann bei großen n vernachlässigt werden; er liefert nur einen logarithmischen Beitrag, der aber für $n \to \infty$ von den anderen Summanden völlig überlagert wird.

Für $M(n)$ ergibt sich:

$$M(n) = \frac{n!}{\frac{n}{2}!2^{\frac{n}{2}}} = e^{T+O(\ln n)}$$

$$\begin{aligned} T &= n\ln n - n + \frac{n}{2} - \frac{n}{2}\ln\frac{n}{2} - \frac{n}{2}\ln 2 = n\ln n - \frac{n}{2} - \frac{n}{2}\ln n - \frac{n}{2}\ln\frac{1}{2} + \frac{n}{2}\ln\frac{1}{2} \\ &= \frac{n}{2}\ln n - \frac{n}{2} \end{aligned}$$

Für Binomialkoeffizienten ergibt sich:

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{k!(n-k!)} = e^{T+O(\ln n)} \\ T &= n\ln n - n + k - k\ln k + (n-k) - (n-k)\ln(n-k) \\ &= n\ln n - k\ln k - (n-k)\ln(n-k)\end{aligned}$$

Diese Formeln helfen noch nichts, wenn man wissen will, was für den Fall $n \to \infty$ passiert. Dazu benutzt man folgende Darstellung:

$$\begin{aligned}\binom{\alpha n}{\beta n} &= e^{T+O(\ln n)} \\ T &= \alpha n \ln \alpha n - \beta n \ln \beta n - (\alpha-\beta)n\ln([\alpha-\beta]n)\end{aligned}$$

Jetzt zerlegt man T in Anteile, die $n\ln n$ und n als Faktoren enthalten und erhält:

$$\begin{aligned}T &= n\ln n(\alpha - \beta - (\alpha-\beta)) + n(\alpha\ln\alpha - \beta\ln\beta - (\alpha-\beta)\ln(\alpha-\beta) \\ &= n[\alpha\ln\alpha - \beta\ln\beta - (\alpha-\beta)\ln(\alpha-\beta)]\end{aligned}$$

Das heißt, daß sich bei vorgegebenen α und β der Binomialkoeffizient für große n wie eine e-Funktion der Form e^{kn} verhält, wobei k nur von α und β abhängt.

Zusätzlich wird noch eine Abschätzung für folgenden Ausdruck benötigt:

$$(\alpha n)!\frac{M(\beta n - \alpha n)M(dn - \beta n - \alpha n)}{M(dn)} = e^{T+O(\ln n)}$$

$$\begin{aligned}T &= \alpha n\ln\alpha n - \alpha n + \frac{\beta n - \alpha n}{2}\ln(\beta n - \alpha n) - \frac{\beta n - \alpha n}{2} \\ &\quad + \frac{dn - \beta n - \alpha n}{2}\ln(dn - \beta n - \alpha n) - \frac{dn - \beta n - \alpha n}{2} + \frac{dn}{2} - \frac{dn}{2}\ln dn \\ &= n\ln n\left[\alpha + \frac{\beta}{2} - \frac{\alpha}{2} + \frac{d}{2} - \frac{\beta}{2} - \frac{\alpha}{2} - \frac{d}{2}\right] \\ &\quad + n\left[\alpha\ln\alpha + \frac{\beta-\alpha}{2}\ln(\beta-\alpha) + \frac{d-\beta-\alpha}{2}\ln(d-\beta-\alpha) - \frac{d}{2}\ln d\right] \\ &= n\left[\alpha\ln\alpha + \frac{\beta-\alpha}{2}\ln(\beta-\alpha) + \frac{d-\beta-\alpha}{2}\ln(d-\beta-\alpha) - \frac{d}{2}\ln d\right]\end{aligned}$$

4.5 6-reguläre Expander

Jetzt soll mit Hilfe der oben ausgearbeiteten Formeln gezeigt werden, daß ein 6-regulärer zufällig erzeugter Graph mit hoher Wahrscheinlichkeit ein Expander ist. Dazu wird wieder das Modell von Bollobas verwendet.

Nehmen wir an, daß eine Menge mit s Knoten in einem Graphen mit n Knoten ausgewählt ist. Die Wahrscheinlichkeit, daß diese Menge genau i Cut-Kanten hat, ist:

$$\begin{aligned} P(s,i) &= \binom{ds}{i}\binom{dn-ds}{i} i! \frac{M(ds-i)M(dn-ds-i)}{M(dn)} \\ &= \frac{(ds)!}{i!}\frac{(dn-ds)!}{\left(\frac{ds-i}{2}\right)!}\frac{2^i}{\left(\frac{dn-ds-i}{2}\right)!}\frac{\left(\frac{dn}{2}\right)}{(dn)!} \end{aligned}$$

Begründung:
Durch die s ausgewählten Knoten im Graphen werden ds Knoten in der Konfiguration ausgewählt; $dn - ds$ Knoten sind unausgewählt. Es müssen jeweils i Endpunkte für die Cut-Kanten in den ausgewählten und unausgewählten Knoten bestimmt werden. Die Endpunkte kann man auf $i!$ Arten verbinden. Auf die restlichen $ds - i$ ausgewählten Knoten und auf die restlichen $dn - ds - i$ unausgewählten Knoten muß jeweils ein Matching geworfen werden. Insgesamt gibt es $M(dn)$ Möglichkeiten ein Matching zu werfen.

Nennen wir eine Menge, die weniger Cut-Kanten als Knoten hat „Bad-Set“. Die Wahrscheinlichkeit, daß eine fest ausgewählte Menge mit s Knoten ein Bad-Set ist, ergibt sich als:

$$P(s) = \sum_{i=0}^{s-1} P(s,i)$$

$P(s,i)$ steigt mit i monoton an, denn für $i \le s \le n/2$ und $d \ge 3$ gilt:

$$\begin{aligned} \frac{P(s,i)}{P(s,i-2)} &= \frac{\left(\frac{ds-i}{2}+1\right)\left(\frac{dn-ds-i}{2}+1\right)2^2}{i(i-1)} = \frac{(ds-i+2)(dn-ds-i+2)}{i(i-1)} \\ &\ge \frac{(ds-s+2)(dn-ds-s+2)}{s(s-1)} > 1 \end{aligned}$$

Daraus folgt:

$$P(s) = \sum_{i=0}^{s-1} P(s,i) \le sP(s,s)$$

Die Wahrscheinlichkeit, daß ein Bad-Set mit s Knoten existiert ergibt sich deshalb als:

$$H(s) = \binom{n}{s} s\, P(s,s)$$

Die Wahrscheinlichkeit, daß überhaupt ein Bad-Set existiert und somit der Graph *nicht* die gewünschten Expandereigenschaften hat, ist:

$$H = \sum_{s=1}^{n/2} H(s) \leq \frac{n}{2} \max\left\{H(s) : 1 \leq s \leq \frac{n}{2}\right\}$$

Damit der zufällig erzeugte d-reguläre Graph für eine randomisierte Reduktion geeignet ist, muß also das Maximum im obigen Ausdruck schneller als $O(1/n^2)$ gegen Null gehen.

Um das Maximum abzuschätzen, setzt man $s = \alpha n$. d.h. man muß das Maximum des Ausdrucks

$$\begin{aligned} H(\alpha n) &= \binom{n}{\alpha n} \alpha n \binom{d\alpha n}{\alpha n} \binom{(d-d\alpha)n}{\alpha n} \\ &\quad (\alpha n)! \frac{M((d\alpha-\alpha)n) M((d-d\alpha-\alpha)n)}{M(dn)} \\ &= e^{n \cdot F(\alpha,d) + O(\ln n)} \end{aligned}$$

bestimmen. Der Faktor αn geht im Exponenten mit $O(\ln n)$ ein; für den Rest benutzt man die oben ausgearbeiteten Formeln und erhält:

$$\begin{aligned} F(\alpha,d) = &+ 1 \ln 1 - \alpha \ln \alpha - (1-\alpha)\ln(1-\alpha) \\ &+ (d\alpha)\ln(d\alpha) - \alpha \ln \alpha - (d\alpha-\alpha)\ln(d\alpha-\alpha) \\ &+ (d-d\alpha)\ln(d-d\alpha) - \alpha \ln \alpha - (d-d\alpha-\alpha)\ln(d-d\alpha-\alpha) \\ &+ \alpha \ln \alpha + 0.5(d\alpha-\alpha)\ln(d\alpha-\alpha) \\ &\quad + 0.5(d-d\alpha-\alpha)\ln(d-d\alpha-\alpha) - 0.5d \ln d \end{aligned}$$

Die Funktion $F(\alpha, 6)$ sieht wie in Abbildung 3 gezeigt aus, wobei $F(0.5, 6) \leq -0.034$ ist. Das bedeutet, die Wahrscheinlichkeit der Existenz eines Bad-Sets mit mehr als $0.01n$ Knoten bei einem zufällig erzeugten 6-regulären Graphen geht mit mindestens $O(e^{-0.034n})$ gegen Null; das reicht für eine randomisierte Reduktion leicht aus.

Was noch untersucht werden muß, ist der Rand, wenn $\alpha \to 0$ geht. Schreibt man $H(s)$ in Fakultäten und vereinfacht, ergibt sich:

$$H(s) = \frac{n!}{s!(n-s)!} \frac{s \cdot (ds)!}{s!} \frac{(dn-ds)!}{\left(\frac{s}{2}(d-1)\right)!} \frac{2^s}{\left(\frac{dn}{2} - \frac{s}{2}(d+1)\right)!} \frac{\frac{dn}{2}!}{(dn)!}$$

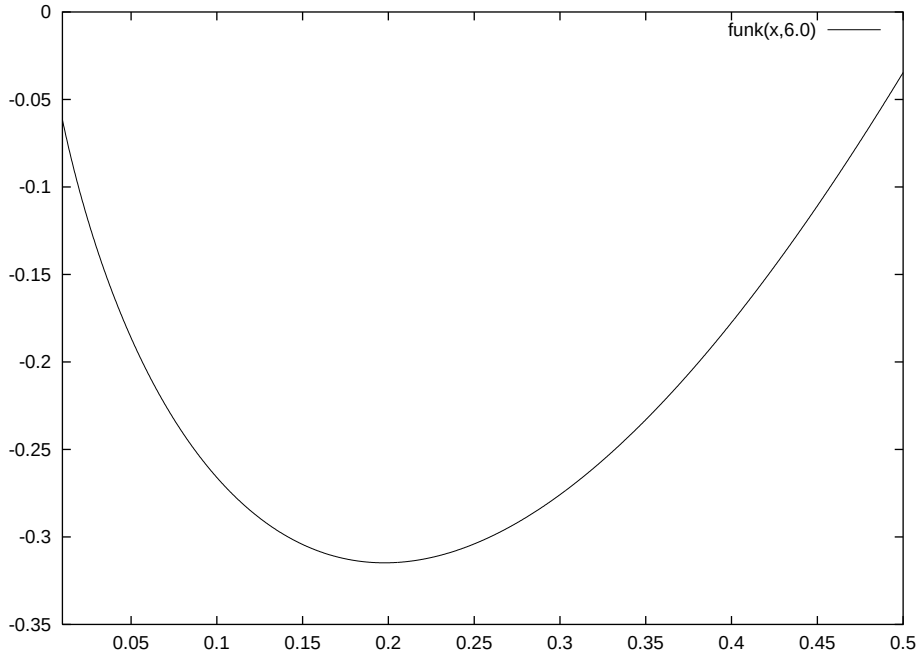


Abbildung 3: $F(\alpha, 6)$

Jetzt kann man $H(s)/H(s-2)$ berechnen und erhält

$$\frac{H(s)}{H(s-2)} = \frac{2^2 \prod_{i=1}^{2}(n-s+i)}{\prod_{i=0}^{1}(s-i)} \frac{\frac{s}{s-2}\prod_{i=0}^{2d-1}(ds-i)}{\prod_{i=0}^{1}(s-i)\prod_{i=1}^{2d}(dn-ds+i)} \frac{\prod_{i=1}^{d+1}\left(\frac{dn}{2}-\frac{s}{2}(d+1)+i\right)}{\prod_{i=0}^{d-2}\left(\frac{s}{2}(d-1)-i\right)}$$

Um den maximalen Wert abzuschätzen, nimmt man für den Zähler die maximalen und für den Nenner die minimalen Faktoren; gleichzeitig ersetzt man s durch αn und vereinfacht:

$$\frac{H(\alpha n)}{H(\alpha n-2)} \leq \frac{(n-\alpha n+2)^2}{(\alpha n-1)^4} \frac{\alpha n}{(\alpha n-2)} \frac{(d\alpha n)^{2d}}{(dn-d\alpha n+1)^{2d}} \frac{2^2\left(\frac{1}{2}\right)^{d+1}}{\left(\frac{1}{2}\right)^{d-1}} \frac{(dn-\alpha n(d+1)+2(d+1))^{d+1}}{(\alpha n(d-1)-2(d-2))^{d-1}}$$

n und dn werden ausgeklammert und gekürzt:

$$\frac{H(\alpha n)}{H(\alpha n-2)} \leq \frac{\left(1-\alpha+\frac{2}{n}\right)^2}{\alpha^4\left(1-\frac{1}{\alpha n}\right)^4} \frac{\alpha}{\alpha\left(1-\frac{2}{\alpha n}\right)} \frac{\alpha^{2d}}{\left(1-\alpha+\frac{1}{dn}\right)^{2d}}$$

$$\frac{\left(d-\alpha(d+1)+\frac{2}{n}(d+1)\right)^{d+1}}{\alpha^{d-1}\left((d-1)-\frac{2(d-2)}{\alpha n}\right)^{d-1}}$$

Da $n\alpha \geq 3$ gilt:

$$\begin{aligned}\frac{H(\alpha n)}{H(\alpha n-2)} &\leq \frac{1^2\cdot 3}{\left(1-\frac{1}{3}\right)^4}\frac{\alpha^{d-3}}{(1-\alpha)^{2d}}\frac{d^{d+1}}{\left(d-1-\frac{2}{3}(d-2)\right)^{d-1}} \\ &= \frac{243}{16}\frac{\alpha^{d-3}}{(1-\alpha)^{2d}}\left(\frac{d}{\frac{1}{3}d+\frac{1}{3}}\right)^{d-1}d^2\end{aligned}$$

Da wir den Rand betrachten gilt $\alpha \leq 0.01$. Für die Abschätzung des Maximums werden wieder im Zähler Maximal- und im Nenner Minimalwerte eingesetzt; für $d = 6$ ergibt sich:

$$\frac{H(\alpha n)}{H(\alpha n-2)} \leq \frac{243}{16}\frac{0.01^3}{0.99^{12}}\left(\frac{18}{7}\right)^5 36 < 0.07 < 1$$

Das bedeutet, daß $H(s)$ für $0 \leq s \leq 0.01n$ monoton fällt. Für eine obere Schranke der Wahrscheinlichkeit der Existenz eines kleinen Bad-Sets, genügt es also die Wahrscheinlichkeit für ein minimales Bad-Set zu berechnen.

