Public-Key-Kryptosysteme und NP-Vollständigkeit

Studienarbeit von Daniel Raible
betreut von Professor Dr. Peter Hauck
Arbeitsbereich Diskrete Mathematik

29. Januar 2004

Ich versichere hiermit, dieses Dokument selbständig angefertigt und nicht mehr als die zitierten Quellen hinzugezogen zu haben.

Daniel Raible

Inhaltsverzeichnis

Einleitung

Ein Hauptanliegen der Kryptologie ist es Nachrichten so zu verschlüsseln, so dass es praktisch unmöglich ist den Originaltext ohne den entsprechenden Schlüssel zu erhalten. Ein in der Vergangenheit viel versprechender Ansatz hierfür war die Verwendung von NP-vollständigen Problemen. Die Lösung für das Problem sollte ebenfalls der Entschlüsselung der versandten Nachricht dienen. Da die Lösung für schwierige Instanzen von NP-vollständigen Problemen in der Praxis unmöglich ist, würde sich diese Eigenschaft ebenfalls auf die Entschlüsselung vererben. Ein erster (misslungener) Versuch in diese Richtung war das *Knapsack*-Kryptosystem.
Diese Studienarbeit befasst sich mit einem weiteren Kryptosystem dieser Art: "Polly Cracker". Der erste Teil wird wichtige mathematische Grundlagen vermitteln, die Funktionsweise von "Polly Cracker" erklären und exemplarisch dessen Verletzlichkeit demonstrieren.
Teil zwei behandelt das NP-vollständige Problem "Komplettierung lateinischer Quadrate". Zuerst wird dessen Verwandtschaft zu einem Triangulierungsproblem in einem Graphen aufgezeigt und dessen Codierung in SAT-Formeln erläutert. Dann wird darauf eingegangen wie dieses Problem in "Polly Cracker" eingebunden werden kann, so dass die gewünschten Eigenschaften vorhanden sind. Darauf folgend richten wir unser Augenmerk auf die Generierung von Probleminstanzen von "Komplettierung lateinischer Quadrate" und auf die Aufwandsabschätzung für die Lösung dieser. Hierzu entwickeln wir einen einfachen rekursiven Algorithmus zur Lösung. Die Aufwandsabschätzung wird dann empirisch mit Hilfe dieses Algorithmus und eines SAT-Beweisers betrieben.
Der Appendix skizziert die hierfür geschriebene Software bezüglich Aufbau, Anwendung und Ausgabe.

Kapitel 1

Das Kryptosystem "Polly Cracker"

1.1 Grundlagen

1.1.1 Algebraische Strukturen

Um die Funktionsweise von "Polly Cracker" verstehen zu können, müssen gewisse allgemeine algebraische Strukturen bekannt sein:

1.1.1.1 Gruppen

Definition. *Eine Gruppe ist ein Paar* $(G, \circ)$*, welches aus einer Menge* G *und einer Verknüpfung* $\circ$ *besteht:*

$$G \times G \to G, (x, y) \to x \circ y$$

so dass nachfolgende Bedingungen erfüllt sind:

(I) *Assoziativität:* $x \circ (y \circ z) = (x \circ y) \circ z \quad \forall x, y, z \in G$

(II) *Existenz eines Neutralelements: es existiert ein* $e \in G$*, so dass für alle* $x \in G$ *gilt* $x \circ e = e \circ x = x$

(III) *Existenz eines inversen Elements: für alle* $y \in G$ *gibt es ein* $x \in G$*, so dass gilt* $x \circ y = y \circ x = e$

Gilt für alle $x, y \in G$ *die Beziehung* $x \circ y = y \circ x$ *so wird die Gruppe kommutativ oder abelsch genannt.*

1.1.1.2 Ringe

Definition. *Ein Ring mit Einselement ist ein Tripel* $(R, +, \cdot)$ *bestehend aus einer Menge* R *und zwei Verknüpfungen:*

$$+ \ : \ R \times R \to R, \ (x, y) \to x + y$$

$$\cdot \ : \ R \times R \to R, \ (x, y) \to x \cdot y$$

so dass die folgenden Bedingungen erfüllt sein müssen:

(I) $(R, +)$ *ist eine abelsche Gruppe*

(II) $(R \setminus \{0\}, \cdot)$ *ist ein Monoid, d.h.* $\cdot$ *ist assoziativ und es existiert ein Neutralelement* $1 \in R \setminus \{0\}$ *mit* $1 \cdot x = x \cdot 1 = x \quad \forall x \in R \setminus \{0\}$

