6 Algebraische Methoden

6.1 Der Satz von Schwartz-Zippel

In diesem Abschnitt betrachten wir Polynome mit mehreren Variablen $X_1, \ldots, X_n$ über einem Körper K. Ein solches Polynom f kann man als endliche Summe von Termen der Form

$$a_{\ell_1,\ldots,\ell_n} \cdot X_1^{\ell_1} X_2^{\ell_2} \cdots X_n^{\ell_n}$$

schreiben, wobei $a_{\ell_1,\ldots,\ell_n}$ ein Element von K ist, und in verschiedenen Termen die Exponentenfolge $\ell_1, \ldots, \ell_n$ verschieden ist. Der *Grad* eines solchen Terms ist $\ell_1 + \cdots + \ell_n$, der Grad $\deg(f)$ des Polynoms f ist das Maximum der Grade seiner Terme. Die Polynome von Grad 0 sind die Terme $a = aX_1^0 X_2^0 \cdots X_n^0$ mit $a \in K - \{0\}$. Das Nullpolynom hat keinen Term, sein Grad ist $-\infty$.

Beispiele für Polynome über dem Körper $K = \mathbb{Z}_5$:

$$X_1^2 X_2^2 + 4X_1 X_4^3 + 2X_2^3 X_3^3, \quad X_1 X_2 + 2X_2 + X_3, \quad X_1^3 + 3X_1^2 X_2, \quad 3, \quad 0$$

mit den Graden 6, 2, 3, 0 und $-\infty$. Polynome kann man addieren, subtrahieren und multiplizieren, und vereinfachen, indem man Terme, die zu derselben Exponentenfolge $\ell_1, \ldots, \ell_n$ gehören, zusammenfasst.

Ein Vektor $(a_1, \ldots, a_n) \in K^n$ heißt eine Nullstelle von f, wenn sich beim Einsetzen von a_i für X_i in f und Ausrechnen der Wert $f(a_1, \ldots, a_n) = 0$ ergibt. Beispielsweise ist $(1, 3)$ eine Nullstelle von $X_1^3 + 3X_1^2 X_2$ (über $K = \mathbb{Z}_5$).

Folgendes ist leicht zu beweisen (und zumindest für den Körper der reellen Zahlen aus der Schule bekannt, siehe auch Fakt 5.4.2(a)):

Lemma 6.1
Wenn f ein Polynom mit einer Variablen $X = X_1$ über einem Körper K ist, mit Grad $d \geq 0$, so besitzt f höchstens d Nullstellen.

Wir können dies auf Polynome mit n Variablen verallgemeinern:

Satz 6.2 ***Schwartz-Zippel***
Wenn f ein Polynom mit n Variablen über einem Körper K ist, mit Grad $d \geq 0$, und $S \subseteq K$ eine endliche Menge ist, dann besitzt f in S^n höchstens $d|S|^{n-1}$ Nullstellen.

Diese Aussage kann nicht verbessert werden: Betrachte den Körper $K = \mathbb{Z}_p$ für eine Primzahl p. Seien $1 \leq d \leq s \leq p$. Setze $S = \{0, \ldots, s-1\}$. Das Polynom $f = (X_1 - 0)(X_1 - 1)(X_1 - 2) \cdots (X_1 - (d-1))$ (mit n möglichen Variablen $X_1, \ldots, X_n$, von denen aber $X_2, \ldots, X_n$ in f nicht vorkommen) hat in S^n die ds^{n-1} Nullstellen $(i, a_2, \ldots, a_n)$, mit $0 \leq i < d$ und $a_2, \ldots, a_n \in S$ beliebig.

Beweis von Satz 6.2: Induktion über $d \geq 0$. Der Induktionsanfang $d = 0$ ist trivial, weil dann $f(X_1, \ldots, X_n)$ ein konstantes Polynom, aber nicht das Nullpolynom ist, also überhaupt keine Nullstelle hat.

Sei also jetzt $d \geq 1$, und sei als Induktionsvoraussetzung angenommen, dass die Aussage für Polynome mit Graden zwischen 0 und $d-1$ richtig ist.

Weil $d \geq 1$, enthält f mindestens eine Variable. Wir nehmen an, dass X_1 in f vorkommt. (Sonst benennt man die Variablen passend um.) Wir wählen $\delta \geq 1$ so, dass X_1^δ die höchste Potenz ist, mit der X_1 in f vorkommt, und klammern X_1^δ aus allen Termen aus, in denen dieser Faktor vorkommt. Dies liefert eine Zerlegung

$$f(X_1, \ldots, X_n) = X_1^\delta \cdot q(X_2, \ldots, X_n) + r(X_1, \ldots, X_n),$$

wobei q nicht das Nullpolynom ist, in q die Variable X_1 nicht vorkommt, der Grad von $q(X_2, \ldots, X_n)$ maximal $d - \delta < d$ ist, und in $r(X_1, \ldots, X_n)$ der Faktor X_1 nur mit Exponenten $< \delta$ auftaucht.

Wir definieren

$$A := \{(a_1, \ldots, a_n) \in S^n \mid q(a_2, \ldots, a_n) = 0\}$$

und

$$B := \{(a_1, \ldots, a_n) \in S^n \mid q(a_2, \ldots, a_n) \neq 0 \wedge f(a_1, \ldots, a_n) = 0\}.$$

Man sieht sofort, dass

$$f(a_1, \ldots, a_n) = 0 \;\Rightarrow\; (a_1, \ldots, a_n) \in A \cup B.$$

Es genügt also zu zeigen, dass $|A \cup B| \leq |A| + |B| \leq d|S|^{n-1}$ ist.

Dazu beobachten wir: Nach Induktionsvoraussetzung ist

$$|A| \leq |S| \cdot (d - \delta)|S|^{n-2} = (d - \delta)|S|^{n-1}.$$

(Die erste Komponente a_1 ist in S frei wählbar; für $(a_2, \ldots, a_n)$ benutzt man die Induktionsvoraussetzung.) Die Zahl $|B|$ schätzen wir folgendermaßen ab: Für jeden Vektor $(a_2, \ldots, a_n) \in S^{n-1}$ mit $q(a_2, \ldots, a_n) \neq 0$ ist $f(X_1, a_2, \ldots, a_n) = q(a_2, \ldots, a_n)X_1^\delta + r(X_1, a_2, \ldots, a_n)$ ein Polynom in der einen Variablen X_1, vom Grad genau $\delta \geq 1$. Insbesondere ist dieses Polynom nicht das Nullpolynom, es hat also (nach Lemma 6.1)

maximal δ Nullstellen in K. Wir summieren über alle $(a_2, \ldots, a_n) \in S^{n-1}$ und erhalten, dass $|B| \leq |S|^{n-1} \cdot \delta$ gilt. Wenn wir die Abschätzungen für A und B addieren, folgt $|A| + |B| \leq d|S|^{n-1}$, also hat f maximal $d|S|^{n-1}$ Nullstellen in S^n. Das ist die Induktionsbehauptung. □

6.2 Vergleich von Polynomprodukten

Im folgenden sei K irgendein (endlicher oder unendlicher) Körper.

Problemstellung: Gegeben seien Polynome $f_1, \ldots, f_r$ und $g_1, \ldots, g_s$ über K mit Variablen $X_1, \ldots, X_n$. Sei $f = f_1 \cdot \ldots \cdot f_r$ und $g = g_1 \cdot \ldots \cdot g_s$. Gilt $f = g$?

Um die Schwierigkeit der Fragestellung zu illustrieren, betrachten wir ein Beispiel. Die Polynome $f_1, \ldots, f_{2n}$ seien

$$X_1^s + 1, \ldots, X_n^s + 1, X_1^s - 1, \ldots, X_n^s - 1,$$

die Polynome $g_1, \ldots, g_n$ seien

$$X_1^{2s} - 1, \ldots, X_n^{2s} - 1.$$

Dabei kann s eine Zahl mit $O(n)$ Bits sein.[1] Wir „sehen“ (aufgrund der Formel $(X_i^s + 1)(X_i^s - 1) = X_i^{2s} - 1$), dass die Produkte der beiden Folgen gleich sind. Algorithmisch ist dies aber nicht ganz so einfach zu behandeln. (Die Terme könnten anders angeordnet sein, und zudem mit weniger offensichtlichen anderen Faktoren multipliziert sein.) Der nächstliegende deterministische Algorithmus wird die beiden Seiten ausmultiplizieren und dann vergleichen. Im Beispiel entstehen dann aber exponentiell viele Terme, der Aufwand ist also immens.

Tatsächlich ist für das Problem „Vergleich von Polynomprodukten“ kein deterministischer Algorithmus bekannt, der polynomielle Laufzeit hat. Wir geben einen sehr einfachen randomisierten Algorithmus an.

Es sei

$$d = \max\{\deg(f_1) + \cdots + \deg(f_r), \deg(g_1) + \cdots + \deg(g_s)\}.$$

[1] Die rechnerinterne Darstellung von Polynomen für dieses Berechnungsproblem sollte man sich folgendermaßen vorstellen: Die Terme $a_{\ell_1,\ldots,\ell_n} \cdot X_1^{\ell_1} X_2^{\ell_2} \cdots X_n^{\ell_n}$, die nicht 0 sind, werden als $(\ell_1, \ell_2, \ldots, \ell_n, a_{\ell_1,\ldots,\ell_n})$ geschrieben, wobei die Zahlen binär dargestellt sind. Alle diese Terme sind als Liste gegeben. Auf diese Weise lassen sich auch Polynome mit sehr großen Graden sehr kompakt darstellen. Zum Beispiel hat das Polynom $X_1^{2^n-1} + X_2 + \cdots + X_n$ Darstellungsgröße $\Theta(n^2)$. Wenn man einen Term mit großen Exponenten für einen gegebenen Input $(a_1, \ldots, a_n) \in K$ auswerten möchte, benutzt man schnelle Exponentiation wie in Algorithmus 5.1.19. Die Größe $\text{size}(f_1, \ldots, f_r; g_1, \ldots, g_s)$ der Eingabe ist die gesamte (Bit-)Länge der Darstellung der beiden Listen von Polynomen.

(Dann ist offenbar $\deg(f-g) \leq d$.) Wir legen eine beliebige endliche Menge[2,3] $S \subseteq K$ mit $|S| \geq 2d$ fest. Nun wählen wir $\boldsymbol{a} = (a_1, \ldots, a_n) \in S^n$ uniform zufällig und berechnen

$$b = f_1(\boldsymbol{a}) \cdot \ldots \cdot f_r(\boldsymbol{a}) \quad \text{und} \quad c = g_1(\boldsymbol{a}) \cdot \ldots \cdot g_s(\boldsymbol{a}).$$

(Es wird in die einzelnen Faktoren eingesetzt und ausgewertet, dann in K muliplizierd.)

Ausgabe: $[b \neq c]$ (d. h. 1, falls $b \neq c$, und 0, falls $b = c$).

Algorithmus 6.1 ***Polynomprodukt***
Input: Polynome $f_1 \ldots, f_r$ und $g_1, \ldots, g_s$ über K mit Variablen $X_1, \ldots, X_n$.
Methode:
1 $d \leftarrow \max\{\deg(f_1) + \cdots + \deg(f_r), \deg(g_1) + \cdots + \deg(g_s)\}$;
2 Wähle endliche Menge $S \subseteq K$ mit $|S| \geq 2d$;
3 Wähle $\boldsymbol{a} = (a_1, \ldots, a_n) \in S^n$ zufällig;
4 **return** $[f_1(\boldsymbol{a}) \cdot \ldots \cdot f_r(\boldsymbol{a}) \neq g_1(\boldsymbol{a}) \cdot \ldots \cdot g_s(\boldsymbol{a})]$.

Es ist leicht zu sehen, dass Algorithmus 6.1 nur eine polynomielle Anzahl von Körperoperationen benötigt. Sogar wenn die Exponenten in den Polynomen groß sind, ist – mit schneller Exponentiation, siehe Algorithmus 5.1.19 – die Anzahl der K-Operationen nur linear in der Gesamtbitlänge dieser Exponenten.

Wir analysieren die Fehlerwahrscheinlichkeit.

1. Fall: $f = g$. – Dann ist $f(\boldsymbol{a}) = g(\boldsymbol{a})$ für jedes $\boldsymbol{a}$, also ist die Ausgabe immer 0.

2. Fall: $f \neq g$. – Dann gilt:

$$\text{Ausgabe ist } 0 \;\Leftrightarrow\; (f-g)(\boldsymbol{a}) = 0.$$

Das heißt: Die Ausgabe ist fehlerhaft genau dann wenn der zufällig gewählte Vektor $\boldsymbol{a}$ eine Nullstelle von $f-g$ ist. Nun ist $f-g$ ein Polynom von einem Grad ≥ 0 (weil $f \neq g$) und höchstens d. Nach dem Satz von Schwartz-Zippel (Satz 6.2) hat $f-g$ also höchstens $d|S|^{n-1}$ Nullstellen in S^n. Die Wahrscheinlichkeit, die falsche Ausgabe 0 zu erhalten, ist demnach höchstens

$$\frac{d|S|^{n-1}}{|S^n|} = \frac{d}{|S|} \leq \frac{1}{2}.$$

[2]Falls $|K| < 2d$, muss man mit aus der Algebra bekannten Methoden eine Körpererweiterung $K' \supseteq K$ mit $|K'| \geq 2d$ konstruieren und in K' rechnen.