Ein Bad-Set muß für $d = 6$ mindestens 7 Knoten enthalten, da eine beliebige Teilmenge mit weniger als 7 Knoten in einem 6-regulären Graphen mindestens 6 Cut-Kanten hat und damit kein Bad-Set sein kann.

Berechnet man die Wahrscheinlichkeit $H(7)$ ergibt sich:

$$H(7) = K\cdot\binom{n}{7}\binom{6n-42}{7}\frac{M(6n-49)}{M(6n)} = O(n^{-10})$$

Die Wahrscheinlichkeit für die Existenz eines kleinen Bad-Set sinkt mit mindestens $O(n^{-10})$ und damit die Wahrscheinlichkeit der Existenz eines beliebigen Bad-Sets mit $O(n^{-9})$, denn wir müssen die maximale Wahrscheinlichkeit mit $\frac{n}{2}$ multiplizieren, um die Gesamtwahrscheinlichkeit zu erhalten.

4.6 Eine Reduktion von E2-LIN2 auf E2-LIN2-7OCC

An Hand eines Beispiels soll gezeigt werden, wie man das gerade Bewiesene verwendet, um eine Reduktion zu bauen. Dazu soll E2-LIN2 auf die Variante des Problems reduziert werden, bei dem jede Gleichung *genau* 7 Variablen enthält.

Jede Variable in E2-LIN2 wird wie vorher erklärt auf n Hilfsvariablen im E2-LIN2-7OCC abgebildet, und die Hilfsvariablen werden mit Hilfe eines zufällig erzeugten 6-regulären

Graphen verbunden. Um eine gültige Lösung für das E2-LIN2 zu erhalten, wird wie oben beschrieben verfahren.

Nehmen wir an, daß das E2-LIN2 m Gleichungen hat. Nehmen wir weiterhin an, daß es keine Gleichungen der Form $x + x = z$ gibt, da die Erfüllung solcher Gleichungen unabhängig von der Belegung von x ist. An jeder der m Gleichungen sind dann genau 2 Knoten aus 2 verschiedenen 6-regulären Graphen im E2-LIN2-70CC beteiligt.

In einem 6-regulären Graphen mit C Knoten gibt es genau $3C$ Gleichungen; hinzu kommen die Gleichungen, die die 6-regulären Graphen untereinander verbinden und die aus dem ursprünglichen E2-LIN2 stammen. In dem transformierten E2-LIN2-7OCC gibt es also $2 \cdot 3m + m = 7m$ Gleichungen.

Wenn es im E2-LIN2 S erfüllte Gleichungen gibt, dann gibt es im E2-LIN2-7OCC nach der Umbelegung der 6-regulären Teilgraphen genau $6m + S$ erfüllte Gleichungen (weil alle Gleichungen in den 6-regulären Teilgraphen erfüllt sind).

Laut Håstad [26] läßt sich ein E2-LIN2 mit $16m$ Gleichungen nicht besser als 12/11 approximieren. Daraus folgt, daß sich ein E2-LIN2-7OCC mit $112m$ Gleichungen nicht besser als 108/107 approximieren läßt, falls die Reduktion korrekt ist.

Um das zu zeigen, muß sichergestellt sein, daß die Anzahl der Knoten in den 6-regulären Graphen mit der Anzahl der Variablen (und damit mit der Anzahl der 6-regulären Teilgraphen) im E2-LIN2 mitwächst, damit die Wahrscheinlichkeit, daß ein 6-regulärer Graph *kein* Expander ist schneller als $O(1/k)$ gegen Null geht (wobei k die Anzahl der Variablen im E2-LIN2 ist).

Das wird dadurch erreicht, daß vor der Transformation die Gleichungen im E2-LIN2 vervielfältigt werden: Nehmen wir an, daß das E2-LIN2 m Gleichungen und k Variablen hat; jede Gleichung wird dann genau k mal aufgeschrieben, so daß ein E2-LIN2 mit km Gleichungen entsteht, bei dem allerdings jede Variable mindestens k-mal vorkommt.

Sind im nicht vervielfältigten E2-LIN2 S Gleichungen erfüllt und im entsprechenden E2-LIN2-7OCC $6m+S$ Gleichungen, dann sind im vervielfältigten Fall kS Gleichungen erfüllt und im entsprechenden E2-LIN2-7OCC $6km+kS = k(6m+S)$ Gleichungen. Daraus folgt, daß sich ein E2-LIN2-7OCC mit $112km$ Gleichungen nicht besser als $108k/107k = 108/107$ approximieren läßt; die Approximierbarkeit hat sich also nicht geändert. Allerdings ist jetzt sichergestellt, daß die Anzahl der Knoten pro 6-regulärem Graphen größer als die Anzahl der benötigten Expander ist. Daraus folgt, daß die Wahrscheinlichkeit, daß ein 6-regulärer Teilgraph *kein* Expander ist, schneller als $O(1/k)$ gegen Null geht (wobei k die Anzahl der benötigten Expander ist) und damit ist die Reduktion korrekt.

4.7 E2-LIN2-3OCC

Natürlich sind besonders die Extremfälle eines Problems interessant. Es ist bekannt, daß man für ein E2-LIN2, in dem jede Variable nicht mehr als 2-mal vorkommt in polynomieller Zeit eine optimale Belegung findet.

Beweis:
Ein E2-LIN2 kann auch als Graph aufgefaßt werden. Die Knoten des Graphen stellen die Variablen dar und die Kanten die Gleichungen. Nehmen wir an, jede Variable kommt genau zweimal vor, und es gibt keine Gleichungen der Form $x+x = z$ (weil die Erfüllbarkeit dieser Gleichungen unabhängig von der Belegung von x ist).

In diesem Fall besteht der Graph aus einem oder mehreren Kreisen. Wir fangen an einem beliebigen Knoten in einem beliebigen Kreis an und belegen die entsprechende Variable mit dem aktuellen Wert 1.

Wir gehen im Uhrzeigersinn zum nächsten Knoten im Kreis; wird dabei eine Kante benutzt, die für eine Gleichung der Form $x + y = 1$ steht, dann wird der Wert gewechselt (von 1 auf 0 oder von 0 auf 1); steht die Kante für eine Gleichung der Form $x + y = 0$, dann wird der aktuelle Wert beibehalten. So fährt man fort, bis man wieder an der ersten Variable angelangt ist, die im Kreis belegt wurde. Alle Kreise werden mit dieser Methode abgearbeitet.

Kreise, die ein ungerade Anzahl von Kanten enthalten, die für Gleichungen der Form $x + y = 1$ stehen, enthalten immer genau eine unerfüllte Gleichung.

Mit dem beschriebenen Algorithmus erhält man eine optimale Belegung.

Erlaubt man auch Variablen, die nur einmal vorkommen, dann enthält der Graph neben Kreisen auch Pfade. Die Gleichungen in einem Pfad können immer alle erfüllt werden, wenn man an einem Ende anfängt und sonst wie bei den Kreisen vorgeht.

Die Frage ist also, wie gut man E2-LIN2-3OCC approximieren kann. Könnte man einen 2-regulären Expander bauen (schließlich braucht man pro Knoten noch eine Kante nach außen), dann würde eine Reduktion von E2-LIN2 auf E2-LIN2-3OCC genauso wie auf E2-LIN2-7OCC funktionieren.

Da man offensichtlich keinen 2-regulären Graphen mit den gewünschten Expandereigenschaften bauen kann, ist für E2-LIN2-3OCC eine kompliziertere Konstruktion nötig: Man bildet wieder jede Variable des E2-LIN2 auf n Variablen im E2-LIN2-3OCC ab, wobei n wieder die Anzahl der Vorkommen der Variable im E2-LIN2 ist; jede dieser Variablen im E2-LIN2-3OCC ist also wieder an genau einer Gleichung des ursprünglichen E2-LIN2 beteiligt. Diese Variablen werden im weiteren *Contacts* genannt.

Es werden jetzt weitere, zusätzliche Variablen benötigt, um eine Verbindungsstruktur zwischen den Variablen im E2-LIN2-3OCC zu erzeugen, die garantiert, daß man eine Belegung im E2-LIN2-3OCC so verändern kann, daß man eine gültige Belegung für das E2-LIN2 erhält. Diese Variablen werden im weiteren *Checker* genannt.

4.8 Eine vollständig randomisierte Konstruktion

Zunächst soll eine vollständig randomisierte Konstruktion analysiert werden.

Da ein 3-regulärer Graph benötigt wird, braucht jeder Contact noch zwei zusätzliche Kanten; die Checker sind bis jetzt an gar keiner Gleichung beteiligt und deshalb braucht man noch drei Kanten pro Checker.

Die Verbindungsstruktur muß garantieren, daß bei einer Umbelegung, die allen C Contacts, die aus einer Variablen des E2-LIN2 entstanden sind, den gleichen Wert zuweist, mehr Gleichungen erfüllt als gebrochen werden.

Nehmen wir an, daß wir zu den C Contacts noch kC Checker hinzufügen und die Kanten, die die Contacts und Checker verbinden, völlig zufällig erzeugen. Die Frage ist, wie groß k sein muß, damit ein Graph, der zufällig auf den $(k+1)C$ Knoten erzeugt wird, mit hoher Wahrscheinlichkeit die gewünschte Eigenschaft besitzt; nämlich: Eine beliebige Teilmenge mit s Knoten, die $a \leq C/2$ Contacts besitzt, muß mindestens a Cut-Kanten haben.

Diese Frage kann wieder mit einer ähnlichen Vorgehensweise wie bei E2-LIN2-7OCC analysiert werden. Wir benutzen wieder die Konstruktion von Bollobas, d.h. wir weisen den Contacts 2 und den Checkern 3 Hilfsknoten zu (weil die Contacts noch 2 und die Checkers noch 3 Kanten benötigen). Nehmen wir an, wir haben eine vorgegebene Menge an Knoten, die a Contacts und s Checkers enthält. Die Wahrscheinlichkeit, daß die Menge *genau* i Cut-Kanten hat, ist (siehe Abb. 4):

$$\begin{aligned} P(a,s,i) &= \binom{3s+2a}{i}\binom{(3k+2)C-3s-2a}{i} \\ &\quad i!\frac{M[3s+2a-i]M[(3k+2)C-3s-2a-i]}{M[(3k+2)C]} \end{aligned}$$

Die Wahrscheinlichkeit steigt mit i an, also benutzen wir als Abschätzung $i = a$, d.h. die Wahrscheinlichkeit, daß eine feste Teilmenge mit a Contacts und s Checkern ein Bad-Set ergibt sich als:

$$P(a,s) = \sum_{i=0}^{a-1} P(a,s,i) \leq aP(a,s,a)$$

Die Wahrscheinlichkeit muß noch mit der Anzahl der Möglichkeiten, eine Menge mit a Contacts und s Checkern zu erzeugen, multipliziert werden. Man erhält:

$$P(\exists\,\text{Bad-Set}) = \sum_{a=1}^{C/2}\sum_{s=0}^{kC} \max\left\{\binom{C}{a}\binom{kC}{s}P(a,s)\right\}$$

Setzt man für $a = \alpha C$ mit $0 \leq \alpha \leq 0.5$ und $s = \beta kC$ mit $0 \leq \beta \leq 1$, kann man diese Wahrscheinlichkeit wieder als $e^{C\cdot F(\alpha,\beta,k)+O(\ln n)}$ schreiben. Dabei ergeben sich die höchsten Werte für F bei $\alpha = 0.5$. Solange $k < 6$ ist, gilt $F(0.5,\beta,k) > 0$; erst bei $k = 6$ gilt $F(0.5,\beta,6) < 0$ für $0.34 \leq \beta \leq 0.66$.