(III) Es gelten die Distributivgesetze:
Für alle $x, y, z \in R$ *gilt* $x \cdot (y+z) = x \cdot y + x \cdot z$ *und* $(x+y) \cdot z = x \cdot y + x \cdot z$

Gilt $x \cdot y = y \cdot x$ *so ist* R *kommutativ.*

1.1.1.3 Ideale

Definition. *Sei* R *ein kommutativer Ring. Eine Teilmenge* I *von* R *heisst ein Ideal von* R*, falls folgende Eigenschaften gelten:*

- $I \neq \emptyset$
- $a, b \in I \Rightarrow (a + b) \in I$ und $(a - b) \in I$
- $a \in I$ und $r \in R \Rightarrow a \cdot r \in I$

Definition. *Eine Menge* $E = \{e_1 e_2 ..... e_k\}$ *mit* $E \subset I$*,* I *ist Ideal, ist ein Erzeugendensystem, falls gilt* $I = \langle e_1 e_2 ..... e_k \rangle$*, also jedes Element aus* I *durch eine* R*-Linearkombination aus Elementen aus* E *darstellbar ist. Ist* E *eine endliche Menge so ist* I *endlich erzeugt.*

1.1.1.4 Polynomringe

Definition. *Der Polynomring $R[X]$ über einem Ring R ist:*

$$R[X] = \{\sum_{i \geq 0} a_i \cdot X^i \mid a_i \in R\}$$

Sei $f, g \in R$, $f = a_0 + a_1 t + \cdots + a_n t^n$ und $g = b_0 + b_1 t + \cdots + b_m t^m$, dann werden die Operationen $+$ und $\cdot$ wie folgt definiert:

$$f + g = (a_0 + b_0) + (a_1 + b_1)t + \cdots + (a_n + b_n)t^n$$

O.B.d.A. $m \leq n$, wobei $b_{m+1} = 0, ..., b_n = 0$

$$f \cdot g = c_0 + c_1 t + \cdots + c_{n+m} t^{n+m} \text{ mit } c_k = \sum_{i+j=k} a_i b_j$$

Der Polynomring in mehreren Variablen $X_1 ... X_n$ lässt sich, da $R[X_1, ..., X_{n-1}]$ ebenfalls ein Ring ist, induktiv definieren als $R[X_1, ..., X_{n-1}][X_n]$. Aus diesem Grund ist jedes Element $f \in R[X_1, ..., X_n]$ wie folgt darstellbar:

$$f = \sum_{i_1, i_2, ..., i_n \geq 0} a_{i_1} a_{i_2} \cdots a_{i_n} X_1^{i_1} X_2^{i_2} \cdots X_n^{i_n}$$

Definition. *Ein Polynom z heisst Monom, falls es folgende Gestalt aufweist: $z = a X_1^{i_1} X_2^{i_2} \cdots X_n^{i_n}$, $a \in R$.*

1.1.2 Graphenfärbungen

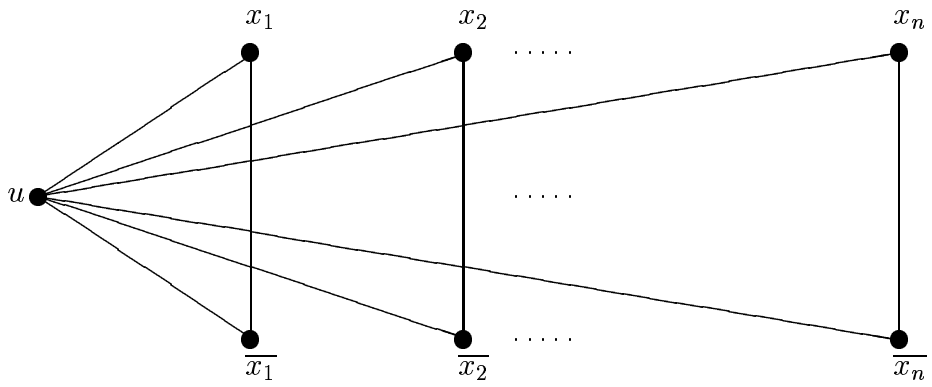
Definition. *Sei der Graph $G(V, E)$ gegeben. Eine Abbildung $f : V \to \{1...n\}$ heisst eine Färbung von G, falls $uv \in E \Rightarrow f(u) \neq f(v)$ für alle $u, v \in V$. G heisst k-färbbar, falls eine Färbung $f : V \to \{1...k\}$ existiert. Die chromatische Zahl $\chi(G)$ ist die kleinste Zahl von Farben, die eine Färbung von G erlaubt. G heisst k-chromatische, falls $\chi(G) = k$ ist.*