[3]Es ist nicht nötig, die Elemente von S aufzulisten. Beispiel: Wenn $K = \mathbb{Z}_p$, könnte $S = \{0, \ldots, t-1\}$ sein, für eine Zahl $t < p$. Deshalb sind sehr große Grade d auch hier kein Hindernis.

Es handelt sich bei Algorithmus 6.1 also um einen Monte-Carlo-Algorithmus mit einseitigem Fehler, Fehlerschranke $d/|S|$. Wenn wir ihn ℓ-mal mit unabhängig gewählten Vektoren $\boldsymbol{a}$ durchführen, erhalten wir eine Fehlerschranke von $(d/|S|)^\ell$.

6.3 Polynomdeterminanten

Ganz ähnlich wie beim Vergleich von Polynomprodukten ist es bei Determinanten von Matrizen, deren Einträge Polynome sind. Gegeben sei also eine $m \times m$-Matrix

$$A(X_1, \ldots, X_n) = \begin{pmatrix} f_{11} & \cdots & f_{1m} \\ \vdots & & \vdots \\ f_{m1} & \cdots & f_{mm} \end{pmatrix}$$

mit Polynomeinträgen $f_{ij} = f_{ij}(X_1, \ldots, X_n)$. Wir nehmen an, dass $d_0 \geq \deg(f_{ij})$ für alle i, j gilt.[4] Aus der linearen Algebra kennt man die (Leibniz-Formel für die) Determinante:

$$\det(A) = \sum_{\sigma \in S_m} \operatorname{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdot \ldots \cdot f_{m,\sigma(m)}.$$

Dabei ist S_m die Menge aller Permutationen σ der Menge $\{1, \ldots, m\}$, und $\operatorname{sign}(\sigma) \in \{-1, +1\}$ das Vorzeichen der Permutation σ.

Problem 1: Gegeben eine Polynommatrix A. Ist $\det(A) = 0$, also das Nullpolynom?

Problem 2: Gegeben eine Polynommatrix A und eine Folge $g_1, \ldots, g_s$ von Polynomen. Ist $\det(A) = g_1 \cdot \ldots \cdot g_s$?

Die Summe in der Definition der Determinante hat $n!$ Terme. Daher ist es normalerweise nicht möglich, das Polynom $\det(A)$ explizit zu berechnen. Auch hier hilft Randomisierung.

Sei $d = d_0 m$ oder eine andere obere Schranke für $\deg(\det(A))$. Wähle eine endliche Menge $S \subseteq K$ mit $|S| \geq 2d$. Nun wähle zufällig eine Folge $\boldsymbol{a} = (a_1, \ldots, a_n) \in S^n$. Berechne

$$\det(A(\boldsymbol{a})) = \det \begin{pmatrix} f_{11}(\boldsymbol{a}) & \cdots & f_{1m}(\boldsymbol{a}) \\ \vdots & & \vdots \\ f_{m1}(\boldsymbol{a}) & \cdots & f_{mm}(\boldsymbol{a}) \end{pmatrix}$$

wie folgt: Erst wird $\boldsymbol{a}$ in jede Komponente eingesetzt, und es wird jede für sich ausgewertet, dann wird die Determinante in K berechnet (Gauss-Elimination, $O(m^3)$ Körper-Operationen). Die Ausgabe ist $[\det(A(\boldsymbol{a})) \neq 0] \in \{0, 1\}$. Ganz analog zur Analyse von Algorithmus 6.1 sieht man folgendes: Wenn $\det(A)$ das Nullpolynom ist, dann ist die Ausgabe 0, und wenn $\det(A) \neq 0$, dann gilt $\mathbf{Pr}(\text{Ausgabe ist } 0) \leq d/|S| \leq \frac{1}{2}$.

[4] In Anwendungen sind die Einträge oft lineare Polynome, also $d_0 = 1$.

Ein analog gebauter Algorithmus löst auch Problem 2.

Eine graphentheoretische Anwendung der Polynomdeterminanten werden wir in Abschnitt 6.4 kennenlernen.

6.4 Perfektes Matching in bipartiten Graphen

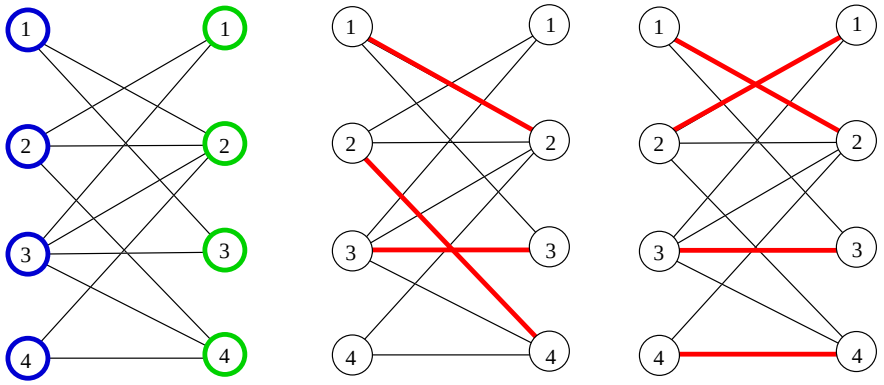

Abbildung 1: Links: Ein bipartiter Graph, $n = 4$ Knoten auf jeder Seite, $m = 11$ Kanten. Mitte: Ein nicht perfektes Matching. Rechts: Ein perfektes Matching.

In diesem Abschnitt betrachten wir bipartite Graphen $G = (U, V, E)$. Dabei ist $U = V = \{1, \ldots, n\}$ und $E \subseteq U \times V$. Eine Menge $M \subseteq E$ heißt ein *Matching* („paarweise Zuordnung“) in G, wenn für $(i, j), (i', j') \in M$ gilt: $i = i' \Leftrightarrow j = j'$. In Worten: Kanten in M sind entweder knotendisjunkt oder gleich. Ein Matching M heißt *perfekt*, wenn M aus n Kanten besteht – dann ist jeder Knoten in U genau einem Knoten in V zugeordnet.

Problem 1: Gegeben sei ein bipartiter Graph G.
Frage: Besitzt G ein perfektes Matching?

Problem 2: Gegeben sei ein bipartiter Graph G.
Aufgabe: Berechne ein perfektes Matching, wenn dies möglich ist.

Es gibt Standardalgorithmen, die diese Probleme lösen. Sie beruhen auf Flussberechnungen oder verwandten Verfahren. (Siehe Vorlesung „Effiziente Algorithmen“ im Master Informatik.) Wir stellen einen randomisierten Algorithmus für das Entscheidungsproblem vor. Dieser hat allerdings schlechtere Laufzeiten als die entsprechenden Flussalgorithmen. Der Vorteil liegt darin, dass er parallelisierbar ist, was für die Flussalgorithmen (vermutlich) nicht gilt.

Definition 6.3
$G = (U, V, E)$ sei ein bipartiter Graph auf 2 mal n Knoten. Die **Edmonds-Matrix** $\boldsymbol{A_G}$ ist eine $n \times n$-Matrix mit Einträgen

$$f_{ij} = \left\{ \begin{array}{ll} X_{ij}, & \text{falls } (i,j) \in E, \\ 0, & \text{falls } (i,j) \notin E, \end{array} \right\}, \quad \text{für } 1 \leq i,j \leq n.$$

Dabei sind die Variablen X_{ij}, $1 \leq i,j \leq n$, alle verschieden.[5]

Beispiel: Der bipartite Graph G aus Abb. 1 hat die Adjazenzmatrix bzw. Edmonds-Matrix

$$M_G = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \quad \text{bzw.} \quad A_G = \begin{pmatrix} 0 & X_{12} & X_{13} & 0 \\ X_{21} & X_{22} & 0 & X_{24} \\ X_{31} & X_{32} & X_{33} & X_{34} \\ 0 & X_{42} & 0 & X_{44} \end{pmatrix}.$$

Satz 6.4 ***Edmonds***
G hat ein perfektes Matching $\Leftrightarrow$ $\det(A_G) \neq 0$.

Die Polynomdeterminante $\det(A_G)$ wird dabei über einem beliebigen Körper K berechnet. Der Satz sagt also, dass man nur testen muss, ob $\det(A_G)$ das Nullpolynom ist, um die Existenz eines perfekten Matchings zu testen.

Beweis von Satz 6.4: Aus der Leibniz-Formel für die Determinante folgt:

$$\det(A_G) = \sum_{\sigma \in S_n} \underbrace{\operatorname{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)}}_{=:t_\sigma}. \tag{1}$$

Alle Terme in dieser Summe, die einen Faktor $f_{i,\sigma(i)} = 0$ haben, fallen heraus. Ein gegenseitiges Auslöschen anderer Terme ist unmöglich. Mit anderen Worten: Ein Summand $\operatorname{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)}$ ist genau dann in der Summe $\det(A_G)$ enthalten, wenn $f_{i,\sigma(i)} = X_{i,\sigma(i)}$ für alle i ist, und das heißt, dass E alle Kanten $(i, \sigma(i))$ enthält.

Daraus folgt: Die Summe in (1) ist genau dann nicht leer, wenn es eine Permutation σ gibt, für die E alle Kanten $(i, \sigma(i))$ enthält. Dies ist äquivalent zur Existenz eines perfekten Matchings. □

Beispiel: Beim bipartiten Graph G aus Abb. 1 hat $\det(A_G)$ den Term $X_{13}X_{21}X_{34}X_{42}$, entsprechend dem perfekten Matching $M = \{(1,3), (2,1), (3,4), (4,2)\}$.

[5] Achtung: Wir haben die Notation geändert. n bezeichnet die Knotenanzahl. Die Anzahl der Variablen ist n^2. In A_G kommen aber nur $m = |E|$ viele Variablen X_{ij} tatsächlich vor.

Algorithmus 6.2 ***Perfektes Matching, bipartit***
Input: Bipartiter Graph $G = (U, V, E)$ mit n Knoten auf beiden Seiten.
Methode:
1 Wähle Primzahl $p > 2n$; // Körper $\mathbb{Z}_p$, $S = \mathbb{Z}_p$
2 Bilde die Edmonds-Matrix A_G (eine $n \times n$-Matrix);
3 Wähle $\boldsymbol{a} = (a_{11}, \ldots, a_{1n}, \ldots, a_{n1}, \ldots, a_{nn}) \in \mathbb{Z}_p^{n \times n}$ zufällig;
4 $B \leftarrow A_G(\boldsymbol{a})$; // bilde $A_G(\boldsymbol{a})$ durch Einsetzen von $\boldsymbol{a}$ in A_G
5 **return** $[\det(B) \neq 0]$. // Gauss-Elimination über $\mathbb{Z}_p$

Wie vorher sehen wir: Wenn G kein perfektes Matching hat, also $\det(A_G)$ das Nullpolynom ist, dann gilt $\det(A_G(\boldsymbol{a})) = 0$ für alle $\boldsymbol{a}$, also ist die Antwort auf jeden Fall 0. Wenn G ein perfektes Matching hat, also $\det(A_G)$ nicht das Nullpolynom ist, dann gilt nach dem Satz von Schwartz-Zippel (Satz 6.2):

$$\mathbf{Pr}(\text{Antwort ist } 0) \leq \frac{\deg(\det(A_G))}{|\mathbb{Z}_p|} = \frac{n}{|\mathbb{Z}_p|} < \frac{1}{2}.$$

Bemerkung: (1) Das Bilden der Edmonds-Matrix im Algorithmus ist überflüssig; man erhält die Matrix $A_G(\boldsymbol{a})$ direkt aus der Adjazenzmatrix M_G, indem man jeden 1-Eintrag in M_G durch ein Zufallselement aus $\mathbb{Z}_p$ ersetzt (und die 0-Einträge stehen lässt).
(2) Anstatt in $\mathbb{Z}_p$ könnte man auch in $\mathbb{Q}$ rechnen; man setzt dann z. B. $S = \{1, \ldots, 2n\}$. Der Nachteil hierbei ist, dass die Zahlen in Zwischenergebnissen zwischen $\Omega(n \cdot \log n)$ und $O((n \cdot \log n)^2)$ Bits haben, so dass die einzelnen arithmetischen Operationen sehr teuer sind.
(3) Man kann zeigen, dass man mit polynomiell vielen Prozessoren die Determinante einer $n \times n$-Matrix mit $O((\log n)^2)$ Runden von parallel ausgeführten Ringoperationen berechnen kann (*Algorithmus von Samuelson/Berkowitz*). Damit kann man auch die Existenz eines perfekten Matchings in paralleler Zeit $O((\log n)^2)$ [Körperoperationen] testen – aber nur randomisiert. Für dieses Problem war bis 2016 kein schneller deterministischer paralleler Algorithmus bekannt.[6]

Problem 2, die Konstruktion eines perfekten Matchings, lässt sich wie folgt lösen: Zunächst testet man, ob G ein perfektes Matching besitzt. Nur im positiven Falle fährt man fort wie folgt. Setze $G' = (U, V, E') = G$. Man behandelt die Kanten $(i, j) \in E'$

[6] Aktualisierung 2018: Die Arbeit „Stephen A. Fenner, Rohit Gurjar, Thomas Thierauf: Bipartite perfect matching is in quasi-NC. STOC 2016. S. 754–763“ präsentiert einen deterministischen parallelen Algorithmus, der die Existenz eines perfekten Matchings mit einem „fast effizienten“ parallelen Algorithmus testet. Als effizient gelten normalerweise Algorithmen, die mit $n^{O(1)}$ („polynomiell vielen“) Prozessoren und paralleler Rechenzeit $(\log n)^{O(1)}$ auskommen. Der neue Algorithmus benutzt $n^{O(\log n)}$ („quasipolynomiell viele“) Prozessoren und $O((\log n)^2)$ parallele Zeit.

nacheinander, und verändert dabei eventuell den Graphen G'. Die Untersuchung von Kante $e = (i, j) \in E'$ verläuft so: Teste, ob $G'' = (U, V, E' - \{e\})$ ein perfektes Matching besitzt. Wenn dies der Fall ist, streiche e aus E'. Am Ende teste, ob E' ein perfektes Matching ist. Falls ja, wird E' ausgegeben, falls nein, beginne von vorne (wie bei Las-Vegas-Algorithmen üblich). Die Fehlerwahrscheinlichkeit bei einem Versuch kann man auf $1/n$ senken, indem man für jeden Test Algorithmus 6.2 $\log(|E| \cdot n)$-mal wiederholt.