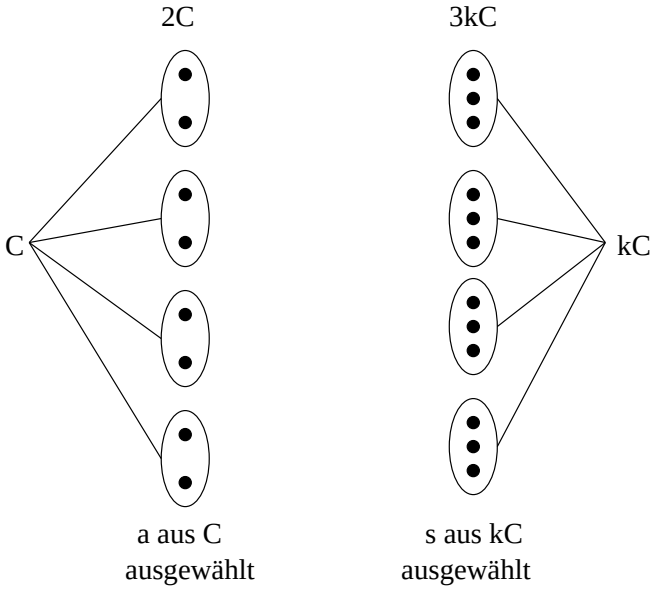


Abbildung 4: Das Modell für die zufällige Konstruktion

Das Problem ist, daß die vollständig randomisierte Konstruktion, bei einer ausgewählten Menge an Contacts in einem Bad-Set, nicht automatisch eine Menge an ausgewählten Checkern erzwingt.

Speziell der Fall, in dem keine Checker ausgewählt sind, bereitet Schwierigkeiten. Die Wahrscheinlichkeit, daß es eine Menge mit 3 Contacts gibt (siehe Abb. 5), die weniger als 3 Cut-Kanten hat, geht nämlich mit diesem Berechnungsmodell gar nicht gegen Null.

Abbildung 5: Bad-Sets mit 3 Contacts

Um das zu sehen, wird die Wahrscheinlichkeit abgeschätzt, daß es eine Konfiguration mit einer Teilmenge aus 3 Contacts gibt, die gar keine Cut-Kante hat. Es sei $d = (3k + 2)$:

$$\begin{aligned}
\binom{C}{3} P(3,0,0) &= K \cdot \binom{C}{3} \frac{M(dC-6)}{M(dC)} \\
&= \frac{K \cdot C!}{3!(C-3)!} \frac{(dC-6)!}{\frac{dC-6}{2}! 2^{\frac{dc-6}{2}}} \frac{\frac{dC}{2}! 2^{\frac{dC}{2}}}{(dC)!} \\
&= \frac{K \cdot C(C-1)(C-2)\frac{dC}{2}(\frac{dC}{2}-1)(\frac{dC}{2}-2)2^3}{3!dC(dC-1)(dC-2)(dC-3)(dC-4)(dC-5)}
\end{aligned}$$

$$
\begin{aligned}
&= \frac{K \cdot C(C-1)(C-2)(\frac{d}{2})^3 2^3 C(C-\frac{2}{d})(C-\frac{4}{d})}{6d^6 C(C-\frac{1}{d})(C-\frac{2}{d})(C-\frac{3}{d})(C-\frac{4}{d})(C-\frac{5}{d})} \\
&= \frac{K \cdot C(C-1)(C-2)}{6d^3(C-\frac{1}{d})(C-\frac{3}{d})(C-\frac{5}{d})} \\
\lim_{C\to\infty} \binom{C}{3} P(3,0,0) &= \frac{K}{6d^3} = \frac{K}{6(3k+2)^3}
\end{aligned}
$$

Man zählt bei dieser Abschätzung natürlich zuviel, da Konfigurationen, in denen mehr als zwei solcher Bad-Sets existieren, mehrfach gezählt werden; das ändert aber nichts an dem eigentlichen Problem.

Eine erfolgreiche Konstruktion ist nur möglich, wenn es einem gelingt, die Anzahl der Checker in einem Bad-Set an die Anzahl der Contacts zu koppeln. Eine Konstruktion die das erreicht, wird im folgenden beschrieben.

5 Das Wheel

Piotr Berman und Marek Karpinksi [13] haben beschrieben, wie eine Reduktion von E2-LIN2 auf E2-LIN2-3OCC mit 6 Checkern pro Contact möglich ist. Sie nennen die Verbindungsstruktur zwischen den Contacts und Checkern Wheel; sie sieht wie in Abbildung 6 aus. Das Wheel besteht aus einem Kreis, auf dem die Contacts und Checker wie abgebildet regelmäßig verteilt sind und aus einem zufällig erzeugten perfekten Matching zwischen den Checkern.

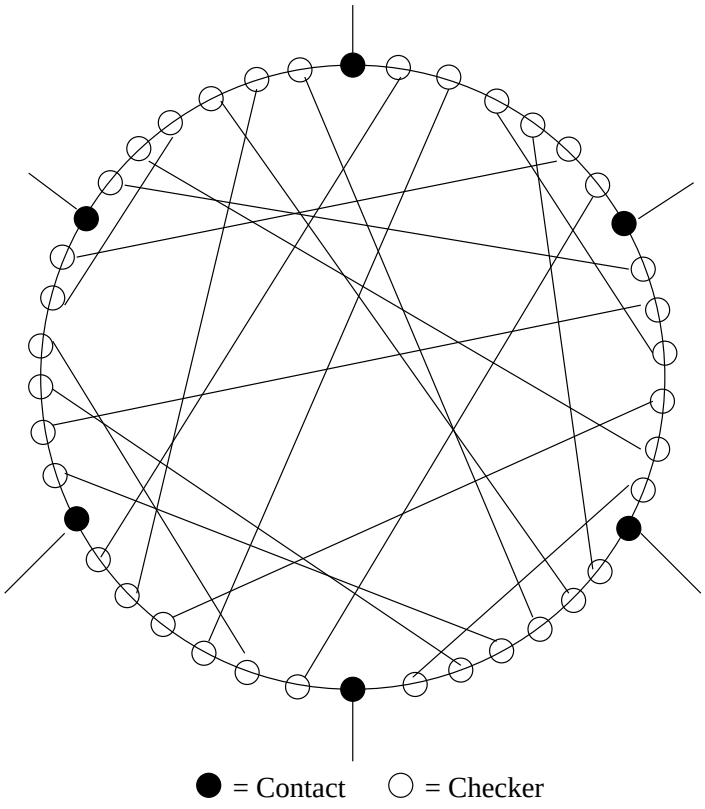


Abbildung 6: Ein Wheel mit 6 Contacts und jeweils 6 Checkern zwischen den Contacts

Damit die Reduktion korrekt ist, muß jede beliebige Teilmenge mit $a \leq C/2$ Contacts und s Checkern mindestens a Cut-Kanten besitzen, wobei C die Anzahl der Contacts im Wheel bezeichnet. Im Folgenden werden Teilmengen mit bestimmten Parametern betrachtet, nämlich:

- a, die Anzahl der Contacts in der Teilmenge.
- s, die Anzahl der Checker in der Teilmenge.

- f, die Anzahl der Fragmente der Teilmenge.

Mit einem „Fragment“ ist dabei eine zusammenhängende Menge an ausgewählten Knoten auf dem Kreis des Wheels gemeint. Jedes Fragment hat mindestens 2 Cut-Kanten, nämlich die beiden Kreis-Kanten, die am Rand des Fragmentes liegen (siehe Abb. 7).

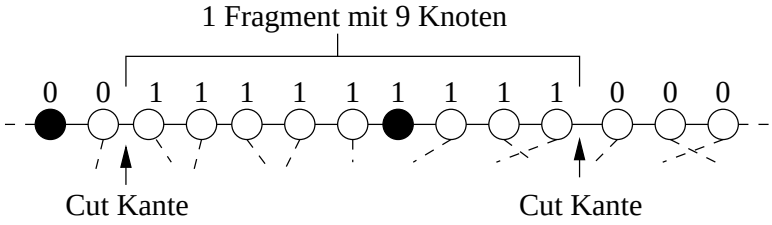


Abbildung 7: 1 Fragment auf dem Kreis im Wheel

5.1 Bad-Set Wahrscheinlichkeit für ein feste Teilmenge

Nehmen wir an, wir haben eine feste Teilmenge an Knoten, die die Parameter a,s und f hat. Für die Wahrscheinlichkeit, daß bei einer solchen Menge genau i Matching-Kanten Cut-Kanten sind (solche Kanten werden im weiteren Matching-Cut-Kanten genannt), ergibt sich nach Bollobas:

$$P(s,i) \quad = \quad \frac{\binom{s}{i}\binom{6C-s}{i} i! M(s-i) M(6C-s-i)}{M(6C)}$$

Die Wahrscheinlichkeit, daß es sich bei der Teilmenge um ein Bad-Set handelt, d.h. daß sie weniger als a Cut-Kanten hat, ergibt sich als Summe über die Anzahl der möglichen Matching-Cut-Kanten. Da die $2f$ Fragmentgrenzen der ausgewählten Knoten bereits $2f$ Cut-Kanten darstellen, darf man höchstens noch $a-2f-1$ Cut-Kanten zusätzlich haben; da $P(s,i)$ in Bezug auf i monoton steigt (siehe Kapitel 4.5), schätzt man mit $i = a-2f$ ab, d.h:

$$P(a,f,s) \quad = \quad \sum_{i=0}^{a-2f-1} P(s,i) \leq (a-2f) P(s, a-2f-1) \leq (a-2f) P(s, a-2f)$$

5.2 Ein paar kombinatorische Erläuterungen

Um die Wahrscheinlichkeit für die Existenz eines Bad-Set mit festen a,f und s zu erhalten, muß wie in den vorangegangenen Analysen die Anzahl der Knotenmengen mit diesen

Parametern gezählt werden; deshalb werden zunächst ein paar kombinatorische Formeln erklärt.

Will man eine Menge von n Kreisknoten in k Fragmente aufteilen, dann stellt man sich einfach vor, daß man $k-1$ Fragmentkanten auf die $n-1$ Zwischenräume verteilt und erhält $\binom{n-1}{k-1}$ (siehe Abb. 8). In diesem Fall enthält jedes Fragment mindestens einen Knoten; als obere Schranke kann man natürlich auch zur Vereinfachung $\binom{n}{k}$ benutzen.

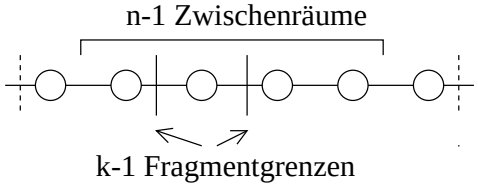


Abbildung 8: n Knoten auf k Fragmente verteilen

Bilden die Knoten einen Kreis (wenn man z.B. alle Knoten des Wheels betrachtet), dann gibt es $\binom{n}{k}$ Möglichkeiten, weil man k Fragmentgrenzen auf n mögliche Zwischenräume verteilen muß.

Will man n Knoten auf k Fragmente aufteilen und erlaubt dabei auch leere Fragmente, die gar keinen Knoten enthalten, dann erhält man $\binom{n+k-1}{k-1}$. Man kann sich das so vorstellen: Man braucht $n+k-1$ Plätze, auf die man $k-1$ Fragmentgrenzen verteilt; die n übriggebliebenen Plätze werden als Knoten betrachtet. Dadurch wird es möglich, daß zwei Fragmentgrenzen direkt nebeneinander liegen, und deshalb kann man auch leere Fragmente erzeugen (Siehe Abb. 9).

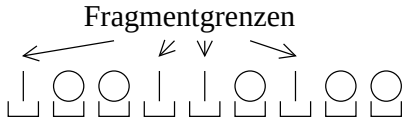


Abbildung 9: 5 Fragmente (2 leere) und 5 Knoten

Auch hier kann man sich die Knoten auf einem Kreis angeordnet vorstellen; in diesem Fall hat man $n+k$ Plätze, auf die man k Fragmentgrenzen verteilt, und erhält $\binom{n+k}{k}$

5.3 Anzahl der Teilmengen mit festen a,f und s

Beim Zählen der Teilmengen kommt die entscheidende Eigenschaft des Wheels zur Geltung: die Anzahl der Checker ist an die Anzahl der Contacts gekoppelt.

Nehmen wir an, ein Fragment enthält a_f Contacts. Die Anzahl der Checker in dem Fragment muß dann zwischen $6(a_f-1)$ und $6(a_f+1)$ liegen. Summiert man über alle Fragmente auf, erhält man, daß eine Teilmenge, mit f Fragmenten und a Contacts, zwischen $6(a-f)$ und $6(a+f)$ Checkern enthalten muß.

Für die weiteren Abschätzungen werden a,f und s wieder durch Anteile von C ausgedrückt:

- $a = \alpha C$ mit $0.01 \leq \alpha \leq 0.5$
- $f = \beta\alpha C$ mit $0 \leq \beta \leq 0.5$
- $s = \gamma\alpha 6C$ mit $(1-\beta) \leq \gamma \leq (1+\beta)$
  $\Leftrightarrow \alpha 6C - \beta\alpha 6C \leq \gamma 6C \leq \alpha 6C + \beta\alpha 6C$
  $\Leftrightarrow 6(a-f) \leq s \leq 6(a+f)$

Zum Zählen der Knotenmengen werden zwei verschiedene Methoden verwendet. Bei beiden Methoden ergibt sich eine e-Funktion der Form $e^{C\cdot Z(\alpha,\beta,\gamma)+O(\ln n)}$; es wird immer die Methode gewählt, die zu einem kleineren Exponenten führt.

Als erstes soll die kompliziertere Methode erläutert werden: In einer Teilmenge gibt es, wie vorher erläutert, mindestens $6(a-f)$ Checker. Diese Minimalbelegungen ergibt sich, wenn man die Fragmentgrenzen genau an die Contacts legt. Zum Zählen verteilt man erst alle Contacts auf die $2f$ Fragmente (f ausgewählte und f unausgewählte Fragmente); da man auch Fragmente ohne Contacts haben kann, gibt es weniger als $2\binom{C+2f}{2f}$ verschiedene Möglichkeiten die Fragmentgrenzen zu verteilen. Die 2 kommt hinzu, weil man bei jeder Möglichkeit noch festlegen muß, welche Fragmente ausgewählt sind und welche nicht. Es werden trotzdem zu viele Möglichkeiten gezählt, weil man nicht mehr als 7 Fragmentgrenzen zwischen zwei Contacts legen darf, da jedes Fragment mindestens einen Knoten enthalten muß und es nur 6 Checker zwischen zwei Contacts gibt.