Wir wollen uns nun ein Bild darüber verschaffen wie schwer es ist eine 3-*Färbbung* für einen Graphen zu finden. Tatsächlich gilt:

Satz. *3-Färbbarkeit ist NP-vollständig*

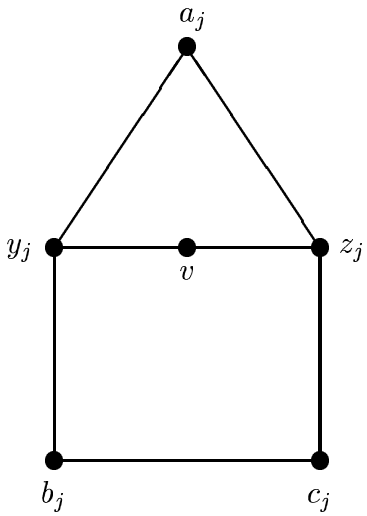
Beweis. Wir reduzieren 3-SAT auf 3-Färbbarkeit. Gegeben sei eine Formel in konjunktiver Normalform mit $F = K_1 \wedge K_2 \wedge .... \wedge K_l$. Wir können davon ausgehen, dass jede Klausel K_i drei Literale besitzt. Dies ist im Zweifelsfall durch wiederholen von Literalen möglich. Seien $x_1 ... x_n$ die Literale die in F vorkommen. Wir erzeugen nun einen Graphen G mit $2n + 5l + 2$ Knoten, welcher genau dann einfärbbar ist wenn F erfüllbar ist.

Konstruieren wir zunächst folgenden Teilgraphen Z von G mit $2n+1$ Knoten:



Haben wir 3 Farben,z.B. blau, wahr und falsch, so färben wir o.B.d.A u mit Farbe blau. Färbt man nun einen Knoten x_i mit wahr (falsch), so muss der Knoten $\overline{x_i}$ mit falsch (wahr) gefärbt werden.

Für jede Klausel K_j in Formel F konstruieren wir folgenden Teilgraphen P.

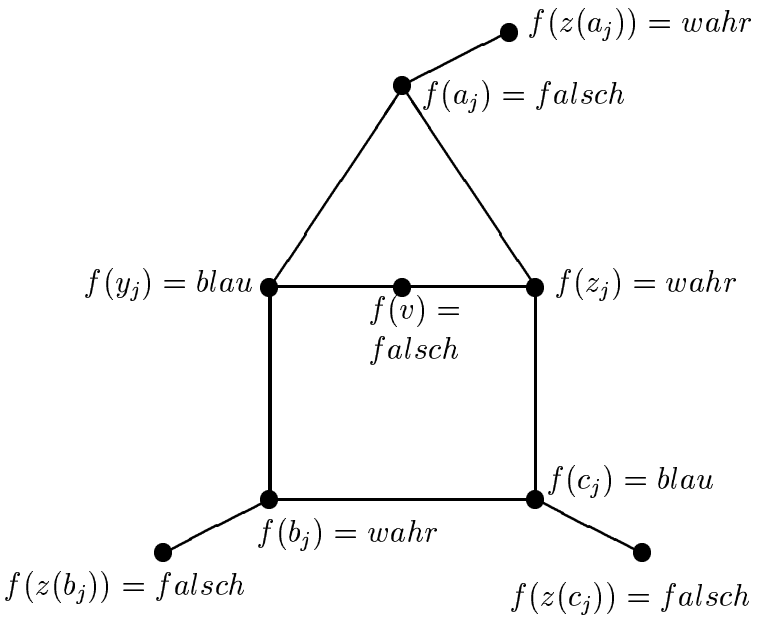


Sei $Kl = \{T_1 ... T_l\}$ die Menge dieser Teilgraphen bezüglich jeder Klausel. Wichtig ist, dass der Knoten v bei all diesen Teilgraphen derselbe ist. Daneben wird v mit Knoten u in Teilgraph Z aus Abbildung 1 verbunden. Für alle $T_j \in Kl$ ziehen wir nun Kanten von a_j, b_j, c_j zum Teilgraphen Z. Nämlich zu den Knoten x_u und $\overline{x_u}$, je nachdem ob diese in der Klausel vorkommen oder nicht.
Ein Beispiel: Sei $K_f = (\overline{x_2}, x_4, \overline{x_5})$ eine Klausel, so hätte man in G die Kanten $\{a_f, \overline{x_2}\}, \{b_f, x_4\}, \{c_f, \overline{x_5}\}$ hinzugefügt.