Leider ist diese Methode zum Finden von perfekten Matchings wieder iterativ ($m = |E|$ Runden), also nicht leicht parallelisierbar. Effiziente parallele Algorithmen benötigen weitere Techniken, die in Abschnitt 6.7 vorgestellt werden, im Kontext von Matchings in beliebigen ungerichteten Graphen.

6.5 Textsuche (String-Matching) mit Fingerprinting

6.5.1 Textvergleich

In diesem Abschnitt geht es um Algorithmen für Texte. Wir nehmen dabei o. B. d. A. stets an, dass wir ein Alphabet $\Sigma = \{0, \ldots, u-1\}$ benutzen, dessen Buchstaben natürliche Zahlen sind.

Zunächst betrachten wir ein Verfahren zum Vergleich zweier gleich langer Wörter $a = (a_1, \ldots, a_n)$ und $b = (b_1, \ldots, b_n)$. Wir wählen eine Primzahl $p > u$ und betrachten die Polynome

$$f_a = a_1 + a_2 X + a_3 X^2 + \cdots + a_n X^{n-1} \text{ und } f_b = b_1 + b_2 X + b_3 X^2 + \cdots + b_n X^{n-1},$$

jeweils über dem Körper $\mathbb{Z}_p$. Offenbar gilt: $f_a = f_b$ genau dann wenn $a = b$. Für $h = f_a - f_b$ gibt es zwei Fälle:

1. Fall: $a = b$. – Dann ist h das Nullpolynom; für jedes $r \in \mathbb{Z}_p$ gilt $f_a(r) = f_b(r)$.
2. Fall: $a \neq b$. – Dann ist h nicht das Nullpolynom. Da $\deg(h) \leq n - 1$, hat h nach Lemma 6.1 nicht mehr als $n - 1$ Nullstellen. Also gibt es höchstens $n - 1$ Elemente $r \in \mathbb{Z}_p$ mit $f_a(r) = f_b(r)$.

Die Idee ist also, ein r aus $\mathbb{Z}_p$ zufällig zu wählen und zu testen, ob $f_a(r) = f_b(r)$ gilt. Die Werte $f_a(r)$ bzw. $f_b(r)$ können als (kurze) „Fingerabdrücke“ von $a = (a_1, \ldots, a_n)$ und $b = (b_1, \ldots, b_n)$ aufgefasst werden, die es gestattet, diese Wörter zu unterscheiden (wenn sie verschieden sind).

Algorithmus 6.3 ***Textvergleich mit Fingerprinting***
Input: $a = (a_1, \ldots, a_n)$ und $b = (b_1, \ldots, b_n)$ aus Σ^n, mit $\Sigma = \{0, \ldots, u-1\}$.
Methode:
1 Wähle eine Primzahl $p > \max\{u, 2n\}$;
2 Wähle zufällig ein r aus $\{0, \ldots, p-1\}$;
3 $\mathrm{fp}_a := (a_1 + a_2 \cdot r + a_3 \cdot r^2 + \cdots + a_n \cdot r^{n-1}) \bmod p$;
4 $\mathrm{fp}_b := (b_1 + b_2 \cdot r + b_3 \cdot r^2 + \cdots + b_n \cdot r^{n-1}) \bmod p$;
5 **if** $\mathrm{fp}_a = \mathrm{fp}_b$ **then return** 0 **else return** 1.

Die Polynomauswertungen in Zeilen 3 und 4 bewerkstelligt man in linearer Zeit mit dem bekannten Horner-Schema. Insgesamt kostet dieser Algorithmus also Zeit $O(n)$. Wenn $a = b$ ist, wird auf jeden Fall 0 ausgegeben. Wenn $a \neq b$ ist, gilt

$$\mathbf{Pr}(\text{Ausgabe ist } 0) = \frac{\#(\text{Nullstellen von } h = f_a - f_b)}{p} \leq \frac{n-1}{p} < \frac{1}{2}.$$

Interessant ist noch die folgende Beobachtung: Der Algorithmus ist durchführbar, selbst wenn sich die Daten a und b nicht am selben Ort (im selben Computer) befinden. Es genügt, am jeweiligen Ort die Fingerprints fp_a und fp_b zu berechnen und einen der beiden zu übermitteln. Dabei ist für eine Fehlerwahrscheinlichkeit von $1/2^\ell$ die Übermittlung von nur $\ell \cdot 2\log(\max\{u, 2n\})$ Bits nötig – viel weniger als wenn man die $n \log u$ Bits des gesamten Textes schicken würde.

6.5.2 Textsuche (Pattern matching, Algorithmus von Rabin-Karp)

Im Fall der Textsuche hat man es mit folgender Problemstellung zu tun:

Gegeben sind:

- „Text“ $t = (t_1, \ldots, t_n) \in \Sigma^n$ und
- „Muster“ $a = (a_1, \ldots, a_m) \in \Sigma^m$.

Die Frage ist, ob a als Teilwort $(t_i, \ldots, t_{i+m-1})$ in t vorkommt, und falls ja, an welcher Stelle bzw. welchen Stellen i.

Beispiel: Das Muster `bra` kommt im Text `abrakadabra` an den Stellen $i = 2$ und $i = 9$ vor.

Es muss gesagt werden, dass es für dieses sehr wichtige Problem auch hocheffiziente deterministische Algorithmen gibt. Dennoch ist die Betrachtung randomisierter Algorithmen interessant. Diese können auch leicht adaptiert werden, wenn es um

mehrdimensionale „Texte“ und „Muster“ geht oder wenn das Muster unbestimmte Buchstabenpositionen hat („wildcards“).

Um (für $1 \leq i \leq n - m + 1$) a mit $(t_i, \ldots, t_{i+m-1})$ zu vergleichen, wollen wir die „Fingerabdruckspolynome“

$$f_i = f_i(X) = t_i X^{m-1} + t_{i+1} X^{m-2} + \cdots + t_{i+m-2} X + t_{i+m-1}$$

und

$$g = g(X) = a_1 X^{m-1} + a_2 X^{m-2} + \cdots + a_{m-1} X + a_m$$

über $\mathbb{Z}_p$ benutzen. Wie in Abschnitt 6.5.1 sieht man: Wenn $a \neq (t_i, \ldots, t_{i+m-1})$ ist, sind die Polynome f_i und g nicht identisch, also ist das Differenzpolynom $h_i = f_i - g$ nicht das Nullpolynom. Weil der Grad von h_i nicht größer als $m - 1$ ist, hat h_i nach Lemma 6.1 höchstens $m - 1$ Nullstellen in $\mathbb{Z}_p$. Wir folgern:

$$|\{r \mid 0 \leq r < p \wedge h_i(r) = 0\}| \leq m - 1. \tag{2}$$

Wir wählen hier $p > knm$, für ein beliebig festzusetzendes k (das auch von n, m abhängen darf). Wenn wir nun r aus $\{0, \ldots, p-1\}$ zufällig wählen, folgt aus (2):

$$\mathbf{Pr}_r(\exists i \leq n - m + 1 \colon a \neq (t_i, \ldots, t_{i+m-1}) \wedge f_i(r) = g(r)) \leq \frac{(n-m)m}{p} < \frac{1}{k}. \tag{3}$$

Was soll nun der Vorteil dieses Verfahrens sein? Wir müssen $n - m + 2$ Polynomauswertungen vornehmen. Auf naive Weise implementiert kostet dies Zeit $O(nm)$, und das ist ebenso langsam wie der naive Textsuchalgorithmus.

Der Trick ist, dass sich die Berechnung der Fingerabdrücke beschleunigen lässt. Aus

$$\begin{aligned} f_i(r) &= t_i \cdot r^{m-1} + \quad t_{i+1} \cdot r^{m-2} + \cdots + t_{i+m-2} \cdot r \quad + t_{i+m-1} \qquad \text{und} \\ f_{i+1}(r) &= \qquad\quad t_{i+1} \cdot r^{m-1} + \cdots + t_{i+m-2} \cdot r^2 \quad + t_{i+m-1} \cdot r + t_{i+m} \end{aligned} \tag{4}$$

erhalten wir

$$f_{i+1}(r) = (f_i(r) - t_i \cdot r^{m-1}) \cdot r + t_{i+m}.$$

Das Körperelement r^{m-1} können wir in Zeit $O(\log m)$ vorab berechnen. Dann kann man aus $f_i(r)$ in Zeit $O(1)$ den nächsten Wert $f_{i+1}(r)$ berechnen. Als Algorithmus ergibt sich zusammengefasst folgendes:

Algorithmus 6.4 ***Textsuche mit Fingerprinting, Rabin-Karp***
Input: $a = (a_1, \ldots, a_m) \in \Sigma^m$ und $t = (t_1, \ldots, t_n) \in \Sigma^n$, $n \geq m$; Zahl $k \geq 2$.
Aufgabe: Finde die Positionen $i \in \{1, \ldots, n-m+1\}$ mit $a = (t_i, \ldots, t_{i+m-1})$.
Methode:

```
1    Wähle eine Primzahl p > nmk;
2    Wähle zufällig ein r aus {0, ..., p − 1};
3    f ← (t_1 · r^{m−1} + t_2 · r^{m−2} + ··· + t_{m−1} · r + t_m) mod p;    // Horner-Schema
4    g ← (a_1 · r^{m−1} + a_2 · r^{m−2} + ··· + a_{m−1} · r + a_m) mod p;    // Horner-Schema
5    i ← 1;
6    j ← m;
7    z ← r^{m−1} mod p;
8    if f = g then print(i);
9    while j < n do
10       j ← j + 1;
11       f ← ((f − t_i · z) · r + t_j) mod p;
12       i ← i + 1;
13       if f = g then print(i).
```

Wir benennen die Eigenschaften dieses Algorithmus.

- Der Rechenaufwand ist $O(m)$ am Anfang und $O(1)$ in jedem der $n-m$ Schleifendurchläufe, insgesamt also $O(n)$.
- Mit (4) zeigt man Folgendes (durch Induktion über $i = 2, \ldots, n-m+1$): In Schleifendurchlauf für $(t_i, \ldots, t_{i+m-1})$ (Inhalt von `i` zu Beginn: $i-1$, nach Zeile 12: i) erhält die Variable `f` in Zeile 11 den Wert $f_i(r)$; dieser wird dann in Zeile 13 auf Gleichheit mit `g` getestet, das den Fingerprint g von a enthält. Wenn $a = (t_i, \ldots, t_{i+m-1})$ ist, muss sich hier Gleichheit ergeben und i wird ausgegeben; wenn $a \neq (t_i, \ldots, t_{i+m-1})$ ist, ist die Wahrscheinlichkeit, dass i ausgegeben wird, höchstens $(m-1)/(knm) < 1/(kn)$.
- Die Wahrscheinlichkeit, dass es ein i mit $a \neq (t_i, \ldots, t_{i+m-1})$ gibt, so dass i ausgegeben wird, ist höchstens $1/k$.