Wenn eine ausgewählte Knotenmenge mehr als $6(a-f)$ Checker enthält, dann müssen die $d = s - 6(a-f)$ zusätzlichen Checker auf die Fragmenträndern verteilt werden, d.h. die Ränder müssen von den Contacts weg verschoben werden. Dazu werden die d Zusatz-Checker auf $2f$ Fragmenträndern verteilt; weil es auch möglich ist, einen Fragmentrand leer zu lassen, gibt es dafür weniger als $\binom{d+2f}{2f}$ verschiedene Möglichkeiten. (Man zählt hier zuviel, weil die Einschränkung, daß auf einen Fragmentrand nicht mehr als 6 Checker verteilt werden können, nicht berücksichtigt wird.)

Ein Sonderfall, der dabei genauer erläutert werden muß, sind Fragmente, die gar keinen Contact enthalten. Sie werden bei der Abschätzung des Minimums an Checkern mit -6 gezählt; man kann sich das vorstellen, indem man bei einem solchen Fragment die beiden Ränder über Kreuz an die benachbarten Contacts schiebt.

Bei der Verteilung der d Checker müssen die überkreuzten Ränder eines Contact-losen Fragmentes *zusammen* um mindestens 7 Checker verschoben werden, damit überhaupt ein gültiges Fragment entsteht (siehe Abbildung 10).

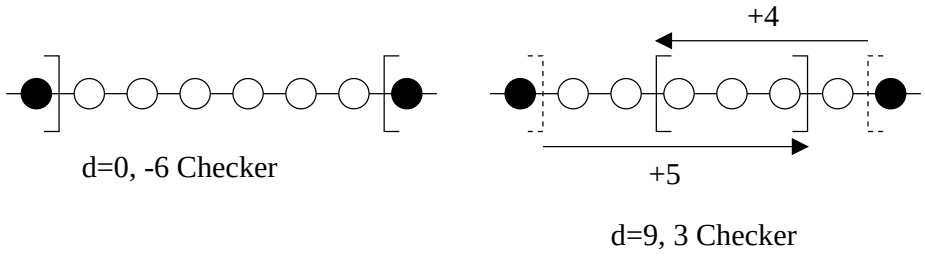


Abbildung 10: Ein Contact-loses Fragment

Da bei der Abschätzung der Möglichkeiten mit $\binom{d+2f}{d}$ diese Beschränkung ebenfalls nicht berücksichtigt wird, zählt man wieder zuviel; es werden aber auf keinen Fall zu wenig Möglichkeiten berücksichtigt, da jede gültige Möglichkeit in der Abschätzung enthalten ist.

Die selbe Methode liefert eine zweite Abschätzung, indem man die Maximalbelegungen nimmt und Checker von den Rändern wegstreicht. Die Maximalbelegung entsteht, wenn man die Minimalbelegung nimmt und an jeden Fragmentrand noch 6 Checker hinzufügt. Es ergeben sich genau die gleichen Formeln, nur verteilt man jetzt $d = 6(a + f) - s$ zu entfernende Checker auf die Ränder.

Bei einem Contact-losen Fragment dürfen hier von beiden Rändern zusammen nicht mehr als 6 Checker entfernt werden, allerdings sind wie bei der ersten Abschätzung alle gültigen Belegungen enthalten, d.h. man zählt ebenfalls nie zu wenig.

Die zweite Methode ist sehr viel einfacher: Man hat im Wheel $7C$ Knoten, nämlich C Contacts und $6C$ Checker. Man verteilt die Knoten auf $2f$ Fragmente, nämlich auf f ausgewählte und f unausgewählte Fragmente. Daher ergeben sich $2\binom{7C}{2f}$ Möglichkeiten die Fragmente auf das Wheel zu verteilen. Die Anzahl der Checker oder Contacts wird bei dieser Methode nicht weiter berücksichtigt.

5.4 Wahrscheinlichkeit für $a > 0.02C$

Wenn man die Formeln noch einmal zusammenfaßt ergibt sich:

$$P(\exists \text{ Bad-Set}) \;=\; \sum_{a=1}^{\frac{C}{2}} \sum_{f=1}^{\frac{a}{2}} \sum_{s=6(a-f)}^{6(a+f)} A(a,f,s)P(a,f,s)$$

Wobei $A(a, f, s)$ die Anzahl der Teilmengen bezeichnet, die genau a Contacts, f Fragmente und s Checker besitzen; $P(a, f, s)$ bezeichnet die Wahrscheinlichkeit, daß eine feste Teilmenge mit diesen Parametern ein Bad-Set ist.

Um eine obere Schranke für diese Wahrscheinlichkeit zu bestimmen, wird nur das Maximum betrachtet, d.h.:

$$P(\exists \text{ Bad-Set}) \leq \frac{C}{2}\frac{C}{4}3C \max\left\{ A(a,f,s)P(a,f,s) : \begin{array}{c} 1 \leq a \leq \frac{C}{2} \\ 1 \leq f \leq \frac{a}{2} \\ 6(a-f) \leq s \leq 6(a+f) \end{array} \right\}$$

$A(a,f,s)$ wird wie oben ausgeführt mit zwei verschiedenen Methoden bestimmt.

Drückt man a, f und s wie oben erklärt durch α,β und γ aus, erhält man eine Funktion, die wieder die Form $e^{C\cdot F(\alpha,\beta,\gamma)+O(\ln C)}$ hat. Dabei ergibt die zweite Methode, die $A(a,f,s)$ mit $\binom{7C}{2f}$ abschätzt, die kleineren Werte im Exponenten, wenn $s \approx 6a$, d.h. wenn $\gamma \approx 1.0$ ist. Die erste Methode liefert die besseren Werte am Rand, d.h. wenn s sich $6(a-f)$ oder $6(a+f)$ nähert.

Ist $\alpha \geq 0.02$ ergibt sich das Maximum bei $\alpha = 0.50$, $\beta = 0.38$, $\gamma = 1.17$ und beträgt $F(0.5, 0.38, 1.17) \approx -0.0573$. Diese Kombination aus α,β und γ, liegt genau an der Grenze zwischen erster und zweiter Zählweise.

5.5 Wahrscheinlichkeit für $a \leq 0.02C$

Auch hier muß noch bewiesen werden, daß die randomisierte Konstruktion auch für $\alpha \leq 0.02$ funktioniert. Das soll - unter der Voraussetzung, daß $a \leq 0.02C$ ist - in 4 Schritten bewiesen werden; Sei $H(a,f,s) = A'(a,f,s)P(a,f,s)$:

- $H(a,f,s-2) \geq H(a,f,s)$
- $H(a,f+1,s_{min1}) \geq H(a,f,s_{min2})$, wenn $f \leq a/4$,
  mit $s_{min1} = 6(a-f-1)$ und $s_{min2} = 6(a-f)$
- $H(a-2,f-1,s_{min1}) \geq H(a,f,s_{min2})$, wenn $f > a/4$,
  mit $s_{min1} = 6(a-f-1)$ und $s_{min2} = 6(a-f)$
- $H(4,1,18) = O(C^{-7})$ und $H(3,1,12) = O(C^{-5})$

Dazu verwenden wir eine modifizierte Zählweise $A'(a,f,s)$: eine Teilmenge mit s Checkern kann auf höchstens $\binom{6C}{f}\binom{s}{f}$ verschiedene Arten erzeugt werden. Dazu bestimmt man erst die f Fragmentanfänge und dann die f Fragmentlängen; die Contacts werden dabei durch eine Kante ersetzt; die ausgewählten Contacts ergeben sich aus den ausgewählten Checkern. Eine Wahlmöglichkeit bleibt nur, wenn die Contacts genau an einem Fragmentrand liegen. Die Belegung dieser Rand-Contacts ändert aber nichts an den ausgewählten Checkern und deshalb kann man diese Variationsmöglichkeit ignorieren.

Ausgeschrieben lautet $H(a,f,s)$:

$$\begin{aligned} H(a,f,s) &= \binom{6C}{f}\binom{s}{f}(a-2f)\binom{s}{a-2f-1}\binom{6C-s}{a-2f-1} \\ &\quad (a-2f-1)!\frac{M(s-a+2f+1)M(6C-s-a+2f+1)}{M(6C)} \end{aligned}$$

Schreibt man die Fakultäten aus und kürzt, ergibt sich:

$$\begin{aligned} H(a,f,s) &= \frac{(6C)!}{f!(6C-f)!)}\frac{s!}{f!(s-f)!}\frac{(a-2f)s!}{(a-2f-1)!} \\ &\quad \frac{(6C-s)!}{\left(\frac{s}{2}-\frac{a}{2}+f+\frac{1}{2}\right)!}\frac{2^{a-2f-1}}{\left(3C-\frac{s}{2}-\frac{a}{2}+f+\frac{1}{2}\right)!}\frac{(3C)!}{(6C)!} \end{aligned}$$

Um zu zeigen, daß $H(a,f,s)$ in Bezug auf s monoton fällt, wird $\frac{H(a,f,s)}{H(a,f,s-2)}$ berechnet:

$$\begin{aligned} \frac{H(a,f,s)}{H(a,f,s-2)} &= \frac{s(s-1)}{(s-f)(s-f-1)}\frac{s(s-1)\left(3C-\frac{s}{2}-\frac{a}{2}+f+\frac{3}{2}\right)}{(6C-s+1)(6C-s+2)\left(\frac{s}{2}-\frac{a}{2}+f+\frac{1}{2}\right)} \\ &= \frac{s(s-1)}{(s-f)(s-f-1)}\frac{s(s-1)(6C-s-(a-2f-1)+2)}{(6C-s+1)(6C-s+2)(s-(a-2f-1))} \end{aligned}$$

Die benötigten Abschätzungen beruhen alle auf zwei Voraussetzung $6(a-f) \leq s-2 < s \leq 6(a+f)$ und $1 \leq f < a/2$ Daraus ergeben sich folgende Ungleichungen:

$$\begin{aligned} \frac{s}{s-f} &\leq \frac{s-1}{s-f-1} \leq \frac{6(a-f)}{6(a-f)-f} = \frac{6}{6-\frac{f}{a-f}} < \frac{6}{5} \\ \frac{s-1}{s-(a-2f-1)} &\leq \frac{s}{s-(a-2f-1)} \leq \frac{6(a-f)}{6(a-f)-(a-f)+f+1} = \frac{6}{5+\frac{f+1}{a-f}} \leq \frac{6}{5} \\ \frac{s}{6C-s+1} &\leq \frac{s}{6C-s} \leq \frac{9a}{6C-9a} \leq \frac{9\cdot 0.02C}{6C-9\cdot 0.02C} < 0.031 \end{aligned}$$

Daraus folgt:

$$\frac{H(a,f,s)}{H(a,f,s-2)} < \left(\frac{6}{5}\right)^3 \cdot 0.031 < 1$$

Als nächstes soll bewiesen werden daß $H(a,f,s_{min})$ in Bezug auf f monoton steigt, für $f \leq a/4$. Dabei gilt $s_{min} = 6a-6f$:

$$\begin{aligned} H(a,f,s_{min}) &= \frac{(6C)!}{f!(6C-f)!}\frac{(6a-6f)!}{f!(6a-7f)!}\frac{(a-2f)(6a-6f)!}{(a-2f-1)!} \\ &\quad \frac{(6C-6a+6f)!}{(2.5a-2f+0.5)!}\frac{2^{a-2f}}{(3C-3.5a+4f+0.5)!}\frac{(3C)!}{(6C)!} \end{aligned}$$

Daraus folgt:

$$\frac{H(a,f+1,s_{min1})}{H(a,f,s_{min2})} = \frac{6C-f}{(f+1)(f+1)} \frac{\prod_{i=0}^{6}(6a-7f-i)}{\prod_{i=0}^{5}(6a-6f-i)} \frac{(a-2f-2)}{(a-2f)} \frac{\prod_{i=1}^{2}(a-2f-i)}{\prod_{i=0}^{5}(6a-6f-i)} \frac{\prod_{i=1}^{6}(6C-6a+6f+i)\prod_{i=0}^{1}(2.5a-2f-i+0.5)}{2^2\prod_{i=1}^{4}(3C-3.5a+4f+i+0.5)}$$

Aus $1 \le f \le a/4$ und $5 \le a \le 0.02C$ folgt:

$$\begin{aligned}
\frac{6a-7f}{6a-6f} &\ge \frac{6(a-f)-f-6}{6(a-f)} = \frac{6-\frac{f+6}{a-f}}{6} \ge \frac{6-\frac{1}{3}-\frac{24}{3a}}{6} \ge 0.66 \\
\frac{a-2f-2}{a-2f} &\ge \frac{1}{3} \\
\frac{a-2f-1}{f+1} &\ge \frac{a-2f-2}{f+1} \ge \frac{\frac{1}{2}-\frac{2}{a}}{\frac{1}{4}+\frac{1}{a}} \ge \frac{2}{9} \\
\frac{2.5a-2f-0.5}{6a-6f} &\ge \frac{2.5a-0.5}{6a} \ge \frac{1}{3} \\
\frac{6C-6a+6f}{3C-3.5a+4f+4.5} &\ge \frac{6C-4.5a}{3C-2.5a+4.5} \ge \frac{6C-4.5\cdot 0.02C}{3C-2.5\cdot 0.02C+4.5} \approx 2 \\
\frac{6C-6a+6f}{6a-6f} &\ge \frac{6C-6a}{6a} \ge \frac{6C-0.02C}{0.12C} \ge 49 \\
\frac{6C-f}{6a-6f} &\ge \frac{6C}{6a} \ge 50
\end{aligned}$$