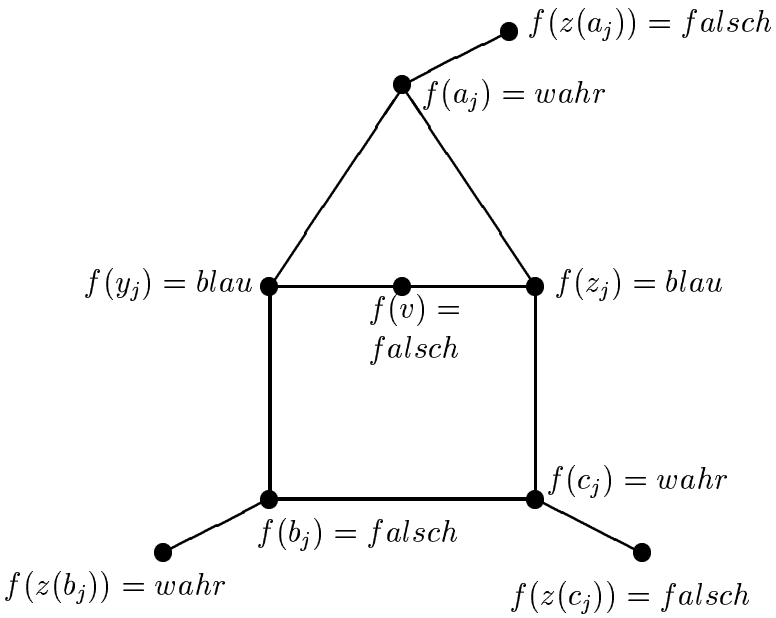
Proposition. *F ist erfüllbar $\Leftrightarrow$ G ist 3-färbbar*

$\Rightarrow$
Sei F nun erfüllbar. Dann kann man $f : V(G) \to \{blau, wahr, falsch\}$ so definieren: $f(u) = blau$, falls $x_i = wahr$ in F dann $f(x_i) = wahr$ und somit $f(\overline{x_i}) = falsch$ und sonst $f(x_i) = falsch$ und $f(\overline{x_i}) = wahr$. Dann können auch alle $T_j \in Kl$ zulässig gefärbt werden, da nicht alle Nachbarknoten von a_j, b_j, c_j in Z die Farbe falsch tragen. Die folgende Fallunterscheidung verdeutlicht dies, wobei $z(a_j)$ den Nachbarknoten von a_j in Z meint, entsprechendes ebenfalls für $z(b_j)$ und $z(c_j)$:

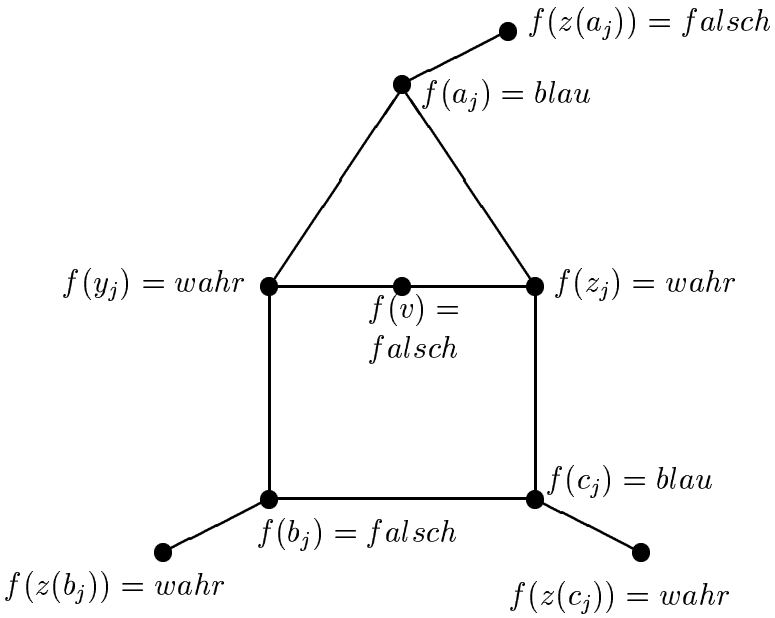
1. $f(z(a_j)) = wahr, f(z(c_j)) = falsch, f(z(b_j)) = falsch$