Wenn man nur das erste Vorkommen von a in t finden möchte, kann man den Algorithmus folgendermaßen variieren: Wenn i ausgegeben wird, wird die Schleife unterbrochen und es wird direkt geprüft, ob $a = (t_i, \ldots, t_{i+m-1})$ ist. Falls dies so ist, ist man fertig, falls nicht, wird die Bearbeitung der Schleife wieder aufgenommen. Der modifizierte Algorithmus kann niemals ein falsches Resultat liefern (es handelt sich also um einen Las-Vegas-Algorithmus), der erwartete zusätzliche Aufwand für diese Tests ist $m \cdot (1 + \frac{1}{k})$ Vergleiche zwischen Buchstaben. – Der direkte Kontrollvergleich

ist problematisch, wenn man *alle* Vorkommen des Musters finden will. Das Muster könnte $\omega(n/m)$-mal in t vorkommen; dann wäre der Vergleichsaufwand $\omega(n)$, also nicht linear.

Bemerkung: (a) Die beschriebene Technik lässt sich auf höhere Dimensionen verallgemeinern. Für $d = 2$ wäre zum Beispiel der „Text“ eine Bitmap aus $m_1 \times m_2$ Pixeln, das Muster eine Bitmap aus $n_1 \times n_2$ Pixeln. Man soll herausfinden, ob das Muster im Text irgendwo auftaucht. Dies lässt sich mit Aufwand $O(m_1 \cdot m_2)$ bewerkstelligen. (Dies ist eine interessante Übungsaufgabe. Wenn man das Muster horizontal verschiebt, kommen ja etwa n_1 neue Bildpunkte hinzu und n_1 viele verschwinden. Wie kann man die Aktualisierung der entsprechenden Polynomwerte in $O(1)$ Zeit realisieren?)

(b) Wenn das Muster „wildcards“ oder „don't care“-Buchstaben, also unbestimmte Positionen, enthält, kann man eine Variante von Algorithmus 6.4 benutzen. Dabei ist für jede unbestimmte Buchstabenposition pro Runde eine konstante Anzahl von arithmetischen Operationen nötig.

6.6 $*$ Äquivalenztest für Read-Once-Branchingprogramme

Wir schicken einige Bemerkungen zu azyklischen gerichteten Graphen („directed acyclic graphs“, „DAGs“) $G = (V, E)$ voraus. Knoten mit Ausgangsgrad 0 heißen *Senken*. Jeder DAG besitzt eine *topologische Sortierung* der Knoten, das ist eine lineare Anordnung der Knoten als $v_1, \ldots, v_N$, so dass für alle Kanten (v_k, v_ℓ) gilt: $k > \ell$. (Eine solche Anordnung erhält man, indem man Tiefensuche auf G ausführt und die Knoten nach *aufsteigenden fin-Nummern* sortiert, ähnlich wie in der AuD-Vorlesung im Kapitel „Tiefensuche“.) Diese Anordnung macht Konstruktionen und Beweise „durch Induktion über die Knotenanordnung“ möglich: Um eine Behauptung $\varphi(v)$ für alle Knoten $v \in V$ zu beweisen, beweist man $\varphi(v)$ für die Senken (Induktionsanfang) und dann beweist man, dass aus $\varphi(w)$ für alle Nachfolger w von v auch $\varphi(v)$ folgt. Entsprechend sehen Definitionen „durch Induktion über die Knotenanordnung“ aus.

Definition 6.5
Ein **Branchingprogramm** („Verzweigungsprogramm“, „Binäres Entscheidungsdiagramm“) $G = (V, E, s)$ für n Boolesche Variablen $x_1, \ldots, x_n$ ist ein azyklischer gerichteter Graph $G = (V, E)$, mit einem ausgezeichneten (Start-)Knoten $s \in V$ und folgenden weiteren Spezifikationen: (i) Jeder Knoten, der keine Senke ist („innerer Knoten“), ist mit einer Variablen x_i beschriftet und hat zwei Ausgangskanten, eine 0-Kante und eine 1-Kante. (ii) Jede Senke ist mit 0 oder mit 1 beschriftet. (Man kann auch, ohne das Modell zu ändern, alle 0-Senken identifizieren und alle 1-Senken identifizieren, so dass es nur eine oder zwei Senken gibt. Dann heißt die 0-Senke t_0 und die 1-Senke t_1.)

Bemerkung: Bei manchen (von Hand entworfenen) Branchingprogrammen wird s die einzige Quelle des Graphen (V, E) sein, also der einzige Knoten mit Eingangsgrad 0. Dies wird in der Definition aber nicht gefordert. Man beachte auch, dass keineswegs alle Variablen $x_1, \ldots, x_n$ als Knotenmarkierungen in G vorkommen müssen.

Beispiel:

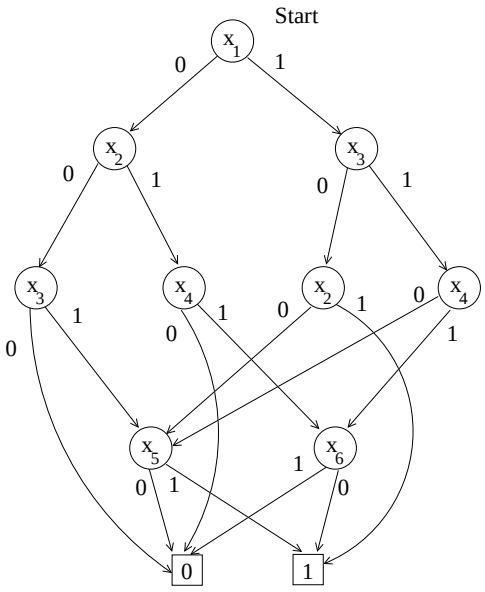

Abbildung 2: Ein Branchingprogramm mit Variablen $x_1, \ldots, x_6$. Der Startknoten ist markiert. Die Senken t_0 und t_1 sind als entsprechend markierte Quadrate dargestellt.

Definition 6.6
Ein Branchingprogramm $G = (V, E, s)$ über den Booleschen Variablen $x_1, \ldots, x_n$ definiert eine n-stellige Boolesche Funktion $f_G \colon \{0,1\}^n \to \{0,1\}$. Gegeben $a = (a_1, \ldots, a_n)$, wird $f(a)$ wie folgt durch das Ablaufen eines Wegs in G ermittelt: Starte in Knoten s. Wenn in Knoten v angekommen, wobei v ein innerer Knoten mit Beschriftung x_i ist, folge der a_i-Kante aus v zum nächsten Knoten. Wenn eine Senke t_b mit $b \in \{0,1\}$ erreicht wurde, ist $f(a) = b$.

Bemerkung: Im Extremfall besteht G nur aus einer Senke t_b. Diese ist dann auch der Startknoten, und die berechnete Funktion f_G ist die Konstante b.

Eine (offensichtlich äquivalente) einfache Beschreibung der berechneten Funktion f_G ist die folgende: Gegeben sei Input a. Wenn der innere Knoten v mit der Variablen x_i beschriftet ist, dann lösche die $\overline{a}_i$-Kante aus v. Auf diese Weise bleibt ein einziger Weg mit Startknoten s übrig. Dieser endet in einer Senke t_b. Dann ist $f_G(a) = b$.

Beispiel:

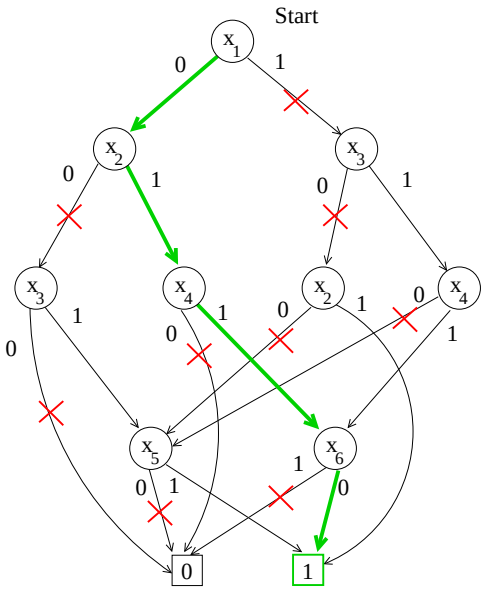

Abbildung 3: Im Branchingprogramm aus Abb. 2 wurden passend zum Input $a = (0, 1, 1, 1, 1, 0)$ Kanten gestrichen; ein Weg vom Startknoten zur 1-Senke bleibt übrig (grün). Dieser Weg würde auch zur Eingabe $a' = (0, 1, 0, 1, 0, 0)$ passen, da die Werte a_3 und a_5 gar nicht abgefragt werden. Es gilt also $f_G(a) = f_G(a') = 1$.

Hilfreich ist aber noch eine andere Beschreibung von f_G. Hierzu definieren wir eine ganze Menge von Funktionen, eine für jeden Knoten $v \in V$: f_v ist diejenige Boolesche Funktion, die von G berechnet wird, wenn man v zum Startknoten erklärt. Für einen festen Input a hängen die Funktionswerte dann auf die folgende Weise zusammen:

(i) Wenn $v = t_b$ eine b-Senke ist, dann ist $f_v(a) = b$.

(ii) Wenn v ein innerer Knoten ist, mit 0-Nachfolger v_0 (also entlang der 0-Kante) und 1-Nachfolger v_1 (entlang der 1-Kante), und wenn v mit x_i markiert ist, dann ist

$$f_v(a) = (a_i \wedge f_{v_1}(a)) \vee (\overline{a}_i \wedge f_{v_0}(a)).$$

(Ein Beweis durch Induktion über die Knotenanordnung zeigt die Korrektheit dieser Beschreibung. Festlegung (i) ist offensichtlich korrekt. Wenn die Rechnung im inneren x_i-Knoten v startet und $a_i = 0$ ist, geht man zum 0-Nachfolger v_0 und wertet G von

dort aus weiter mit a aus. Dies liefert nach I.V. gerade den Wert $f_{v_0}(a)$. Wenn $a_i = 1$ ist, erhält man den Wert $f_{v_1}(a)$. Genau dies wird durch die Formel in (ii) ausgedrückt, die „*if* a_i *then* $f_{v_1}(a)$ *else* $f_{v_0}(a)$“ bedeutet.)

Es ergibt sich die folgende etwas abstraktere Beschreibung von f_G, die mit ganzen Booleschen Funktionen operiert. Durch Induktion über die Knotenanordnung können wir für jeden Knoten $v \in V$ die Funktion $f_v \colon \{0,1\}^n \to \{0,1\}$ kompakt beschreiben.

(i) Wenn $v = t_b$ eine b-Senke ist, dann ist f_v die konstante b-Funktion.

(ii) Wenn v ein innerer Knoten ist, mit 0-Nachfolger v_0 und 1-Nachfolger v_1, und wenn v mit x_i markiert ist, dann ist

$$f_v = (x_i \wedge f_{v_1}) \vee (\overline{x}_i \wedge f_{v_0}).$$

Dann ist $f_G = f_s$ für den Startknoten s.

Beispiel: In Abb. 4 ist unser Beispiel-BP mit Knotennamen A, B, C, D, E, F, H,

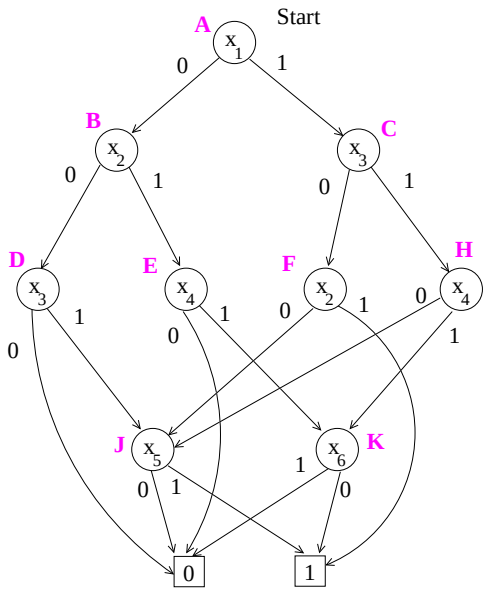

Abbildung 4: Das Branchingprogramm aus Abb. 2 mit Knotennamen A, B, C, D, E, F, H, J, K.

J, K dargestellt. Wir haben, aufgeschrieben in der Reihenfolge einer topologischen

Anordnung:[7]

$$
\begin{aligned}
f_{\mathrm{K}} &= \overline{x}_6 \\
f_{\mathrm{J}} &= x_5 \\
f_{\mathrm{H}} &= x_4\overline{x}_6 \vee \overline{x}_4 x_5 \\
f_{\mathrm{F}} &= (x_2 \wedge 1) \vee \overline{x}_2 x_5 = x_2 \vee \overline{x}_2 x_5 \\
f_{\mathrm{E}} &= x_4\overline{x}_6 \vee (\overline{x}_4 \wedge 0) = x_4\overline{x}_6 \\
f_{\mathrm{D}} &= x_3 x_5 \vee (\overline{x}_3 \wedge 0) = x_3 x_5 \\
f_{\mathrm{C}} &= x_3 f_{\mathrm{H}} \vee \overline{x}_3 f_{\mathrm{F}} = x_3(x_4\overline{x}_6 \vee \overline{x}_4 x_5) \vee \overline{x}_3(x_2 \vee \overline{x}_2 x_5) = \ldots \\
f_{\mathrm{B}} &= x_2 f_{\mathrm{E}} \vee \overline{x}_2 f_{\mathrm{D}} = \ldots \\
f_{\mathrm{A}} &= f_G = x_1 f_{\mathrm{C}} \vee \overline{x}_1 f_{\mathrm{B}} = \ldots
\end{aligned}
$$

Fakt 6.7
Zu jeder Booleschen Funktion $g\colon \{0,1\}^n \to \{0,1\}$ gibt es ein Branchingprogramm G_g mit $g = f_{G_g}$. Man kann sogar zusätzlich fordern, dass die Variablen in der Reihenfolge $x_1, \ldots, x_n$ gelesen werden.