Daraus folgt, wenn $f \le a/4$ und $5 \le a \le 0.02C$:

$$\frac{H(a,f+1,s_{min1})}{H(a,f,s_{min2})} \ge (0.66)^7 \frac{1}{3}\left(\frac{2}{9}\right)^2\left(\frac{1}{3}\right)^2 2^4\cdot 49^2\cdot 50\cdot\frac{1}{4} > 1$$

Als nächstes wird gezeigt, daß für $f > a/4$ gilt, $H(a-2,f-1,s_{min1}) \ge H(a,f,s_{min2})$:

$$\frac{H(a,f,s_{min1})}{H(a-2,f-1,s_{min2})} = \frac{6C+f+1}{ff} \frac{\prod_{i=0}^{5}(6a-6f-i)}{\prod_{i=0}^{4}(6a-7f-i)} \frac{\prod_{i=0}^{5}(6a-6f-i)}{\prod_{i=0}^{2}(2.5a-2f-i+0.5)}$$

$$\frac{\prod_{i=1}^{3}(3C - 3.5a + 4f + i + 0.5)}{\prod_{i=1}^{6}(6C - 6a + 6f + i)}$$

Die benötigten Abschätzungen ergeben sich aus $f > a/4$ und $5 \leq a \leq 0.02C$ und lauten:

$$\begin{aligned}
\frac{6a-6f}{f} &\leq \frac{6a-1.5a}{0.25a} = 18 \\
\frac{6a-6f-1}{6a-7f} &\leq \frac{6a-6f-5}{6a-7f-4} = 1 + \frac{f-1}{6a-7f-4} \leq 1 + \frac{a-2}{5a-8} \leq 1.2 \\
\frac{6a-6f-1}{2.5a-2f+0.5} &\leq \frac{6a-6f-3}{2.5a-2f-1.5} = \frac{12a-12f-6}{5a-4f-3} \leq \frac{9a-6}{4a-3} \leq 2.3 \\
\frac{6C+f+1}{6C-6a+6f+1} &\leq \frac{6C+f}{6C-6a+6f} \leq \frac{C+\frac{f}{6}}{0.98C+f} \leq 1.021 \\
\frac{3C-3.5a+4f+1.5}{6C-6a+6f+2} &\leq \frac{3C-3.5a+4f+3.5}{6C-6a+6f+4} \leq \frac{3C-1.5a+3.5}{6C-3a+4} \leq \frac{1}{2} \\
\frac{6a-6f-5}{6C-6a+6f+6} &\leq \frac{6a-6f-4}{6C-6a+6f+5} \leq \frac{6a}{6C-6a} \leq \frac{1}{49}
\end{aligned}$$

Daraus ergibt sich unter den beschriebenen Voraussetzungen:

$$\frac{H(a,f,s_{min1})}{H(a-2,f-1,s_{min2})} \leq 18^2 \cdot (1.2)^5 \cdot (2.3)^3 \cdot 1.021 \cdot \left(\frac{1}{2}\right)^3 \cdot \left(\frac{1}{49}\right)^2 \leq 1$$

Aus dem bisher Bewiesenen läßt sich folgendes ableiten: Wenn eine obere Schranke für die Wahrscheinlichkeit gesucht ist, daß es ein Bad-Set mit den Parametern $3 \leq a \leq 0.02C$, f und s gibt, dann kann man die Parameter iterativ verändern, bis man entweder bei $H(3,1,12)$ oder $H(4,1,12)$ angelangt ist, ohne dabei die Wahrscheinlichkeit zu verringern.

Zuerst setzt man s auf den minimal möglichen Wert. Ist $f < a/4$, dann erhöht man f so lange, bis man einen Wert $f \geq a/4$ erreicht hat (s wird dabei an f angepaßt). Gilt $f \geq a/4$ erniedrigt man a um zwei und f um eins (s wird dabei wieder angepaßt). Bei jeder Veränderung der Parameter, steigt die Abschätzung für die Wahrscheinlichkeit eines Bad-Sets, wie gerade bewiesen wurde. Man verändert die Parameter so lange, bis man entweder $a = 4, f = 1, s = 18$ oder $a = 3, f = 1, s = 12$ erreicht. Diese beiden Parametersätze stellen also eine obere Schranke für die Wahrscheinlichkeit dar.

Um eine obere Schranke für *alle* Parametermöglichkeiten mit $3 \leq a \leq 0.02C$ zu erhalten, genügt es also diese beiden Randfälle näher zu betrachten.

Benutzt man die Formel für $H(a,f,s)$ in Fakultäten ergibt sich:

$$H(4,1,18) = O(C \cdot C^{-18} \cdot C^{1}0) = O(C^{-7})$$

$$H(3,1,12) \;=\; O(C \cdot C^{-12} \cdot C^{6}) = O(C^{-5})$$

Auf jeden Fall muß die Wahrscheinlichkeit für $H(3,1,12)$ noch mit C^3 multipliziert werden, damit man die Gesamt-Wahrscheinlichkeit für alle möglichen Kombinationen an Parametern $3 \leq a \leq 0.02C$,f und s erhält.

Die Gesamt-Wahrscheinlichkeit liegt daher bei $O(1/C^2)$; das reicht für eine randomisierte Konstruktion aus.

5.6 Das Ergebnis für 6 Checker pro Contact

Es erhebt sich nun die Frage, was für eine untere Schranke für die Approximierbarkeit von E2-LIN2-3OCC durch das Wheel bewiesen werden kann.

Dazu gehen wir wie bei E2-LIN2-7OCC vor: Nehmen wir wieder an, daß das unbeschränkte E2-LIN2 keine Gleichungen der Form $x+x=z$ enthält, da die Erfüllung solcher Gleichungen unabhängig von der Belegung von x ist. Nehmen wir weiter an, daß es im E2-LIN2 n Gleichungen gibt. An jeder Gleichung sind dann genau 2 Variablen, und deshalb sind auch genau 2 Contacts aus 2 verschiedenen Wheels im E2-LIN2-3OCC beteiligt.

In einem Wheel mit C Contacts gibt es (bei 6 Checkern pro Contact) genau $10C$ Gleichungen; hinzu kommen die Gleichungen der Contacts, die das Wheel mit dem Rest des Graphen verbinden und die den Gleichungen im unbeschränkten E2-LIN2 entsprechen. Insgesamt muß es $2n$ Contacts geben. Da jeder Contact 10 zusätzliche Wheel-Gleichungen erzeugt, gibt es also im E2-LIN2-3OCC $10 \cdot 2n + n = 21n$ Gleichungen.

Wenn es im E2-LIN2 S erfüllte Gleichungen gibt, dann gibt es im E2-LIN2-3OCC nach der Umbelegung der Wheels genau $20n + S$ erfüllte Gleichungen (weil alle Gleichungen im Wheel erfüllt sind).

Laut Hastad läßt sich ein E2-LIN2 mit $16n$ Gleichungen nicht besser als 12/11 approximieren. Daraus folgt, daß sich ein E2-LIN2-3OCC mit $21 \cdot 16n = 336n$ Gleichungen nicht besser als $(320n + 12n)/(320n + 11n) = 332/331$ approximieren läßt.

Um die Anzahl der Contacts pro Wheel mit der Anzahl der Wheels insgesamt steigen zu lassen, benutzt man das gleiche Vorgehen wie bei E2-LIN2-7OCC, d.h. man vervielfacht die Gleichungen des E2-LIN2. Wie bereits vorher gezeigt, ändert das nichts an der Schranke, und man erhält dadurch eine korrekte, randomisierte Reduktion.

5.7 Verbesserungsversuche

Es ist nun die Frage, ob die randomisierte Konstruktion über das Wheel auch funktioniert, wenn man das Wheel nicht mit 6 Checkern pro Contact konstruiert, sondern mit nur 5 Checkern pro Contact.

Könnte man zeigen, daß ein Wheel mit 5 Checkern pro Contact immer noch eine korrekte, randomisierte Konstruktion liefert, würde sich die vorher berechnete untere Schranke sofort verbessern: In einem Wheel mit C Contacts gibt es dann nur noch $8.5C$ Gleichungen, was bedeutet, daß die Anzahl der Contacts immer gerade sein muß.

Wenn das E2-LIN2 n Gleichungen besitzt, dann hat das E2-LIN2-3OCC nur noch $18n$ Gleichungen (nicht mehr $21n$). Die untere Schranke würde sich damit auf 284/283 verbessern.

Leider läßt sich leicht überprüfen, daß mit den bisher benutzten Formeln ein Beweis für 5 Checker nicht möglich ist. Um eine Verbesserung zu erzielen, müssen also die Zählmethoden verbessert werden.

5.7.1 Verbesserung von $P(a, f, s)$

Zum Verständnis der folgenden Verbesserungen muß man sich klar machen, was man eigentlich zählen will, nämlich die Anzahl der *Matchings*, bei denen es mindestens 1 Bad-Set gibt.

Wenn man bei der Formel $A(a, f, s) \cdot P(a, f, s)$ den Nenner $1/M(5C)$ (bei 5 Checkern pro Contact) ausklammert, dann bleibt genau die Abschätzung für die Anzahl an Matchings mit mindestens einem Bad-Set übrig. Diese Abschätzung basiert darauf, daß man alle Matchings erfaßt, die eine vorgegebene Teilmenge mit Parametern a,f und s zu einem Bad-Set machen.

Tatsächlich wird dadurch ein Matching, das k Bad-Sets enthält, bei den bisherigen Formeln auch sicher k mal gezählt. Jedes der Bad-Sets entspricht nämlich einer anderen Teilmenge an Checkern, und die bisherige Abschätzung zählt das Matching für jede dieser Teilmengen mit.

Hat man zwei Teilmengen, die beide bei einem bestimmten Matching zu Bad-Sets werden, dann reicht es aus, dieses Matching nur einmal zu zählen.

Eine erste Verbesserung ergibt sich daraus wie folgt: Nehmen wir an, daß bei einer Teilmenge von Checkern und Contacts, ein Checker an dem Rand eines ausgewählten Fragments liegt. Wenn wir dann ein Matching betrachten, bei dem die Teilmenge ein Bad-Set wird und bei dem die Matching-Kante des Checkers eine Cut-Kante ist, dann wird dieses Matching mindestens zweimal gezählt. Läßt man nämlich den Checker samt Cut-Kante weg, dann erhält man eine neue Teilmenge, die das Matching ebenfalls mitzählt, weil durch Weglassen des Checkers nur eine Cut-Kante weniger vorhanden ist, d.h. daß die neue Teilmenge erst recht ein Bad-Set ist.

Liegt vor dem Checker noch ein Contact, dann muß die Anordnung auch ohne Contact und Checker ein Bad-Set sein, da man durch Entfernen von Contact und Checker die Differenz zwischen Contacts und Cut-Kanten nicht ändert, d.h. daß man wieder eine neue Teilmenge erhält, die das Matching schon mitzählt. Diese beiden Folgerungen sind in Abbildung 11 illustriert.

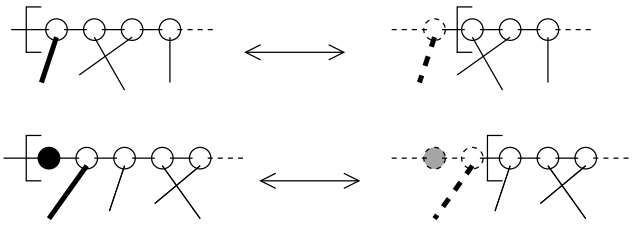


Abbildung 11: Zwei Matchings, die zweimal gezählt werden

Diese Folgerung beeinflußt die Formel für $P(a, f, s)$; anstatt $\binom{s}{i}$ zu benutzen – was der Anzahl der Möglichkeiten, Cut-Kanten auf die ausgewählten Checker zu verteilen, entspricht – reicht es auch, mit $\binom{s-2f}{i}$ zu zählen. Ein Matching, bei dem ein Checker an einem Fragmentrand eine Matching-Kante als Cut-Kante hat, ist nämlich bereits von einer kleineren Teilmenge gezählt worden.

5.7.2 Eine neue Formel für $P(a, f, s)$

Die Idee, mit der man $\binom{s}{i}$ auf $\binom{s-2f}{i}$ verbessert, läßt sich noch erweitern: Werden in einer ausgewählten Menge an Checkern zwei Endknoten für zwei Cut-Kanten nebeneinander gelegt, so ist das Matching ebenfalls schon von einer kleineren Teilmenge mit einem Fragment mehr gezählt worden (siehe Abbildung 12). Diese Teilmenge muß bei dem gleichen Matching ebenfalls ein Bad-Set sein, da sich die Anzahl der Cut-Kanten und Contacts nicht geändert hat.

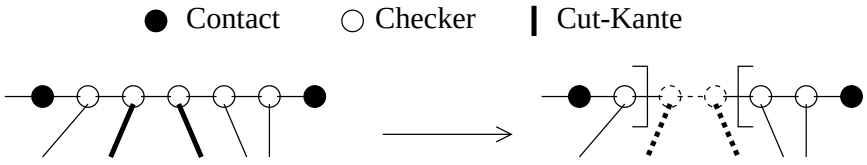


Abbildung 12: Zwei nebeneinander liegende Cut-Kanten

Diese Vorgehensweise funktioniert nur, wenn zwischen den beiden Checkern *kein* Contact liegt; wenn zwischen den Checkern ein Contact liegen würde, müßte man den Contact entfernen, und das würde die Anzahl der erlaubten Cut-Kanten *verringern*.