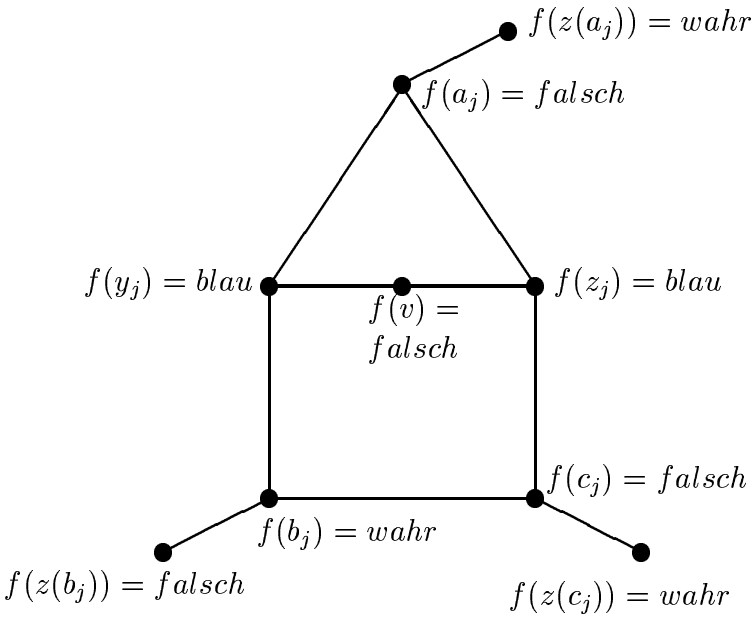
2. $f(z(a_j)) = falsch, f(z(c_j)) = falsch, f(z(b_j)) = wahr$



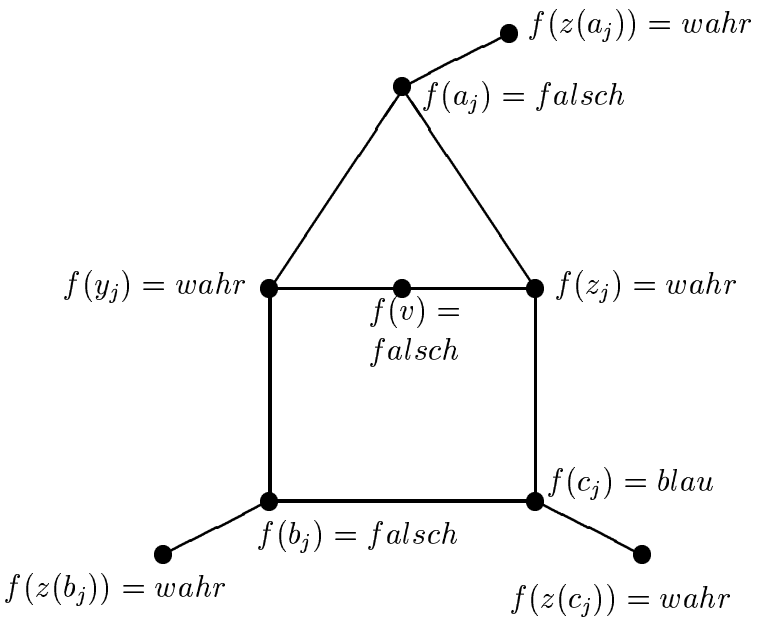
3. $f(z(a_j)) = falsch, f(z(c_j)) = wahr, f(z(b_j)) = wahr$