Der *Beweis* ist nicht schwer. Man legt einen vollständigen Binärbaum mit n Levels von inneren Knoten an, wobei die 2^{i-1} Knoten auf Level i mit x_i markiert sind. Der x_1-Knoten auf Level 1 ist der Startknoten. Für $1 \leq i < n$ hat jeder x_i-Knoten ein 0-Kind und ein 1-Kind auf Level $i+1$. Die x_n-Knoten haben die Senken t_0 und t_1 als mögliche Nachfolger. Diese Nachfolger können so eingestellt werden, dass Eingabe $a \in \{0,1\}^n$ zum Erreichen der $g(a)$-Senke $t_{g(a)}$ führt. Das Branchingprogramm hat $2^n + 1$ Knoten ($2^n - 1$ innere Knoten und zwei Senken.)

Der Nachteil des beschriebenen allgemeinen Branchingprogramms ist natürlich, dass es sehr groß ist. Manche n-stellige Funktionen haben auch viel kleinere Branchingprogramme. Eine interessante Eigenschaft dieser Branchingprogramme ist, dass auf jedem Berechnungsweg jede Variable höchstens einmal vorkommt. Wir interessieren uns besonders für solche.

Definition 6.8
Ein **Read-Once-Branchingprogramm** (Read-Once-BP) $G = (V, E, s)$ ist ein Branchingprogramm, das bei der Berechnung des Wertes für eine beliebige Eingabe a das Bit a_i höchstens einmal liest. (Äquivalent ist: Auf jedem Weg vom Startknoten zu einer der Senken kommt jede Variable maximal einmal als Beschriftung vor.)

[7] Der Konvention folgend, schreiben wir das Zeichen $\wedge$ meist nicht. „$x_2 f_{\mathrm{E}}$“ bedeutet also „$x_2 \wedge f_{\mathrm{E}}$“.

In diesem Abschnitt betrachten wir einen (Nicht-)Äquivalenztest für Read-Once-BPe. Das bedeutet, dass zu zwei gegebenen Read-Once-BPen G und G' zu entscheiden ist, ob sie dieselbe Funktion darstellen. Wieso ist das interessant? Man stelle sich eine Situation vor, in der zwei Darstellungen für eine Funktion gegeben sind: eine als Spezifikation, die andere als (neu entworfener) Schaltkreis, oder zwei Darstellungen als Schaltkreis, einer alt, klassisch erprobt, und der andere ein neuer Entwurf. Angenommen, wir haben Algorithmen, die solche Darstellungen in Branchingprogramme transformieren, sogar in Read-Once-BPe. Um zu testen, ob der neue Entwurf korrekt ist, müssen wir herausfinden, ob die beiden so erzeugten Read-Once-BPe dieselbe Funktion darstellen. (Ansätze dieser Art sind die einfachsten Bausteine in der Hardwareverifikation.)

Definition 6.9
Zwei Branchingprogramme G und G' für dieselbe Variablenmenge $x_1, \ldots, x_n$ heißen *äquivalent*, in Zeichen $G \sim G'$, wenn $f_G = f_{G'}$ gilt.

Das *Äquivalenzproblem für Read-Once-Branchingprogramme* besteht darin, zu zwei vorgelegten Read-Once-BPen G und G' zu entscheiden, ob $G \sim G'$ gilt oder nicht. Wir präsentieren im Folgenden einen Monte-Carlo-Algorithmus mit einseitigem Fehler für das (Nicht-)Äquivalenzproblem. (Es sei angemerkt, dass man keinen deterministischen Polynomialzeitalgorithmus für dieses Problem kennt.)

Die Grundidee ist „Arithmetisierung der Logik“. Gegeben sein ein beliebiger Körper $K = (K, +, \cdot, 0, 1)$. Wie kann man die logischen Operationen $\wedge$, $\vee$ und $\neg$ durch Operationen über dem Körper K so darstellen, dass auf dem Definitionsbereich $\{0,1\} \subseteq K$ derselbe Effekt entsteht? Klar: $b \wedge c = b \cdot c$ und $\neg b = 1 - b$, für $b, c \in \{0,1\}$. Weiter erhält man mit den deMorgan-Regeln: $b \vee c = 1 - (1-b) \cdot (1-c) = 1 - b - c + b \cdot c$, für $b, c \in \{0,1\}$. Für unsere Zwecke wichtig ist noch die *Fallunterscheidung* mit 3 Bits:

$$\textit{if } b \textit{ then } c \textit{ else } d \text{ oder } (b \wedge c) \vee (\overline{b} \wedge d),$$

die sich für $b, c, d \in \{0,1\}$ durch $b \cdot c + (1-b) \cdot d$ darstellen lässt.

Bemerkung: Allgemeiner kann man jede Boolesche Funktion auf n Variablen $x_1, \ldots, x_n$ als Polynom über K mit Variablen $X_1, \ldots, X_n$ darstellen. Beispiele sind:

- $x_1 \wedge x_2$ wird durch $X_1 X_2$ dargestellt.
- $\neg x_1$ wird durch $1 - X_1$ dargestellt.
- $x_1 \vee x_2$ wird durch $X_1 + X_2 - X_1 X_2$ dargestellt.

- $x_1 \oplus x_2$ (exklusives oder) wird durch $X_1(1-X_2)+(1-X_1)X_2 = X_1+X_2-2X_1X_2$ dargestellt.
- $x_1 \oplus x_2 \oplus x_3$ wird durch

  $$X_1 + X_2 + X_3 - 2X_1X_2 - 2X_1X_3 - 2X_2X_3 + 4X_1X_2X_3$$

  dargestellt. (Wenn keines oder zwei der Eingabebits 1 sind, ergibt sich 0, wenn genau ein Eingabebit 1 ist, ergibt sich 1, wenn alle drei 1 sind, ergibt sich $1+1+1-2-2-2+4 = 6$. – Überlegen Sie: Was ist die Darstellung von $x_1 \oplus \cdots \oplus x_5$, von $x_1 \oplus \cdots \oplus x_n$?)
- *if* x_1 *then* x_2 *else* x_3 wird durch $X_1X_2 + (1-X_1)X_3 = X_3 + X_1X_2 - X_1X_3$ dargestellt.

Was sollen die Koeffizienten 2 und 4 in den Polynomen in einem beliebigen Körper bedeuten? Nun, ganze Zahlen $k \in \mathbb{Z}$ können in K auf natürliche Weise interpretiert werden, als $\hat{k} \in K$:

$$\hat{k} = \underbrace{1 + \cdots + 1}_{k\text{-mal, Addition in } K} \quad \text{für } k \geq 1;$$

dann ist $\hat{k}$ als additives Inverses von $\widehat{-k}$ auch für negative $k \in \mathbb{Z}$ definiert.

Zu einem gegebenen BP G mit n Booleschen Variablen definieren wir nun, durch Induktion über die Knotenanordnung, n-stellige Polynome p_v über Variablen $X_1, \ldots, X_n$ mit Koeffizienten aus K, die sich auf $\{0,1\}^n$ wie die oben diskutierten Funktionen f_v verhalten sollen. Ein BP G mit Senken t_0 und t_1 und Startknoten s sei gegeben.

(i) p_{t_0} ist das Nullpolynom; p_{t_1} ist das konstante Polynom 1.

(ii) Wenn v ein innerer Knoten ist, mit Beschriftung x_i, mit 0-Nachfolger v_0 und 1-Nachfolger v_1, dann ist

$$p_v := X_i \cdot p_{v_1} + (1 - X_i) \cdot p_{v_0}.$$

(iii) p_G ist p_s, für den Startknoten s von G.

Lemma 6.10
Für $a \in \{0,1\}^n$ gilt $f_G(a) = p_G(a)$.

(Man beachte, dass in dem Ausdruck $f_G(a)$ die Eingabebits eigentlich Wahrheitswerte darstellen, anhand derer Entscheidungen gefällt werden, in dem Ausdruck $p_G(a)$ dagegen die Elemente 0 und 1 aus K, mit denen gerechnet wird.)

Beweis (Idee): Man zeigt durch Induktion über die Knotenanordnung, dass $p_v(a) = f_v(a)$ gilt, für alle $a \in \{0,1\}^n$. Dies ist mit der oben diskutierten induktiven Definition von f_v reine Routine. □

Nun ist der Plan für unseren Äquivalenztest klar: Wenn G und G' gegeben sind, testen wir mit dem schon bekannten Ansatz aus den vorherigen Abschnitten, ob $p_G = p_{G'}$ gilt oder nicht, und schließen dann auf f_G und $f_{G'}$. Es sind allerdings noch zwei Dinge zu klären:

- Nach Lemma 6.10 gilt: $p_G = p_{G'} \Rightarrow f_G = f_{G'}$. Aber gilt auch die Umkehrung $f_G = f_{G'} \Rightarrow p_G = p_{G'}$? Wenn es vorkommen könnte, dass $f_G = f_{G'}$ gilt, aber $p_G \neq p_{G'}$, würde der Plan sofort fehlschlagen: Wir würden die Polynome auf Identität testen und mit großer Wahrscheinlichkeit Ungleichheit feststellen, obwohl die Booleschen Funktionen gleich sind. Man denke etwa an die (verschiedenen) Polynome $X_1 - X_1^2X_2$ und $X_1^2 - X_1X_2$, die beide die Boolesche Funktion $x_1\overline{x}_2$ berechnen.
- Wie kann man p_G effizient auswerten?

Wir arbeiten diese Punkte nacheinander ab. Zuerst stellen wir fest, dass die Read-Once-Eigenschaft dazu führt, dass Polynome wie $X_1 - X_1^2X_2$ nicht als p_G auftreten können.

Definition 6.11
Ein Polynom $p(X_1, \ldots, X_n)$ heißt *multilinear*, wenn in jedem Term $a_{\ell_1,\ldots,\ell_n} X_1^{\ell_1} \cdots X_n^{\ell_n}$ von p die Exponenten $\ell_1, \ldots, \ell_n$ aus der Menge $\{0,1\}$ kommen.

Beispiel: $5X_1X_2X_4 + 2X_3 + 1$ ist multilinear, $X_1^2 + X_1X_2$ nicht.

Lemma 6.12
Wenn G ein Read-Once-BP mit Variablen $x_1, \ldots, x_n$ ist, dann ist p_G *multilinear* (und hat daher Grad höchstens n).

Beweis: Wir zeigen durch Induktion über die Knotenreihenfolge, dass alle p_v multilinear sind. (Daraus folgt, dass $p_G = p_s$ multilinear ist.) Die Polynome $p_{t_0} = 0$ und $p_{t_1} = 1$ enthalten überhaupt keine Variable. Nun sei v ein innerer Knoten mit 0-Nachfolger v_0 und 1-Nachfolger v_1, der mit x_i beschriftet ist. Aus der Read-Once-Eigenschaft folgt, dass in dem Bereich des Graphen G, der von v_0 und v_1 aus erreichbar ist, die Boolesche Variable x_i nirgends als Beschriftung vorkommen kann. Daher enthalten p_{v_0} und p_{v_1} die Variable X_i nicht. Nach Induktionsvoraussetzung sind p_{v_0} und p_{v_1}

multilinear. Daraus folgt durch Ausmultiplizieren, dass auch $p_v = (1 - X_i)p_{v_0} + X_i p_{v_1}$ keine Variable mit einem Exponenten größer als 1 enthält, also multilinear ist. □

Die nächste Feststellung ist dann zentral: Für *multilineare* Polynome p und p' über einem beliebigen Körper K folgt aus gleichem Werteverlauf auf Inputs in $\{0,1\}^n \subseteq K$, dass sie identisch sind. Dies ergibt sich sofort aus dem folgenden Lemma (anzuwenden auf das Polynom $p - p'$).

Lemma 6.13
Es sei K ein Körper und p ein **multilineares** Polynom mit Variablen $X_1, \ldots, X_n$ und Koeffizienten aus K. Dann gilt: Wenn $p(a_1, \ldots, a_n) = 0$ für alle $a_1, \ldots, a_n \in \{0,1\}$, dann ist p das Nullpolynom.

Beweis: Wir zeigen durch Induktion über die Anzahl m der Variablen aus $X_1, \ldots, X_n$, die in p tatsächlich vorkommen: Wenn p nicht das Nullpolynom ist, dann gibt es ein $a \in \{0,1\}^n$ mit $p(a) \neq 0$.

I.A.: Sei $m = 0$. Dann ist p eine Konstante $c \in K$. Weil p nicht das Nullpolynom ist, gilt $c \neq 0$. Daraus folgt $p(a) = c \neq 0$ für alle $a \in \{0,1\}^n$.