Nehmen wir an, wir wollen i Cut-Kanten auf s Checker verteilen. Nachdem der erste Endpunkt (s Möglichkeiten) für die Cut-Kante ausgesucht worden ist, ist mindestens eine weitere Position für die Auswahl des nächsten Endpunktes gesperrt (deshalb nur noch $(s-2)$ Möglichkeiten). Nachdem 3 Endpunkte ausgewählt wurden, kann es passieren, daß alle 3 Endpunkte zwischen 2 Contacts liegen (siehe Abbildung 13). In diesem Fall

sperren die 3 Endpunkte zusammen nur noch 2 weitere Positionen, und es gibt deshalb noch $(s-5)$ Positionen zur Auswahl für den 4. Endpunkt.

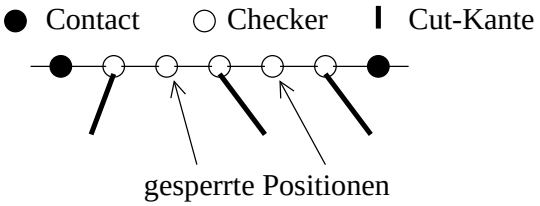


Abbildung 13: Drei Cut-Kanten zwischen zwei Contacts

Wenn man diese Überlegung als Formel ausdrückt, erhält man:

$$\begin{aligned} V(s,i) &= \frac{\overbrace{s(s-2)(s-4) \quad (s-5)(s-7)(s-9) \quad (s-10) \ \ldots}^{i \text{ Faktoren}}}{i!} \\ &< \frac{s^3 \quad (s-5)^3 \quad (s-10)^3 \ \ldots}{i!} \\ &= \frac{\left[\frac{s}{5}\left(\frac{s}{5}-1\right)\left(\frac{s}{5}-2\right)\ldots\right]^3 5^i}{i!} \\ &= \left(\frac{\left(\frac{s}{5}\right)!}{\left(\frac{s}{5}-\frac{i}{3}\right)}\right)^3 \frac{5^i}{i!} \end{aligned}$$

Drückt man s durch αC und i durch βC aus, ergibt sich durch Anwendung der Stirlingschen Formel wieder ein Ausdruck der Form $e^{C \cdot V(\alpha,\beta)+O(\ln C)}$. Für $V(\alpha,\beta)$ erhält man:

$$V(\alpha,\beta) = 3\frac{\alpha}{5}\ln\frac{\alpha}{5} - 3\left(\frac{\alpha}{5}-\frac{\beta}{3}\right)\ln\left(\frac{\alpha}{5}-\frac{\beta}{3}\right) - 5\frac{\beta}{5}\ln\frac{\beta}{5}$$

Um $\binom{s-2f}{i}$ durch $V(s-2f,i)$ ersetzen zu können, ist nun noch zu analysieren, ob die Sperrung der $2f$ Checker am Rand mit der Prämisse kollidiert, daß zwei Cut-Kanten nicht nebeneinander liegen dürfen.

Dazu müssen die verschiedenen Fälle, die an den Fragmenträndern auftreten können, überprüft werden:

- Nehmen wir an, daß der Fragmentrand aus einem oder zwei Checkern vor einem Contact besteht. In diesem Fall darf keiner der beiden Checker ein Endpunkt für eine Cut-Kante sein, da man sonst beide weglassen könnte ohne die Anzahl der

erlaubten Cut-Kanten zu verringern. Da aber nur der Fall ausgeschlossen wird, daß der äußerste Checker ein Endpunkt für eine Matching-Cut-Kante ist, wird zuviel gezählt und auf keinen Fall zu wenig.

- Hat der Fragmentrand mehr als 2 Checker vor einem Contact, dann zählt die Formel aus dem gleichen Grund zu viel.
- Besteht der Fragmentrand aus einem Contact, dann ist der Checker direkt nach dem Contact als Matching-Cut-Kanten-Endpunkt gesperrt. Wird von den übrigen vier Checkern einer ausgewählt, dann wird dadurch wieder mindestens eine weitere Position gesperrt, d.h. die Abschätzung arbeitet immer noch korrekt.

5.7.3 Eine genauere Formel für $A(a, f, s)$

Die beiden bisher benutzten Formeln für die Abschätzung von $A(a, f, s)$ sind natürlich sehr ungenau und können leicht verbessert werden.

Die einfachste Verbesserung ist für den Term $\binom{6C}{2f}$ – wenn man mit 5 Checkern pro Contact rechnet – möglich. Dazu kann man einfach die Anzahl der ausgewählten Knoten mit in die Formel hereinnehmen, d.h. man rechnet mit $\binom{s+a}{f}\binom{6C-s-a}{f}$. Die Formel erklärt sich so: Man teilt zuerst die $s+a$ ausgewählten Knoten (s Checker und a Contacts) in f Fragmente auf; es gibt dafür $\binom{s+a-1}{f-1}$ verschiedene Möglichkeiten, und zur Vereinfachung benutzt man $\binom{s+a}{f}$. Als nächstes werden die $6C-s-a$ unausgewählten Knoten (wieder Checker und Contacts) auf die f Fragmente, die als Zwischenräume dienen, verteilt. Man rechnet wieder vereinfacht mit $\binom{6C-s-a}{f}$. Jetzt legt man einen Knoten auf dem Wheel fest und beginnt mit einem ausgewählten Fragment. Das nächste Fragment ist dementsprechend unausgewählt; so wird fortgefahren, bis alle Fragmente verteilt sind. Was noch fehlt, ist die Anzahl der Möglichkeiten, den Startknoten zu bestimmen. Da jeder Knoten als Startknoten dienen kann, gibt es dafür $6C$ Möglichkeiten, d.h. man erhält als Gesamtformel:

$$6C\binom{s+a}{f}\binom{6C-s-a}{f}$$

Da der lineare Faktor $6C$ im Exponenten der e-Funktion nur als $O(\ln C)$ eingeht, kann er bei großem C vernachlässigt werden.

Auch bei dieser neuen Formel wird wieder nicht darauf geachtet, daß die Anzahl der Contacts stimmt. Es wird zwar insgesamt die richtige Zahl an Knoten ausgewählt, aber ob dabei auch wirklich genau a Contacts ausgewählt werden, ist nicht garantiert.

Die Anzahl der ausgewählten Contacts läßt sich dafür leicht in die zweite Formel $\binom{C+2f}{2f}\binom{d+2f}{2f}$ einarbeiten: Hier kann man einfach den ersten Faktor genauer als $\binom{a+f}{f}\binom{C-a+f}{f}$ berechnen. Die Erklärung ist dieselbe wie bei der Formel zuvor; man verteilt erst die ausgewählten und dann die unausgewählten Contacts. Der zusätzliche Summand

$+f$ kommt dadurch zustande, daß Fragmente in diesem Fall auch gar keinen Contact enthalten dürfen. Für die zweite Zählmethode erhält man also:

$$C\binom{a+f}{f}\binom{C-a+f}{f}$$

Der lineare Faktor C kommt, wie schon zuvor erklärt, aus der Möglichkeit, die Fragmentekombination an C verschiedenen Positionen beginnen zu lassen. Er geht im Exponenten wieder nur mit $O(\ln C)$ ein und kann deshalb bei großem C ignoriert werden.

Der Nachteil der zweiten Formel ist, daß nicht darauf geachtet wird, daß an einem Fragmentrand nicht mehr als 5 Checker angehängt werden können. Zusätzlich kann man mehr als ein ausgewähltes Fragment zwischen zwei Contacts legen, was jedoch nicht sinnvoll ist, wie später noch genauer erläutert wird.

Die Frage ist, ob man die Nachteile der zweiten Formel so weit beheben kann, daß sie zu einer generell besseren Formel wird.

Dazu wurden zuerst zwei neue Parameter eingeführt:

- h, die Anzahl der *ausgewählten* Fragmente, die *keinen* Contact enthalten.
- g, die Anzahl der *unausgewählten* Fragmente, die *keinen* Contact enthalten.

Zunächst kann man die Beobachtung machen, daß ein Fragment ohne Contact mindestens 3 Checker enthalten muß; das gilt für unausgewählte genauso wie für ausgewählte Fragmente.

Nehmen wir wieder an, wir haben ein Matching, das eine Teilmenge zu einem Bad-Set macht, die ein Contact-loses Fragment mit weniger als 3 Checkern enthält. Streicht man dieses Fragment, dann erhält man einen neue Teilmenge, die durch dieses Matching ebenfalls zu einem Bad-Set gemacht wird; dabei ist es egal, ob ein ausgewähltes oder unausgewähltes Fragment gestrichen worden ist. Da das Matching durch diese Teilmenge schon gezählt worden ist, braucht man es bei der Teilmenge davor nicht zu zählen.

Da das für alle Matchings gilt, die die vorangegangene Teilmenge zu einem Bad-Set machen, braucht man für die vorangegangene Teilmenge gar kein Matching zu zählen.

Aus diesen Überlegungen ergeben sich mehrere Verbesserungen:

- Zwischen zwei ausgewählten Contacts kann höchstens *ein* unausgewähltes Fragment liegen.
- Zwischen zwei unausgewählten Contacts kann höchstens *ein* ausgewähltes Fragment liegen.
- Zwischen einem unausgewählten und einem ausgewählten Contact kann kein ausgewähltes Fragment liegen.

Wäre eines der Kriterien nicht erfüllt, dann gäbe es ein Fragment mit weniger als 3 Checkern, und dieser Fall ist ausgeschlossen worden.

Betrachten wir nun den Fall, daß wir, wie zuvor erklärt, die Minimalbelegung von $6(a-f)$ Checkern vergrößern wollen:

- Zuerst werden die a ausgewählten Contacts auf die $(f-h)$ Fragmente mit mindestens einem Contact verteilt. Die Anzahl der Möglichkeiten beträgt vereinfacht:

$$\binom{a}{f-h}$$

- Dann werden die g Contact-losen, unausgewählten Fragmente auf die $(f-h)$ Zwischenräume zwischen den ausgewählten Fragmenten mit Contacts verteilt:

$$\binom{f-h}{g}$$

  Durch diese Auswahl werden g Zwischenräume gestrichen; die anliegenden ausgewählten Fragmente werden dadurch aneinandergehängt. Danach gibt es also nur noch $f-d-g$ Fragmente, die Contacts enthalten.

- Dann wird bestimmt, zwischen welchen Fragmenten mit Contacts sich die Contact-losen ausgewählten Fragmente befinden. Dazu verteilt man sie auf die $f-h-g$ Zwischenräume:

$$\binom{f-h-g}{h}$$

  Nach dieser Verteilung hat man $f-g$ Fragmente, die Contacts enthalten. Aus der Formel kann man erkennen, daß gelten muß: $h+g<f$.
  Daß diese Beschränkung stimmt, wird einem klar wenn man einmal versucht, eine Teilmenge zu zeichnen, bei der sie *nicht* eingehalten wird.

- Zuletzt werden die übrigen $C-a$ Contacts auf die $f-g$ Zwischenräume verteilt:

$$\binom{C-a}{f-g}$$

Die Vorgehensweise ist in Abbildung 14 verdeutlicht.

Wieder kann noch ein Faktor C angefügt werden, um eine Fragmentkombination an C verschiedenen möglichen Positionen anfangen zu lassen. Da der Faktor $6C$ im Exponenten wieder nur mit $O(\ln n)$ eingeht, wird er im weiteren weggelassen.

Durch die Einführung von h und g kann man auch genauer Abschätzen, in welchem Bereich sich s bewegen muß. Zählen wir zuerst die $(f-h)$ ausgewählten Fragmente, die Contacts enthalten; sie besitzen mindestens $5(a-(f-h))$ Checker.

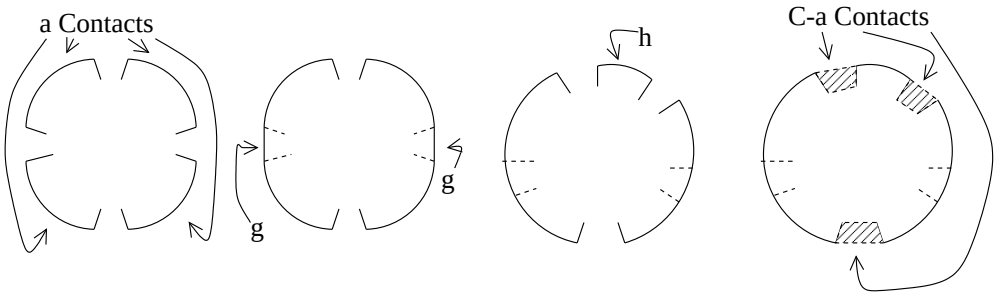


Abbildung 14: Eine Verteilung mit: $f = 5$, $g = 2$, $h = 1$

In der alten Formel sind Contact-lose ausgewählte Fragmente gar nicht berücksichtigt worden; sie wurden bei der Abschätzung mit -6 Checkern gezählt (siehe Kapitel 5.3). Jetzt ist es möglich, sie mit mindestens einem Checker zu zählen, weil man dadurch jede erlaubte Möglichkeit erzeugen kann, indem man die Fragmentgrenzen entsprechend verschiebt (siehe Abbildung 15). Das funktioniert, weil der mittlere Checker immer ausgewählt sein muß.