I.V.: $m \geq 1$ und die Behauptung stimmt für alle $m' < m$.

I.Schritt: Angenommen, in p kommen $m \geq 1$ viele Variablen vor. Wir wählen eine davon, etwa X_1. Wir klammern X_1 aus allen Termen aus, in denen es vorkommt, und erhalten die Darstellung

$$p(X_1, \ldots, X_n) = X_1 \cdot q(X_2, \ldots, X_n) + r(X_2, \ldots, X_n).$$

(Höhere Potenzen von X_1 gibt es nicht, weil p multilinear ist.) Dabei ist q ein multilineares Polynom mit Variablen $X_2, \ldots, X_n$, das nicht das Nullpolynom ist, und in dem strikt weniger Variablen tatsächlich vorkommen als in p. Nach I.V. gibt es $(a_2, \ldots, a_n) \in \{0,1\}^{n-1}$ mit $q(a_2, \ldots, a_n) \neq 0$. Die Werte $p(0, a_2, \ldots, a_n) = r(a_2, \ldots, a_n)$ und $p(1, a_2, \ldots, a_n) = q(a_2, \ldots, a_n) + r(a_2, \ldots, a_n)$ sind also verschieden, und wir können $a_1 \in \{0,1\}$ mit $p(a_1, a_2, \ldots, a_n) \neq 0$ wählen. □

Der randomisierte Algorithmus zum Vergleich von f_G und $f_{G'}$, mit Variablen $x_1, \ldots, x_n$, verläuft nun im Prinzip wie folgt:

1. Wähle Primzahl $p > 2n$.
2. Wähle zufällig $a_1, \ldots, a_n$ aus $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, setze $a = (a_1, \ldots, a_n)$.
3. Werte über $\mathbb{Z}_p$ aus: $y \leftarrow p_G(a)$ und $z \leftarrow p_{G'}(a)$.
4. Die Ausgabe ist $[y \neq z]$.

Das Verhalten dieses Algorithmus ist wie folgt:

1. Fall: $f_G \neq f_{G'}$. – Nach Lemma 6.10 gilt dann $p_G \neq p_{G'}$. Nach dem Satz von Schwartz-Zippel hat $h = p_G - p_{G'}$ in $\mathbb{Z}_p$ maximal $\deg(p_G - p_{G'}) \cdot p^{n-1}$ Nullstellen. Da der Grad von p_G und $p_{G'}$ höchstens n ist, sind dies höchstens $n \cdot p^{n-1}$ viele. Also ist die Wahrscheinlichkeit, dass in Zeile 2 zufällig ein Element a mit $p_G(a) = p_{G'}(a)$, also eine Nullstelle von h, gewählt wird, höchstens $n/p < \frac{1}{2}$. Damit: $\mathbf{Pr}(\text{Ausgabe ist } 0) < \frac{1}{2}$.

2. Fall: $f_G = f_{G'}$. Nach Lemma 6.12 sind p_G und $p_{G'}$ multilineare Polynome. Mit Lemma 6.13 folgt $p_G = p_{G'}$. Daher liefert der Algorithmus immer den Wert 0 zurück.

Wir haben es also mit einem Monte-Carlo-Algorithmus mit einseitigem Fehler zu tun, mit Fehlerschranke $\frac{1}{2}$.

Die einzige verbleibende Frage ist, wie man zu einem gegebenen Branchingprogramm G und einer Eingabe $a \in \mathbb{Z}_p^n$ den Wert $p_G(a)$ effizient berechnet. Es ist nicht möglich, das Polynom explizit aufzuschreiben, da es im Normalfall viel zu viele Terme haben wird. (Zum Beispiel hat die Funktion $x_1 \oplus \cdots \oplus x_n$ ein sehr kleines Branchingprogramm, aber sein Polynom hat $2^n - 1$ Terme.) Die Idee ist, die Werte $r_v = p_v(a)$ durch Induktion über die Knotenreihenfolge in G auszurechnen. Es sei $a = (a_1, \ldots, a_n) \in \mathbb{Z}_p^n$ der Input. Das Branchingprogramm G sei gegeben. Wir rechnen:

(i) $r_{t_0} \leftarrow 0$ und $r_{t_1} \leftarrow 1$ (in $\mathbb{Z}_p$).

(ii) Wenn v ein innerer Knoten ist, mit Beschriftung x_i, mit 0-Nachfolger v_0 und 1-Nachfolger v_1, dann setze

$$r_v \leftarrow (a_i \cdot r_{v_1} + (1 - a_i) \cdot r_{v_0}) \bmod p.$$

(iii) Die Ausgabe ist r_s.

Man zeigt leicht durch Induktion über die Knotenreihenfolge, dass $r_v = p_v(a)$ gilt, und dass daher die Ausgabe r_s tatsächlich gleich $p_s(a) = p_G(a)$ ist. Der Aufwand ist $O(|V|)$ Operationen in $\mathbb{Z}_p$, für die Knotenmenge V von G.

Bemerkung: Man nimmt mit Erstaunen zur Kenntnis, dass der Äquivalenztest anhand von G eine Berechnung in $\mathbb{Z}_p$ durchführt, obwohl die Problemstellung keine Zahlen erwähnt und G überhaupt nicht für die Benutzung mit Zahlen „gedacht“ ist, sondern als Darstellung einer Booleschen Funktion.

6.7 $*$ Perfekte Matchings in beliebigen ungerichteten Graphen

Bevor man diesen Abschnitt liest, sollte man Abschnitt 6.4 zu perfekten Matchings in bipartiten Graphen verstanden haben. Hier betrachten wir beliebige ungerichtete Graphen $G = (V, E)$ mit $V = \{1, \ldots, n\}$. Die Adjazenzmatrix A_G von G ist symmetrisch und hat Nullen auf der Hauptdiagonale. Es geht zunächst um den Test, ob G ein perfektes Matching $M \subseteq E$ besitzt, d. h. eine Kantenmenge M, in der jeder Knoten genau einmal vorkommt. Offensichtlich ist es hierfür notwendig, dass $n = |V|$ eine gerade Zahl ist. Die Größe $|M|$ eines perfekten Matchings muss $n/2$ sein.[8]

Definition 6.14
Die **Tutte-Matrix T_G** zu einem Graphen G mit Knoten $1, \ldots, n$ ist eine $n \times n$-Matrix mit Einträgen

$$f_{ij} = \left\{ \begin{array}{ll} X_{ij}, & \textit{falls } i < j \textit{ und } (i,j) \in E, \\ -X_{ji}, & \textit{falls } j < i \textit{ und } (i,j) \in E, \\ 0, & \textit{sonst.} \end{array} \right\}, \quad \textit{für } 1 \leq i, j \leq n.$$

Beispiel: Zum Graphen $G = (\{1,2,3,4,5,6\}, E)$ mit $E = \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4),(4,5),(4,6),(5,6)\}$ gehört die Tutte-Matrix

$$T_G = \begin{pmatrix} f_{11} & \cdots & f_{16} \\ \vdots & \ddots & \vdots \\ f_{61} & \cdots & f_{66} \end{pmatrix} = \begin{pmatrix} 0 & X_{12} & X_{13} & X_{14} & 0 & 0 \\ -X_{12} & 0 & X_{23} & X_{24} & 0 & 0 \\ -X_{13} & -X_{23} & 0 & X_{34} & 0 & 0 \\ -X_{14} & -X_{24} & -X_{34} & 0 & X_{45} & X_{46} \\ 0 & 0 & 0 & -X_{45} & 0 & X_{56} \\ 0 & 0 & 0 & -X_{46} & -X_{56} & 0 \end{pmatrix}.$$

Offensichtlich gilt ${T_G}^{\mathsf{T}} = -T_G$, die Tutte-Matrix ist also *schiefsymmetrisch*. Die Einträge f_{ii} auf der Diagonalen sind 0. Auch für ungerichtete Graphen gibt es einen Zusammenhang zwischen Determinante der Tutte-Matrix und der Existenz von perfekten Matchings in G.

Satz 6.15 ***(Tutte)***
Sei G ein beliebiger ungerichteter Graph. Dann gilt:
G hat ein perfektes Matching $\Leftrightarrow$ $\det(T_G) \neq 0$.

[8] William T. „Bill“ Tutte (1917–2002) war ein bedeutender britisch-kanadischer Mathematiker. Im 2. Weltkrieg arbeitete er als Kryptanalytiker in Bletchley Park, UK. Nach dem Krieg wandte er sich der Kombinatorik und Graphentheorie zu und erzielte dort bahnbrechende Ergebnisse. Er lehrte von 1962 bis 1985 an der University of Waterloo in Kanada.

Um ein Gefühl dafür zu bekommen, was sich hier abspielt, betrachten wir das Beispiel. Die Kantenmenge $M = \{(1,3),(2,4),(5,6)\}$ ist ein perfektes Matching, und in T_G kommt der Term für die Permutation $\sigma_1 = \left(\begin{smallmatrix}1&2&3&4&5&6\\3&4&1&2&6&5\end{smallmatrix}\right)$ vor, die 1 mit 3, 2 mit 4 und 5 mit 6 vertauscht:

$$t_{\sigma_1} = (-1)X_{13}X_{24}(-X_{13})(-X_{24})X_{56}(-X_{56}) = X_{13}^2X_{24}^2X_{56}^2.$$

(Die Permutation σ_1 ist Produkt von drei Vertauschungen, also hat sie Signum -1.)

Leider ist damit die Sache noch nicht erledigt. Es gibt nämlich auch die Permutation $\sigma_2 = \left(\begin{smallmatrix}1&2&3&4&5&6\\2&3&1&5&6&4\end{smallmatrix}\right)$ mit Signum 1, die scheinbar ebenfalls einen Term in T_G erzeugt:

$$t_{\sigma_2} = (+1)X_{12}X_{23}(-X_{13})X_{45}X_{56}(-X_{46}) = X_{12}X_{23}X_{13}X_{45}X_{56}X_{46}.$$

Wir werden im Beweis sehen, dass alle solchen Terme, die einen Zyklus ungerader Länge (wie hier $1 \to 2 \to 3 \to 1$ oder kurz $(1,2,3)$ enthalten, sich aufgrund von Vorzeicheneffekten gegenseitig auslöschen. Konkret gibt es auch die Permutation $\sigma_3 = \left(\begin{smallmatrix}1&2&3&4&5&6\\3&1&2&5&6&4\end{smallmatrix}\right)$, ebenfalls mit Signum 1, die den Term

$$t_{\sigma_3} = (+1)X_{13}(-X_{12})(-X_{23})X_{45}X_{56}(-X_{46}) = -X_{12}X_{23}X_{13}X_{45}X_{56}X_{46} = -t_{\sigma_2}$$

liefert.

Bleiben noch Permutationen, die nicht direkt von einem Matching kommen, aber keinen ungeraden Kreis enthalten, wie zum Beispiel $\sigma_4 = \left(\begin{smallmatrix}1&2&3&4&5&6\\2&3&4&1&6&5\end{smallmatrix}\right)$. Diese hat Signum $+1$, sie enthält nur Kreise gerader Länge, nämlich $(1,2,3,4)$ und $(5,6)$, und sie führt zu einem Term

$$t_{\sigma_4} = (+1)X_{12}X_{23}X_{34}(-X_{14})X_{56}(-X_{56}) = X_{12}X_{23}X_{34}X_{14}X_{56}^2.$$

Bei diesen Termen findet keine Auslöschung mit anderen Termen statt. Hier ist der Trick der, dass Kreise gerader Länge benutzt werden können, um ein Matching zu erhalten. (Man nimmt aus jedem Kreis jede zweite Kante, also hier $(1,2),(3,4)$ und $(5,6)$.)

Wir führen diese am Beispiel entwickelten Ideen jetzt systematisch durch.

Beweis von Satz 6.15: Man schreibt

$$\det(T_G) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)}. \tag{5}$$

„⇒“: Sei M ein perfektes Matching in G. Dann betrachtet man die Permutation σ_M, die i auf j und j auf i abbildet genau dann wenn $(i,j) \in M$ ist. In der Summe für die Determinante kommt der Term

$$t_{\sigma_M} = \operatorname{sign}(\sigma_M) \cdot f_{1,\sigma_M(1)} \cdots f_{n,\sigma_M(n)} = \operatorname{sign}(\sigma_M) \cdot (-1)^{n/2} \cdot \prod_{\substack{(i,j)\in M\\ i<j}} X_{ij}^2$$

vor. Weil die Permutation σ aus den Faktoren des Terms t_{σ_M} abgelesen werden kann, kann kein Term $t_{\sigma_{M'}}$ für ein perfektes Matching $M' \neq M$ den Term t_{σ_M} auslöschen. Für einen Term t_σ, bei dem σ nicht von einem Matching herrührt, gilt dies erst recht, weil dann t_σ nicht Produkt von Quadraten von Variablen ist. Also ist $\det(T_G)$ nicht das Nullpolynom.

„$\Leftarrow$“: Nun nehmen wir an, dass $\det(T_G)$ nicht das Nullpolynom ist. Wir betrachten die Summe in (5).

Jeder Term $t_\sigma = \operatorname{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)}$, der nicht 0 ist, definiert eine „*Spur*“

$$\operatorname{tr}_\sigma = \{(i, \sigma(i)) \mid 1 \leq i \leq n\},$$

eine Menge von geordneten Paaren, die wir als *gerichtete* Kanten auffassen. Jede dieser Kanten läuft entlang einer (ungerichteten) Kante in G. Weil σ Permutation ist, hat jeder Knoten i Eingangsgrad und Ausgangsgrad 1, also bildet tr_σ eine Menge von Kreisen, die Zyklen von σ entsprechen. Weil $f_{ii} = 0$ ist, haben diese Kreise eine Mindestlänge von 2.

Wir betrachten zunächst Permutationen σ, in deren Spur ein Kreis ungerader Länge vorkommt. Wenn ein solches σ vorliegt, wählen wir den Kreis K_σ ungerader Länge in tr_σ, der den Knoten mit kleinster Nummer enthält. Wir drehen in tr_σ die Umlaufrichtung des Kreises K_σ herum. Dadurch entsteht die Spur $\operatorname{tr}_{\sigma'}$ einer eindeutig bestimmten Permutation σ'. Auf diese Weise entsteht eine eineindeutige Zuordnung $\sigma \mapsto \sigma'$ unter den Permutationen mit Spuren, die einen ungeraden Kreis haben. Diese Zuordnung ist *involutorisch*, d. h., sie ist gleich ihrer eigenen Umkehrung. Auf diese Weise werden die Permutationen σ mit einem ungeraden Kreis in tr_σ einander paarweise zugeordnet. – Es gilt dabei $\operatorname{sign}(\sigma) = \operatorname{sign}(\sigma')$, da das Vorzeichen einer Permutation nur von der Anzahl der geraden Kreise abhängt[9] und diese bei σ und σ' gleich ist.

[9] Genauer gesagt ist $\operatorname{sign}(\sigma)$ gleich 1 bzw. -1 je nachdem, ob tr_σ eine gerade oder ungerade Anzahl gerader Kreise hat.

Beispiel 6.16
Betrachte die folgende Permutation σ:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$\sigma(i)$	16	18	17	2	7	11	1	13	12	8	3	9	10	6	4	5	14	15

Diese Permutation hat die folgenden Kreise:

$k_1 = (9, 12), k_2 = (2, 18, 15, 4), k_3 = (7, 1, 16, 5), k_4 = (8, 13, 10), k_5 = (6, 11, 3, 17, 14).$

Kreise k_1, k_2, k_3 haben gerade Länge, Kreise k_4 und k_5 haben ungerade Länge. Damit ist $\text{sign}(\sigma) = -1$. Unter den beiden Kreisen ungerader Länge ist $K_\sigma = k_5$ derjenige, der den Knoten mit der kleinsten Nummer enthält, nämlich 3. (Knoten 1 und 2 liegen auf geraden Kreisen.) Damit erhalten wir σ' durch Umdrehen von $K_\sigma = k_5$ zu $K_{\sigma'} = k_5' = (14, 17, 3, 11, 6)$. Die Tabelle von σ' (Änderungen gegenüber σ in Fettdruck):

i	1	2	**3**	4	5	**6**	7	8	9	10	**11**	12	13	**14**	15	16	**17**	18
$\sigma'(i)$	16	18	**11**	2	7	**14**	1	13	12	8	**6**	9	10	**17**	4	5	**3**	15

Dann gilt:

$$\begin{aligned}
t_\sigma + t_{\sigma'} &= \text{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)} + \text{sign}(\sigma') \cdot f_{1,\sigma'(1)} \cdots f_{n,\sigma'(n)} \\
&= \text{sign}(\sigma) \cdot \prod_{(i,\sigma(i)) \notin K_\sigma} f_{i,\sigma(i)} \cdot \Big(\prod_{(i,\sigma(i)) \in K_\sigma} f_{i,\sigma(i)} + \prod_{(i,\sigma(i)) \in K_\sigma} f_{\sigma(i),i} \Big) \\
&\overset{(*)}{=} \text{sign}(\sigma) \cdot \prod_{(i,\sigma(i)) \notin K_\sigma} f_{i,\sigma(i)} \cdot \Big(\prod_{(i,\sigma(i)) \in K_\sigma} f_{i,\sigma(i)} + \prod_{(i,\sigma(i)) \in K_\sigma} (-f_{i,\sigma(i)}) \Big) \\
&= \text{sign}(\sigma) \cdot f_{1,\sigma(1)} \cdots f_{n,\sigma(n)} \cdot \Big(1 + (-1)^{\#(\text{Kanten in } K_\sigma)} \Big) = 0.
\end{aligned}$$

Die Gleichheit $(*)$ folgt dabei aus der Definition der Tutte-Matrix, die letzte Gleichheit aus der Tatsache, dass K_σ ungerade Länge hat. Das heißt, dass die Permutationen σ, deren Spuren einen ungeraden Kreis enthalten, insgesamt nichts zum Polynom $\det(T_G)$ beitragen.

Wir hatten angenommen, dass $\det(T_G)$ nicht das Nullpolynom ist. Nach dem eben erreichten Zwischenergebnis muss es dann eine Permutation σ mit $t_\sigma \neq 0$ geben, deren Spur tr_σ nur aus Kreisen gerader Länge besteht. Aus einer solchen Spur lässt sich aber leicht ein perfektes Matching M_σ konstruieren: Man nimmt aus jedem Kreis gerader Länge jede zweite Kante und ignoriert dann die Richtung. □

Der Algorithmus für den Test, ob $G = (V, E)$ ein perfektes Matching enthält, ist nun (bis auf die Sache mit den Vorzeichen) derselbe wie bei den bipartiten Graphen. In der

folgenden Formulierung lassen wir den Umweg über die Tutte-Matrix von vornherein weg.

Algorithmus 6.5 ***Test auf perfektes Matching, ungerichteter Graph***
Input: (Ungerichteter) Graph $G = (V, E)$ mit $V = \{1, \ldots, n\}$.
Problem: Teste, ob G ein perfektes Matching hat.
Methode:
0 Wenn n ungerade ist: **return** 0.
1 Wähle Primzahl $p > 2n$; // Körper $\mathbb{Z}_p$, $S = \mathbb{Z}_p$
2 Für $(i, j) \in E$ mit $i < j$ wähle a_{ij} aus $\mathbb{Z}_p$ zufällig;
3 Für $(i, j) \in E$ mit $i > j$ setze $a_{ij} \leftarrow -a_{ji}$;
4 Für $(i, j) \notin E$ setze $a_{ij} \leftarrow 0$;
5 Bilde Matrix $A = (a_{ij})_{1 \leq i,j, \leq n}$;
6 **return** $[\det(A) \neq 0]$. // z. B. Gauss-Elimination über $\mathbb{Z}_p$

Zeitbedarf und Wahrscheinlichkeitsanalyse sind praktisch identisch zur Analyse bei Algorithmus 6.2. Ebenso gelten hier auch die Bemerkungen zur Parallelisierung.

Es sei bemerkt, dass *deterministische* Algorithmen zum Test auf die Existenz von perfekten Matchings in allgemeinen ungerichteten Graphen zwar existieren, aber ungleich komplizierter sind als die für bipartite Graphen.

Nehmen wir nun an, wir haben mit Algorithmus 6.5, eventuell nach mehreren Wiederholungen, festgestellt, dass Graph G ein perfektes Matching besitzt. Nun wollen wir ein solches perfektes Matching *berechnen*. Man könnte dazu vorgehen wie im Fall von bipartiten Graphen (s. Ende von Abschnitt 6.4), d. h. die Kanten (i, j) nacheinander darauf testen, ob der Graph ohne (i, j) immer noch ein perfektes Matching besitzt und (i, j) zu streichen, falls dies so ist.

Wir betrachten einen alternativen Algorithmus, der auch gut parallelisierbar ist. (Literatur für das Folgende: K. Mulmuley, U. V. Vazirani, V. V. Vazirani, Matching is as easy as matrix inversion, *Combinatorica* 7 (1): 105113. 1987.) Wir benötigen eine Tatsache aus der Algorithmik für lineare Algebra.

Fakt 6.17 ***(Samuelson 1942, Berkowitz 1985)***
Sei K ein Körper. Dann gilt: $O(n^3)$ Prozessoren können die Determinante einer Matrix $A \in K^{n \times n}$ in $O((\log n)^2)$ Zeit (d. h. Additionen und Multiplikationen in K) berechnen.

Die Idee für den Matchingalgorithmus ist dann im Prinzip einfach: Für jede Kante $(i, j) \in E$ testet eine Gruppe von $O(n^3)$ Prozessoren (die jeweils eine Addition oder Multiplikation in K in einem Schritt ausführen können), z. B. durch Berechnen einer

passenden Determinante, ob $(i,j) \in E$ Element eines „gesuchten“ perfekten Matchings ist.

Dabei stellt sich aber folgendes Problem: Ein Graph G kann viele verschiedene perfekte Matchings haben, und man müsste die Prozessoren dazu bringen, dass alle ihren Test bezüglich eines festen perfekten Matchings durchführen. Um eine solche Auswahl zu treffen, verwendet man wieder Randomisierung. Man versieht die Kanten in E mit zufällig gewählten Gewichten, und zwar so, dass es mit einer gewissen Wahrscheinlichkeit nur *ein* perfektes Matching M_0 mit minimalem Gesamtgewicht gibt. Dann zeigt man, dass eine Determinantenberechnung pro Kante (i,j) genügt, um zu testen, ob $(i,j) \in M_0$ gilt.

Definition 6.18
Ein (endliches) Mengensystem $(X, \mathcal{F})$ besteht aus einer Menge X mit $|X| = m \geq 1$ und einer Menge $\mathcal{F}$ von Teilmengen von X. Wenn eine Gewichtsfunktion

$$w\colon X \ni x \mapsto w(x) \in \mathbb{N}$$

gegeben ist, definieren wir $w(S)$ als $\sum_{x \in S} w(x)$, für $S \in \mathcal{F}$.

In unserer Anwendung wird $X = E$ sein und $\mathcal{F}$ die Menge aller perfekten Matchings in G. Gewichte für die Kanten werden zufällig gewählt. Dann sind die Werte $w(S)$ für Mengen $S \in \mathcal{F}$ Zufallsvariable.

Lemma 6.19 ***Isolationslemma***
Wenn man $w(x)$, für $x \in X$, aus $\{1, \ldots, 2m\}$ zufällig wählt, dann gilt:

$$\mathbf{Pr}(\text{in } \mathcal{F} \text{ gibt es eine eindeutig bestimmte Menge mit minimalem Gewicht}) \geq \frac{1}{2}.$$

(Anstelle der Mengengröße $2m$ kann man auch Größe rm für einen anderen Faktor $r \geq 2$ wählen und erhält Wahrscheinlichkeit $\geq 1 - 1/r$.)

Beweis: Wir können o. B. d. A. annehmen, dass jedes $x \in X$ zu mindestens einem $S \in \mathcal{F}$ gehört, aber nicht zu allen. (Wenn ein x zu keinem S gehört oder zu allen, dann hat $w(x)$ auf die Anordnung der Gewichte $w(S)$ für $S \in \mathcal{F}$ keinen Einfluss, und wir können dieses x für die Zwecke des Beweises ignorieren.)

Für $x \in X$ definieren wir die folgenden Zufallsvariablen:

$$\begin{aligned} W_x &:= \min\{w(S - \{x\}) \mid S \in \mathcal{F}, x \in S\}; \\ \bar{W}_x &:= \min\{w(S) \mid S \in \mathcal{F}, x \notin S\}. \\ \alpha_x &:= \bar{W}_x - W_x. \end{aligned}$$

Da W_x und $\bar{W}_x$ und damit auch α_x nur von $w(x')$ mit $x' \neq x$ abhängen, sind α_x und $w(x)$ unabhängig.

Nun seien alle Werte $w(x')$ für $x' \neq x$ und auch $w(x)$ gewählt. Wir wählen $S_0 \in \mathcal{F}$ mit $x \in S_0$, so dass $w(S_0 - \{x\}) = W_x$ gilt. Dann ist

$$w(S_0) = w(S_0 - \{x\}) + w(x) = W_x + w(x) \tag{6}$$

das kleinste Gewicht einer Menge, die x enthält. Wir betrachten zwei Fälle:

1. Fall: $w(x) < \alpha_x$. – Dann gilt für jede Menge $S' \in \mathcal{F}$, die x nicht enthält:

$$w(S') \geq \bar{W}_x \overset{(6)}{=} \bar{W}_x - W_x - w(x) + w(S_0) = \alpha_x - w(x) + w(S_0) > w(S_0).$$

Daher muss *jede* Menge S mit minimalem Gewicht das Element x enthalten.