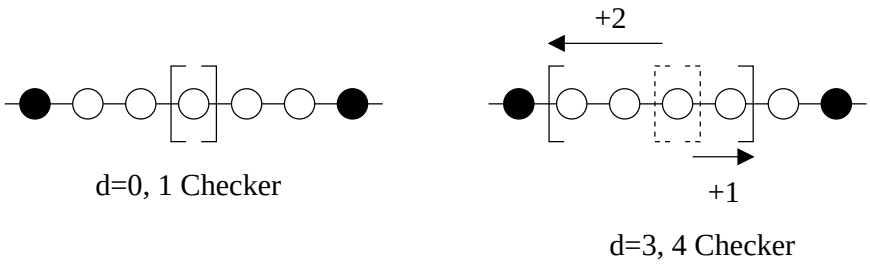


Abbildung 15: Ein ausgewähltes Fragment ohne Contacts

Der Überstand wird also mit $d = s - (5a - 5f + 6h)$ berechnet und dann mit

$$\binom{d + 2f}{2f}$$

Möglichkeiten verteilt.

Die vorhergehende Formel ist nur eine gute Abschätzung, wenn d klein ist. Es gibt aber auch eine Methode, die verschiedenen Kombinationen an Fragmenträndern bei Contact-losen Fragmenten anders zu zählen.

Jedes ausgewählte Contact-lose Fragment steuert eigentlich mindestens 3 Contacts bei; man muß also insgesamt mindestens $(5a - 5f + 8h)$ Checker ausgewählt haben.

Wenn wir den echten Überstand d berechnen, gilt also $d = s - (5a - 5f + 8h)$. Die Anzahl der Möglichkeiten, den Überstand zu verteilen, wird bei der alten Formel mit $\binom{d+2f}{2f} =$

$\binom{d+2f}{d}$ berechnet. Durch die Umformung sieht man, daß die Anzahl der Möglichkeiten mit wachsendem Überstand monoton ansteigt, d.h. heißt für eine obere Schranke kann man ein maximal mögliches d benutzen.

Die Formel kann verbessert werden, indem man wieder die Contact-losen Fragmente g und h betrachtet; für beide Fragmenttypen gibt es nur jeweils 6 Möglichkeiten, Contacts auf sie zu verteilen (siehe Abbildung 16).

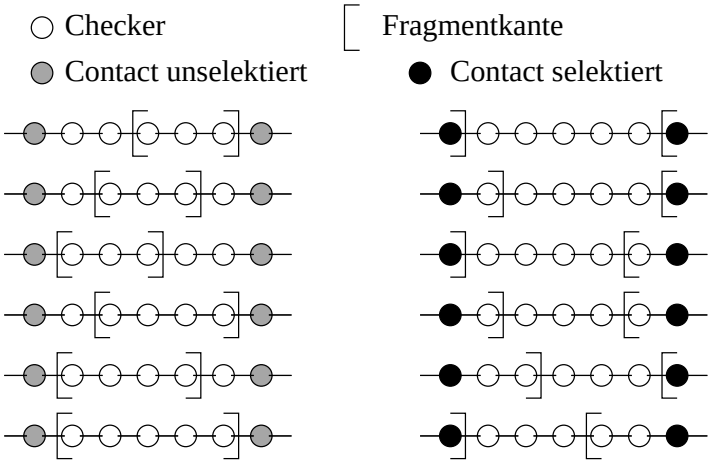


Abbildung 16: Fragmente ohne Contact

Der maximale Überstand wird deshalb nur auf die $(2f-2g-2h)$ unbeschränkten Fragmentränder verteilt, und für die Contact-losen Fragmente werden statt dessen 6^{h+g} Möglichkeiten berechnet. Es ergibt sich als Gesamtformel:

$$\binom{d+2f-2g-2h}{d} 6^{h+g} \qquad \text{mit } d = s-(5a-5f+8h)$$

Das Prinzip, die Anzahl der möglichen Konfigurationen an einem Fragmentrand zu zählen, anstatt die Checker auf die Ränder zu verteilen, läßt sich auch auf die normalen Fragmente anwenden. Es gibt $(2f-2g-2h)$ unbeschränkte Fragmentränder. An jedem davon können zwischen 0 und 5 Checker angehängt werden, d.h. es gibt pro Fragmentrand 6 Möglichkeiten. Anstatt die überstehenden Checker auf die Fragmentränder zu verteilen, kann man also auch mit

$$6^{2f-2g-2h}6^{g+h} = 6^{2f-g-h}$$

rechnen.

Die maximale Anzahl von Checkern berechnet man auf folgende Weise: Jedes der h Contact-losen ausgewählten Fragmente kann maximal 5 Checker haben. In jedes der g Contact-losen unausgewählten Fragmente können höchstens 2 Checker hineinragen (sonst

hätte das unausgewählte Fragment weniger als 3 Checker). Die übrigen $(2f - 2g - 2h)$ unbeschränkten Ränder können wieder maximal 5 Checker enthalten. Die $(f - h)$ ausgewählten Fragmente mit Contacts enthalten $5(a-(f-h))$ Checker. Zusammengenommen kann man also höchstens $s = 5a + 5f - 8g$ Checker auswählen.

Wollen wir jetzt wie, schon in Kapitel 5.3 Checker von den Fragmenträndern streichen, dann muß man allerdings mit $d = (5a + 5f - 6g) - s$ im einfachen Fall rechnen. Das liegt daran, daß bei einem g Fragment nur garantieren ist, daß der mittlere Checker nie ausgewählt ist, aber nicht mehr.

Rechnet man wieder mit 6^{g+h}, dann ergibt sich $d = (5a+5f-8g)-s$, weil jetzt garantiert ist, daß an einem g Fragment höchstens 2 Checker gezählt werden müssen.

Zusammengefaßt erhält man folgende Formel, um festzulegen, wie die Fragmente verteilt sind:

$$\binom{a}{f-h}\binom{f-h}{g}\binom{f-h-g}{h}\binom{C-a}{f-g}$$

Die Anzahl der Möglichkeiten, den Überstand auf die Fragmentränder zu verteilen, kann auf 5 verschiedene Arten berechnet werden:

$$\binom{d+2f}{d} \qquad \begin{array}{l}\text{mit } d = s - (5a - 5f + 6h)\\ \text{oder } d = (5a + 5f - 6g) - s\end{array}$$

$$\binom{d+2f-2g-2h}{d}6^{h+g} \qquad \begin{array}{l}\text{mit } d = s - (5a - 5f + 8h)\\ \text{oder } d = (5a + 5f - 8g) - s\end{array}$$

$$6^{2f-g-h}$$

Man schätzt wieder mit einer e-Funktion ab. Von den genannten fünf Berechnungsmöglichkeiten wählt man immer die aus, die den kleinsten Exponenten für einen vorgegebenen Parametersatz ergibt. Das Ergebnis dieser Berechnung multipliziert man mit der Anzahl der Möglichkeiten, die Fragmente auf die Contacts zu verteilen. Damit erhält man eine neue, genauere Formel für $A(a, f, s, g, h)$.

Mann kann jetzt wieder a, f, s, g, h durch Anteile von C ausdrücken und berechnet dann damit $A(a, f, s, g, h)P(a, f, s)$; man erhält wieder eine e-Funktion, die die Form $e^{C\cdot K+O(\ln(C))}$ hat. K hängt nur noch von den gewählten Parametern a, f, s, g, h hab. Leider stellt sich heraus, daß die genauere Abschätzung von $A(\ldots)$ immer noch nicht ausreicht, um die Korrektheit der Konstruktion zu beweisen; d.h. es gibt immer noch Parameterkombinationen, bei denen $K > 0$ ist.

5.7.4 Ein anderer Ansatz für $A(a, f, s)$

Bisher wurde A(a,f,s) auf die zwei beschriebenen Methoden abgeschätzt. Bereits bei Piotr Bermann und Marek Karpinski wird eine Verbesserung eingeführt, der eine ganz andere

Idee zugrunde liegt.

Zählt man im Fall mit 5 Checkern mit $\binom{6C}{2f}$, dann zählt man zuviel, weil man dabei die Verteilung der Contacts auf die Fragmente mitberücksichtigt. Ob ein Matching eine Teilmenge zu einem Bad-Set macht, hängt aber prinzipiell nur von der *Anzahl* der ausgewählten Contacts, nicht aber von ihrer Lage ab. Abbildung 17 macht das deutlich.

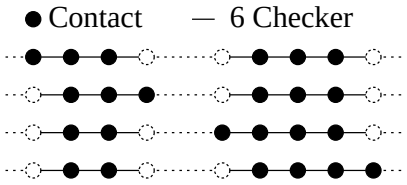


Abbildung 17: Vier Möglichkeiten, einen Contact auf 2 Fragmente zu verteilen

Also reicht es, nur $\binom{5C}{2f}$ Möglichkeiten zu zählen. Durch die Festlegung der Fragmente auf den Checkern werden auch die ausgewählten Contacts festgelegt, sofern sie nicht genau an einem Fragmentrand liegen. Welche Contacts an den Fragmenträndern ausgewählt sind oder nicht spielt aber keine Rolle, so lange die Gesamtanzahl stimmt.

Eine Lücke bei der bisherigen Argumentation stellen Fragmentkonfigurationen dar, bei denen ein einzelner Contact ein unausgewähltes Fragment bildet (siehe Abbildung 18). Alle Möglichkeiten dieser Art werden mit der beschriebenen Verbesserung nicht direkt erfaßt. Man kann wieder argumentieren, daß man alle Matchings, die diese Konfiguration zu einem Bad-Set machen, bereits in der Teilmenge mitgezählt hat, bei der der Contact ausgewählt war. Die Frage ist nur, was passiert, wenn man eine Teilmenge betrachtet, die bereits die maximal mögliche Anzahl an Contacts enthält. Hier greift die Argumentation der mehrfachen Zählung nicht, weil eine Teilmenge mit einem Contact mehr nicht gezählt wird.

Allerdings wird eine solche Möglichkeit auf eine andere Weise mitgezählt; nehmen wir wieder Abbildung 18 als Ausgangsbasis. Eine Möglichkeit, die tatsächlich mitgezählt wird, ist die Zusammenfassung der abgebildeten Checker zu einem Fragment mit den Grenzen bei Knoten 1 und 2.

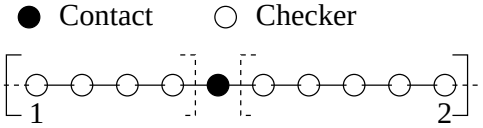


Abbildung 18: Ein einzelner unausgewählter Contact

Eine solche Anordnung würde erfordern, daß der Contact ebenfalls ausgewählt wäre; nehmen wir momentan an, daß dies so ist – auch wenn damit mehr Contacts als erlaubt

ausgewählt sind – und daß es sich bei der gesamten Anordnung um *kein* Bad-Set handelt, d.h. daß sie mehr Cut-Kanten als Contacts enthält. Entfernt man aus dieser Anordnung den einen Contact, dann hat man danach eine Cut-Kante mehr, denn man hat zwar einen Contact gestrichen, aber dadurch 2 Cut-Kanten gewonnen. D.h. damit die Anordnung nach dem Streichen des Contacts ein Bad-Set ist, muß sie auf jeden Fall davor schon eines gewesen sein; deshalb hat man wieder zu viele Matchings gezählt, aber nicht zu wenige.

Piotr Berman und Marek Karpinski haben dieses Vorgehen in der Weise beschrieben, daß sie die Contacts durch Kanten, so genannte Contact-Kanten, ersetzen. Danach enthält das Wheel nur noch Checker.

Entwickelt man diese Idee konsequent weiter, dann wird klar, daß die Ersetzung der Contacts durch Kanten noch viel weitreichendere Folgen hat: Durch dieses Modell, kann man sich die Anzahl der Contacts als künstlichen Parameter vorstellen, d.h. völlig unabhängig von der tatsächlich ausgewählten Menge an Checkern.

Findet man zu einer beliebigen (ohne die Contacts zu berücksichtigen) ausgewählten Teilmenge an Checkern ein Bad-Set, d.h. auch gleichzeitig ein Matching, was diese Teilmenge zu einem Bad-Set macht, gibt es drei Möglichkeiten:

- Die ausgewählte Teilmenge an Checkern paßt zur vorgegebenen Anzahl an Contacts; d.h. man kann mit der vorgegebenen Anzahl an Contacts die ausgewählten Contact-Kanten besetzen. Bei Contact-Kanten, die als Fragmentgrenze dienen, kann frei entschieden werden, ob sie besetzt werden sollen oder nicht (auf diese Kanten kann man überflüssige Contacts verteilen).

  In diesem Fall hat man ein Matching entdeckt, das wirklich ein Bad-Set enthält.

- Die ausgewählte Teilmenge an Checkern benötigt weniger als die angegebene Anzahl an Contacts (selbst wenn man alle Contact-Kanten, die als Fragmentgrenze dienen, mit Contacts auffüllt).

  In diesem Fall werden mehr Cut-Kanten zugelassen als eigentlich erlaubt sind, weil man mit zu vielen Contacts rechnet. Man findet dadurch Matchings, die Bad-Sets erzeugen, die in dem echten Wheel gar keine Bad-Sets sind. Da auch alle Matchings und dazugehörigen Bad-Sets gezählt werden, die weniger Cut-Kanten haben, wird aber auf jeden Fall nie zu wenig gezählt.

- Die ausgewählte Teilmenge an Checkern benötigt mehr Contacts als angegeben sind, und man hat bereits die maximale Anzahl an Contacts zur Verfügung.

  In diesem Fall müßte man die Teilmenge an einigen Contact-Kanten aufspalten und so in mehr Fragmente aufteilen als vorgegeben sind, um eine erlaubte Anordnung zu erhalten. Dadurch sinkt aber wieder nur die Anzahl der erlaubten Cut-Kanten; das bedeutet wieder nur, daß man Matchings zählt, die im echten Wheel gar keine Bad-Sets enthalten würden; alle echten Bad-Sets werden aber wieder erfaßt.

Zusammen mit dem Argument des mehrfachen Zählens von Matchings kann man die Abschätzung $\binom{5C}{2f}$ noch weiter verbessern; die bisherige Abschätzung erlaubt nämlich ausgewählte Fragmente, die aus weniger als drei Checkern bestehen.

Ein Matching, das von einer so gearteten Teilmenge gezählt worden ist, wurde aber auch sicher von der Teilmenge gezählt, bei der die Checker nicht ausgewählt waren. (Diese Teilmenge hat ein Fragment und ein oder zwei Checker weniger; siehe Abbildung 19).

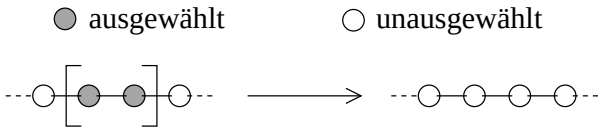


Abbildung 19: Ein ausgewähltes Fragment mit 2 Checkern

Ein ausgewähltes Fragment sollte also mindestens 3 Checker enthalten. Um diese Verbesserung in die Formel einzuarbeiten, werden einfach $2f$ Checker reserviert, bevor man die einzelnen Fragmente bestimmt. An jedes ausgewählte Fragment werden dann zwei Checker hinzugefügt, so daß ein ausgewähltes Fragment mindestens 2 Checker enthält. Als Formel ergibt sich: $\binom{5C-2f}{2f}$.

Mit der gleichen Argumentation kann man auch begründen, daß ein unausgewähltes Fragment mindestens 3 Checker enthalten muß. Man erhält dann als neue Formel $A(a, f, s) = \binom{5C-4f}{2f}$.

5.7.5 Kollisionen bei den Verbesserungen

Leider gibt es bei beiden Ansätzen $A(a, f, s)$ zu berechnen, Kollisionen mit der neuen Formel $V(a, f, s)$ für die Abschätzung von $P(a, f, s)$.

Diese Formel beruhte nämlich auf der Idee, daß man zwei nebeneinander liegende ausgewählte Checker streichen kann, wenn sie die Endpunkte von zwei Matching-Cut-Kanten sind; d.h. man ordnet diese Checker der Menge an unausgewählten Checkern zu.

Die Verbesserungen von $A(a, f, s)$ basieren aber in beiden Ansätzen auf der Idee, daß ein unausgewähltes Fragment aus mindestens 3 Checkern besteht. Damit die gerade erklärte Umbelegung erlaubt ist, müßte man aber Teilmengen mitzählen, die auch unausgewählte Fragmente mit 2 Checkern enthalten.

Wenn man den Einfluß der Verbesserungen auf die gesamte Abschätzung miteinander vergleicht, dann ist es günstiger die Verbesserungen für $A(a, f, s)$ zu behalten und $P(a, f, s)$ mit Hilfe von $\binom{s-2f}{a-2f}$ zu berechnen.

Diese einfach Verbesserung für $P(a, f, s)$ kollidiert nicht mit den Verbesserungen für $A(a, fs)$. Das ist nicht so offensichtlich ist wie man meinen könnte: Da nämlich die mini-

male Länge der ausgewählten Fragmente auf 3 nach unten begrenzt ist, muß man überprüfen, was mit Fragmenten passiert, die genau 3 Checker und eine Matching-Kante als Cut-Kante am Rand enthalten.

Vorher wurde argumentiert, daß das Matching, welches diese Teilmenge zu einem Bad-Set macht, bereits von einer Teilmenge gezählt wurde, bei der der Checker mit Matching-Cut-Kante fehlt. Da jetzt die Fragmente alle mindestens 3 Checker groß sein müssen, greift diese Argumentation nicht mehr. Statt dessen kann man leicht sehen, daß die Teilmenge, bei der das gesamte Fragment aus 3 Checkern fehlt, das betrachtete Matching schon gezählt hat, da durch Löschen des gesamten Fragmentes die Anzahl der erlaubten Cut-Kanten nur um mindestens 1 erhöht wird (siehe Abbildung 20).

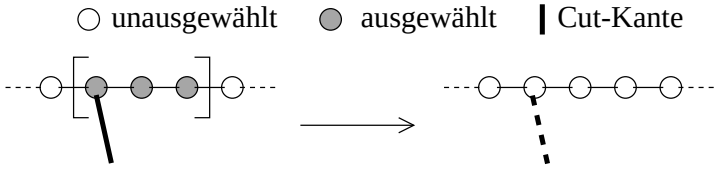


Abbildung 20: Ein ausgewähltes Fragment, das gelöscht wird

5.8 Die Lücke bei den Abschätzungen

Durch die Ersetzung der Contacts durch Contact-Kanten kann man das Sperren von Checkern für Cut-Kanten auch auf die unausgewählten Checker übertragen, d.h. man kann mit $\binom{5C-s-2f}{a-2f}$ anstatt $\binom{5C-s}{a-2f}$ bei $P(a, f, s)$ rechnen. Die Argumentation ist dieselbe: Wenn bei einem unausgewählten Checker an einem Fragmentrand die Matching-Kante eine Cut-Kante ist, dann wurde dieses Matching bereits von der Teilmenge gezählt, bei dem der Checker ausgewählt ist und deshalb die Matching-Kante keine Cut-Kante mehr ist. Der Sonderfall, daß ein Fragmentrand aus einem Contact besteht, muß durch die Ersetzung der Contacts durch Kanten nicht mehr beachtet werden.

Durch konsequente Weiterentwicklung der Idee der Contact-Kanten kann man sogar argumentieren, daß es genügt, mit $\binom{s-3f}{a-2f}$ und $\binom{5C-s-3f}{a-2f}$ zu rechnen; in diesem Fall kommt man einem Beweis sehr nahe, daß die Konstruktion mit 5 Checkern korrekt ist.

Leider hat der gesamte Ansatz mit Contact-Kanten einen entscheidenden Haken. Wie man leicht nachrechnet, liegt durch die bisherige Argumentation das Maximum der berechneten Wahrscheinlichkeit immer bei $a = 0.5$. In diesem Fall wurde bereits erklärt, wie man argumentiert, um den Fall eines einzelnen unausgewählten Contacts zu erfassen. Leider enthält die Argumentation eine weitere Lücke, die nur schwer zu erkennen ist.

Die Argumentation basiert darauf, daß man für solche isolierten Contacts eine andere Teilmenge zählt, bei der dieselben Checker ausgewählt sind und bei der mehr Cut-Kanten erlaubt sind.

Leider wird aber in Extremfällen genau *so* eine Teilmenge *nicht* gezählt. Zur genauen Erklärung gehen wir davon aus, daß das Wheel 100 Contacts hat und wir den Fall betrachten, daß 50 davon ausgewählt sind.

Nehmen wir an, daß sich alle ausgewählten Contacts in einem einzigen Fragment befinden und daß dieses Fragment genau $5(50+1) = 255$ Checker enthält, d.h. die beiden Fragmenträndern voll besetzt sind. Nehmen wir weiterhin an, daß in der Hälfte, in der die Contacts nicht ausgewählt sind, zwei weitere Fragmente aus jeweils 5 Checkern genau nebeneinander liegen und daß diese Fragmente nur durch einen unausgewählten Contact getrennt sind.

Diese Knotenmenge enthält insgesamt 265 Checker.

Die bisherige Argumentation besagt, daß diese Teilmenge von einer anderen Teilmenge mitgezählt wird, in der die beiden Fragmente mit 5 Checkern in ein Fragment mit 10 Checkern zusammengefaßt sind. Diese Annahme ist falsch.

Die gemeinte Teilmenge müß 50 Contacts enthalten und aus 2 Fragmenten bestehen; eines davon mit 255 Checkern und eines davon mit 10 Checkern; insgesamt hat sie also auch 265 Checker.

Durch die Bedingung $s \leq 5(a+f) = 5(50+2) = 260$ wird aber diese Teilmenge nie gezählt.

Die ganze Idee der Contact-Kanten enthält das Problem, daß der Zusammenhang zwischen der Anzahl der ausgewählten Checker und der ausgewählten Contacts ausgehebelt wird. Dieser Zusammenhang wird nur noch künstlich hergestellt, indem man beide Zahlen vorgibt und nur Teilmengen betrachtet, bei denen die Bedingung $5(a-f) \leq s \leq 5(a+f)$ erfüllt ist.

Ein großer Teil der Verbesserungen in diesem Modell beruht aber darauf, daß man s und f unabhängig voneinander ändern kann; wenn man z.B. ein Fragment mit zwei unausgewählten Checkern streicht, dann hat man danach ein Fragment weniger und zwei ausgewählte Checker mehr. Durch diese Umbelegung hat man aber eventuell die Bedingung $s \leq 5(a+f)$ gebrochen und erhält deshalb eine neue Teilmenge, die gar nicht gezählt worden ist.

Leider ist es nicht gelungen, dieses Dilemma zu beseitigen; ein Beweis, daß die Konstruktion auch mit 5 Checkern funktioniert, steht also noch aus. Der Versuch, einen Gegenbeweis zu führen stößt, auf genau die gleichen Probleme, d.h. es ist schwierig die Bad-Sets so zu zählen, daß man dabei keine Matchings ausläßt und trotzdem keine Matchings doppelt zählt.

Literatur

[1] M. Ajtai. Recursive construction for 3-regular expanders. *Combinatorica*, 14, 1994.

[2] N. Alon and V. D. Milman. λ_1, isoperimetric inequalities for graphs, and superconcentrators. *Journal of Combinatorial Theory. Series B*, 38:73–88, 1985.

[3] Noga Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

[4] S. Arora. The approximability of NP-hard problems. In *28th Annual Symposium on Theory of Computing*, pages 337–348, 1998.

[5] S. Arora and C. Lund. Hardness of approximations. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, 1995.

[6] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, 1998. Preliminary version in FOCS'92.

[7] S. Arora and S. Safra. Probabilistic checking of proofs: a new characterization of NP. *Journal of the ACM*, 45(1):70–122, 1998.

[8] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer-Verlag, Berlin, 1999.

[9] L. Babai. Trading group theory for randomness. In *Proceedings of the 17th Annual Symposium on Theory of Computing*, pages 421–429, 1985.

[10] L. Babai. Transparent proofs and limits to approximations. In *First European Congress of Mathematicians*, pages 31–91. Birkhäuser, Basel, 1994.

[11] M. Bellare, S. Goldwasser, and M. Sudan. Free bits, PCPs and non-approximability – Towards tight results. *SIAM Journal on Computing*, 27:804–915, 1998.

[12] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson. Multi-prover interactive proofs: How to remove intractability assumptions. In *Proceedings of the 20th Annual Symposium on Theory of Computing*, pages 113–131, Chicago, 1988.

[13] P. Berman and M. Karpinski. On some tighter inapproximability results (extended abstract). *Lecture Notes in Computer Science*, 1644:200 ff, 1999.

[14] Bela Bollobas. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1:311–316, 1980.

[15] Bela Bollobas. The isoperimetric number of random regular graphs. *European Journal of Combinatorics*, 9:241–244, 1988.

[16] Anne Condon. The complexity of the max word problem and the power of one-way interactive proof systems. *Computational Complexity*, 3:292–305, 1993.

[17] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symposium on Theory of Computing*, pages 151–158, New York, 1971. Association for Computing Machinery.

[18] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan. Structure in approximation classes. *SIAM Journal on Computing*, 28:1759–1782, 1999.

[19] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Interactive proofs and the hardness of approximating cliques. *Journal of the ACM*, 43(2):268–292, 1996. Preliminary version in FOCS'91.

[20] Uriel Feige and Michel X. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the Third Israel Symposium on Theory of Computing and Systems*, 1995.

[21] Lance Fortnow, John Rompel, and Michael Sipser. On the power of multi-prover interactive protocols. *Theoretical Computer Science*, 134(2):545–557, November 1994.

[22] Michael R Garey and David S Johnson. *Computers and Intractibility, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, 1979.

[23] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42:1115–1145, 1995.

[24] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Journal on Computing*, 418:186–208, 1989.

[25] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*, pages 627–636, 1996.

[26] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual Symposium on Theory of Computing*, pages 1–10, 1997.

[27] V. Heun, W. Merkle, and U. Weigand. Proving the PCP-Theorem. In E.W. Mayr, H.J. Prömel, and A. Steger, editors, *Lectures on Proof Verification and Approximation Algorithms*, pages 83–160. Springer-Verlag, Berlin, 1998.

[28] S. Hougardy, H.J. Prömel, and A. Steger. Probabilistically checkable proofs and their consequences for approximation algorithms. *Discrete Mathematics*, 136:175–223, 1994.

[29] H. Karloff and U. Zwick. A 7/8-approximation algorithm for MAX 3SAT. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS-97)*, pages 406–415, Los Alamitos, October 1997. IEEE Computer Society Press.

[30] R.M. Karp. Reducibility among combinatorial problems. In J.W. Thatcher and R.E. Miller, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[31] L. A. Levin. Universal sorting problems. *Problems of Information Transmission*, 9:265–266, 1973.

[32] Lubotzky, Phillips, and Sarnak. Ramanujan graphs. *Combinatorica*, 8, 1988.

[33] Margulis. Explicit group-theoretical constructions of combinatorial schemes and their application to the design of expanders and concentrators. *Problems of Information Transmission*, 24:39–46, 1988.

[34] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[35] L. J. Stockmeyer. Planar 3-colorability is polynomial complete. *ACM SIGACT News*, 5(3):19–25, 1973. NCPY.

[36] L. Trevisan, G.B. Sorkin, M. Sudan, and D.P. Williamson. Gadgets, approximation, and linear programming. In *Proceedings of the 37th Symposium on Foundations of Computer Science*, pages 617–626, 1996.