2. Fall: $w(x) > \alpha_x$. – Dann ist $0 > \alpha_x - w(x) = \bar{W}_x - W_x - w(x) \overset{(6)}{=} \bar{W}_x - w(S_0)$. Also gibt es eine Menge S' ohne x, die $w(S') < w(S_0)$ erfüllt. Daher kann *keine* Menge S mit minimalem Gewicht das Element x enthalten.

Wir nennen $x \in X$ *unentschieden*, wenn $w(x) = \alpha_x$.

Wir haben $\mathbf{Pr}(x \text{ ist unentschieden}) \leq \frac{1}{2m}$, weil α_x und $w(x)$ unabhängig sind und $w(x)$ zufällig in $\{1, \ldots, 2m\}$ ist.

Mit der Vereinigungsschranke folgt: $\mathbf{Pr}(\exists x \in X : x \text{ ist unentschieden}) \leq \frac{|X|}{2m} \leq \frac{1}{2}$.

Nehmen wir nun an, kein x ist *nicht* unentschieden, d. h. alle x erfüllen $w(x) \neq \alpha_x$. Wir haben Folgendes gesehen:

(i) Wenn $w(x) < \alpha_x$, dann ist $x \in S$ für *jede* Menge S mit minimalem Gewicht.

(ii) $w(x) > \alpha_x$, dann ist $x \notin S$ für *jede* Menge S mit minimalem Gewicht.

Daraus ergibt sich: Es gibt genau eine Menge mit minimalem Gewicht, nämlich $S_0 = \{x \mid w(x) < \alpha_x\}$. □

Wir wenden das Isolationslemma auf die Menge $X = E$ und $\mathcal{F} = \{M \subseteq E \mid M \text{ ist perfektes Matching}\}$ an. Wir wählen also für jede Kante (i, j) in M ein zufälliges Gewicht w_{ij} aus $\{1, \ldots, 2m\}$, mit $m = |E|$. Dadurch bekommt jedes Matching M ein Gewicht $w(M) = \sum_{(i,j) \in M} w_{ij}$.

Nach dem Isolationslemma gibt es mit Wahrscheinlichkeit mindestens $\frac{1}{2}$ nur ein einziges Matching M_0 mit minimalem Gewicht.

Nun wollen wir für jede Kante $(i, j) \in E$ eine Prozessorengruppe (unabhängig von den anderen) testen lassen, ob $(i, j) \in E$ gilt. Hierzu bilden wir zunächst aus T_G eine

neue Matrix $B = (B_{ij})_{1 \le i,j \le n}$, indem wir X_{ij} in T_G durch $2^{w_{ij}}$ ersetzen. Das heißt:

$$B_{ij} = \left\{ \begin{array}{ll} 2^{w_{ij}}, & \text{falls } i < j \text{ und } (i,j) \in E, \\ -2^{w_{ji}}, & \text{falls } j < i \text{ und } (i,j) \in E, \\ 0, & \text{sonst.} \end{array} \right\}, \quad \text{für } 1 \le i,j \le n.$$

Wir betrachten $\det(B)$.

Lemma 6.20
Wenn der Gewichtssatz $(w_{ij})_{1 \le i,j \le n}$ ein eindeutig bestimmtes minimales Matching M_0 bestimmt, dann ist $\det(B) \neq 0$. Genauer: Für $w_0 = w(M_0)$ gilt: 2^{2w_0} ist Teiler von $\det(B)$, aber 2^{2w_0+1} ist kein Teiler von $\det(B)$.

(Aus dem Lemma ergibt sich, dass man aus $\det(B)$ den Wert w_0 ablesen kann.)

Beweis: Wir argumentieren mit denselben Strukturen wie im Beweis des Satzes von Tutte. Es sei $B = (B_{ij})_{1 \le i,j \le n}$. Dann gilt

$$\det(B) = \sum_{\sigma \in S_n} \operatorname{sign}(\sigma) \cdot B_{1,\sigma(1)} \cdots B_{n,\sigma(n)}. \tag{7}$$

Im Polynom $\det(T_G)$ heben sich die Terme t_σ, deren Spuren tr_σ mindestens einen ungeraden Kreis haben, gegenseitig weg. Das gilt dann auch für die Terme in (7), deren Permutationen σ Zyklen ungerader Länge enthalten.

Wir betrachten jetzt einen beliebigen Summanden $\operatorname{sign}(\sigma) \cdot B_{1,\sigma(1)} \cdots B_{n,\sigma(n)}$, für dessen Permutation σ die Spur tr_σ nur Kreise gerader Länge besitzt. Diese Kreise kann man stets in zwei (nicht notwendig disjunkte) Matchings M_1 und M_2 zerlegen (jede zweite Kante eines Kreises kommt in M_1, die anderen in M_2). Aus der Wahl der Faktoren B_{ij} folgt

$$\begin{aligned} &|B_{1,\sigma(1)} \cdots B_{n,\sigma(n)}| \\ &\quad = \prod_{(i,j) \in M_1} 2^{w_{ij}} \cdot \prod_{(i,j) \in M_2} 2^{w_{ij}} = 2^{\sum_{(i,j) \in M_1} w_{ij} + \sum_{(i,j) \in M_2} w_{ij}} = 2^{w(M_1)+w(M_2)}. \end{aligned}$$

Wenn $M_1 \neq M_0$ oder $M_2 \neq M_0$, dann ist $w(M_1) + w(M_2) > 2w_0$, also ist der Term zu σ durch 2^{2w_0+1} teilbar.

Es gibt in (7) nur einen Summanden mit geraden Kreisen, bei dem die Zerlegung der Kreise in zwei Matchings zu zwei Kopien von M_0 führt, und zwar ist das der Term zu der schon erwähnten Permutation σ_0 mit $\sigma_0(i) = j$ genau dann wenn $(i,j) \in M_0$ gilt. Der Term zu σ_0 hat Betrag 2^{2w_0}. Daher ist $\det(B)$ Summe von $\pm 2^{2w_0}$ und anderer Terme, die durch 2^{2w_0+1} teilbar sind. Die Behauptung des Lemmas folgt. □

Um zu entscheiden, ob eine Kante zu M_0 gehört oder nicht, muss man noch einen weiteren Trick anwenden. Zu Indexpaar (i,j), $i \neq j$, betrachten wir die „doppelte Streichungsmatrix“ $B^{(i,j)}$, die aus B entsteht, indem man Zeilen i und j und Spalten i und j durch Nullen ersetzt; nur an der Kreuzung von Zeile i mit Spalte j bleibt B_{ij} stehen, an der Kreuzung von Zeile j mit Spalte i bleibt B_{ji} stehen.

Bei der Bildung der Determinante $\det(B^{(i,j)})$ bleiben dann nur diejenigen Terme t_σ stehen, für die $\sigma(i) = j$ und $\sigma(j) = i$ gilt. Das heißt:

$$\det(B^{(i,j)}) = \sum_{\substack{\sigma \in S_n \\ \sigma(i)=j \wedge \sigma(j)=i}} \operatorname{sign}(\sigma) \cdot B_{1,\sigma(1)} \cdots B_{n,\sigma(n)}.$$

Wie vorher argumentiert man, dass Beiträge von Permutationen mit Spuren, die ungerade Kreise enthalten, sich gegenseitig aufheben. Beiträge zu $\det(B^{(i,j)})$ kommen also nur von Summanden, die $\sigma(i) = j$ und $\sigma(j) = i$ erfüllen und daneben nur gerade Kreise enthalten. Wieder kann man jeden dieser Terme als $2^{w(M_1)+w(M_2)}$ schreiben, wobei sowohl M_1 als auch M_2 die Kante (i,j) enthalten. Nun gibt es zwei Fälle.

1. Fall: $(i,j) \notin M_0$. – Dann ist $\det(B^{(i,j)})$ Summe von Termen $2^{w(M_1)+w(M_2)}$ mit $M_1 \neq M_0$ oder $M_2 \neq M_0$, so dass $w(M_1) + w(M_2) > 2w_0$ gilt. Also ist $\det(B^{(i,j)})$ durch 2^{2w_0+1} teilbar.

2. Fall: $(i,j) \in M_0$. Dann ist $\det(B^{(i,j)})$ Summe von $\pm 2^{2w_0}$ und anderen Termen, die alle durch 2^{2w_0+1} teilbar sind. Also ist $\det(B^{(i,j)})$ nicht durch 2^{2w_0+1} teilbar.

Wir haben gezeigt:

Lemma 6.21
Sei M_0 eindeutig bestimmtes Matching mit minimalem Gewicht w_0. Dann gilt:

$$(i,j) \in M_0 \qquad \Leftrightarrow \qquad \frac{\det(B^{(i,j)})}{2^{2w_0}} \text{ ist ungerade.}$$

Es ergibt sich der folgende Algorithmus für die Ermittlung eines perfekten Matchings.

Algorithmus 6.6 ***Finde perfektes Matching in ungerichtetem Graphen***
Input: Graph $G = (V, E)$ mit n Knoten und m Kanten; G hat perfektes Matching
Aufgabe: Finde ein perfektes Matching in G.
Methode:

1 Für Kante (i, j), $i < j$, wähle zufällige Zahl w_{ij} aus $\{1, \ldots, 2m\}$; $B_{ij} := 2^{w_{ij}}$;
2 Für Kante (i, j), $i > j$, setze $B_{ij} := -B_{ji}$;
3 Für $(i, j) \notin E$ setze $B_{ij} := 0$;
4 Bilde Matrix $B = (B_{ij})_{1 \leq i,j, \leq n}$;
5 $b := \det(B)$;
6 ermittle maximales k so dass 2^k Teiler von b ist;
7 **if** k ungerade **then return** „Fehlschlag“;
9 für jede Kante $(i, j) \in E$ tue folgendes: // parallele Ausführung möglich
10 $B^{(i,j)} :=$ doppelte Streichungsmatrix von B (wie beschrieben);
11 $q_{ij} := \det(B^{(i,j)})/2^k$;
12 $M := \{(i, j) \mid q_{ij}$ ungerade$\}$;
13 **if** $|M| = n/2$ und jedes $i \in V$ kommt in M vor
14 **then return** M **else return** „Fehlschlag“

Satz 6.22
Algorithmus 6.6 ist ein selbstverifizierender Algorithmus mit polynomieller Laufzeit, der zu gegebenem Graphen G mit perfektem Matching entweder ein perfektes Matching oder „Fehlschlag“ ausgibt. Die Wahrscheinlichkeit für „Fehlschlag“ ist durch $\frac{1}{2}$ beschränkt. Der Algorithmus ist parallelisierbar (polynomiell viele Prozessoren, $O((\log n)^3)$ Zeit).

Beweis: Zur Korrektheit: Der Test in Zeilen 13–14 führt dazu, dass auf keinen Fall eine Menge M ausgegeben wird, die kein perfektes Matching in G ist. In allen anderen Fällen wird „Fehlschlag“ ausgegeben. Es handelt sich also um einen selbstverifizierenden Algorithmus. Wir müssen die Wahrscheinlichkeit für die Ausgabe „Fehlschlag“ abschätzen. Nach dem Isolationslemma 6.19 erzeugt der Gewichtssatz w_{ij} mit Wahrscheinlichkeit höchstens $\frac{1}{2}$ mehrere perfekte Matchings in G mit minimalem Gewicht. Dies trägt also Wahrscheinlichkeit höchstens $\frac{1}{2}$ zur Fehlerwahrscheinlichkeit bei. (Zum Beispiel könnte k ungerade sein, oder Zeilen 9–12 berechnen eine Menge, die kein perfektes Matching ist.) Nun nehmen wir an, dass es nur ein perfektes Matching M_0 mit minimalem Gewicht gibt. Nach Lemma 6.20 ist dann k gerade und $w_0 = w(M_0) = k/2$. Nach Lemma 6.21 landet Kante (i, j) genau dann in der Menge M, wenn $(i, j) \in M_0$ ist. Es wird also ein perfektes Matching ausgegeben, nämlich M_0.

Zur Rechenzeit: Die Berechnung einer Determinante benötigt $O(n^3)$ Additionen und Multiplikationen in $\mathbb{Z}$. Es müssen $m + 1$ viele Determinanten berechnet werden.

Durch die Berechnungen können allerdings Zahlen bis zu einer Ziffernanzahl von $O(n^2 m \log n)$ entstehen, daher ist eine einzelne Multiplikation bis zu $O((n^2 m \log n)^2)$ teuer. Die gesamte Rechenzeit ist groß, aber immer noch polynomiell.

Zur parallelen Berechnung: Determinanten von $n \times n$-Matrizen können in paralleler „Zeit“ $O((\log n)^2)$ berechnet werden; dabei wird eine Addition oder Multiplikation in $\mathbb{Z}$ als Elementaroperation gezählt. Mit den schon erwähnten sehr langen Zahlen muss man auch für diese Elementaroperationen parallele Verfahren einsetzen. Mit $O(\ell^2)$ Prozessoren lassen sich zwei ℓ-ziffrige Zahlen in Zeit $O(\log \ell)$ addieren und multiplizieren. Damit kann man das gesamte Verfahren aus Algorithmus 6.6 parallel implementieren, mit polynomiell vielen Prozessoren und Rechenzeit $O((\log n)^3)$. □