4. $f(z(a_j)) = wahr, f(z(c_j)) = wahr, f(z(b_j)) = falsch$



5. $f(z(a_j)) = wahr, f(z(c_j)) = wahr, f(z(b_j)) = wahr$

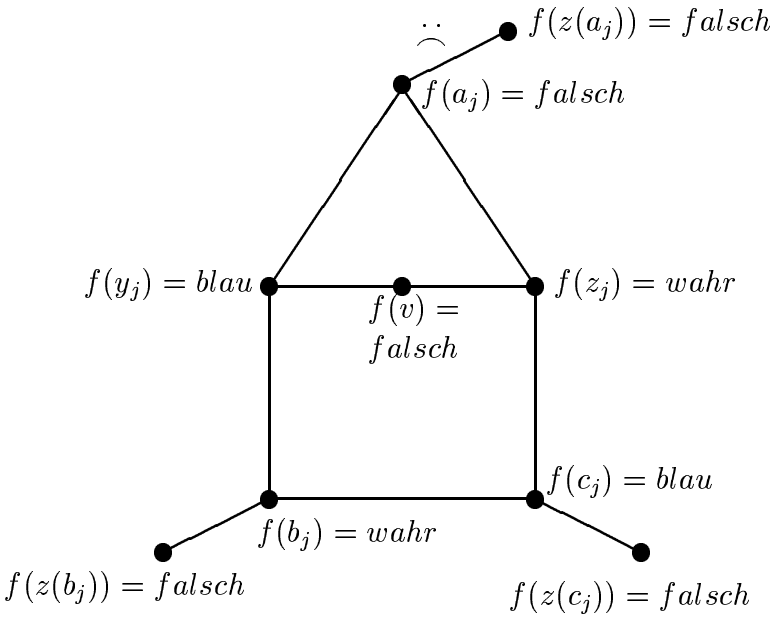


Der Fall $f(z(a_j)) = falsch, f(z(b_j)) = falsch, f(z(c_j)) = wahr$ ist äquivalent zu 2.
Der Fall $f(z(a_j)) = wahr, f(z(b_j)) = wahr, f(z(c_j)) = falsch$ ist äquivalent zu 4.

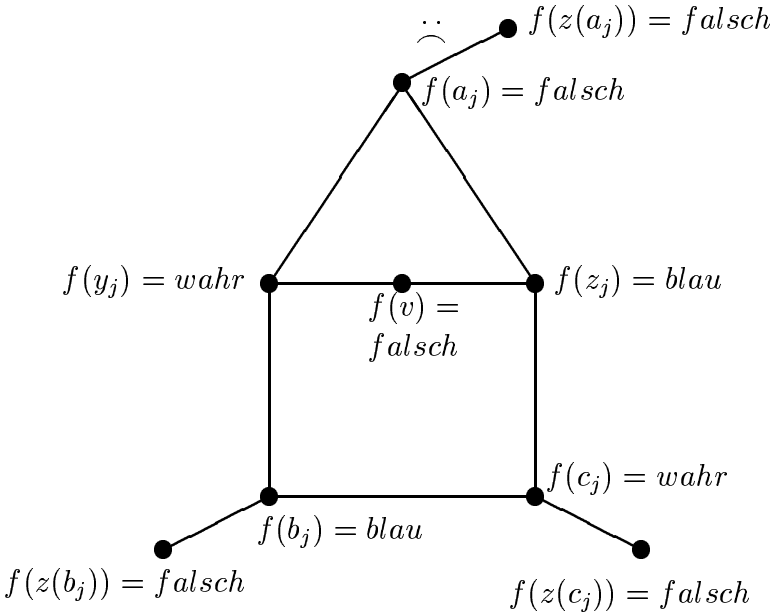
$\Leftarrow$

Sei G 3-färbbar, so sei o.B.d.A $f(u) = blau$ und $f(v) = falsch$. Wir wollen nun beweisen, dass dann die Zuweisung der Farben $wahr$ und $falsch$ zu x_i und $\overline{x_i}$ $(i = 1....n)$ einer erfüllenden Belegung entspricht. Nehmen wir das Gegenteil an. So gibt es eine Klausel $K_j = (x_a \vee x_b \vee x_c)$, so dass gilt $f(x_a) = f(x_b) = f(x_c) = falsch$. Für die Nachbarknoten von x_a, x_b, x_c, nämlich a_j, b_j, c_j, gilt dann, dass sie entweder auf $blau$ oder $wahr$ abgebildet werden *. Aus diesem Grund können y_j und z_j auch nur auf $wahr$ und $blau$ abgebildet werden. Dies bedeutet aber $f(a_j) = falsch$, da a_j Nachbar von y_j und z_j ist. Dies ist aber ein Widerspruch zu *.

1.Fall


$f(z(a_j)) = falsch$
$f(a_j) = falsch$
$f(y_j) = blau$
$f(z_j) = wahr$
$f(v) = falsch$
$f(c_j) = blau$
$f(b_j) = wahr$
$f(z(b_j)) = falsch$
$f(z(c_j)) = falsch$

2.Fall


$f(z(a_j)) = falsch$
$f(a_j) = falsch$
$f(y_j) = wahr$
$f(z_j) = blau$
$f(v) = falsch$
$f(c_j) = wahr$
$f(b_j) = blau$
$f(z(b_j)) = falsch$
$f(z(c_j)) = falsch$

□

1.2 "Polly Cracker"

1.2.1 Funktionsweise von "Polly Cracker"

"Polly Cracker" ist ein Public-Key-Kryptosystem. Wie jedes Public-Key-Kryptosystem besitzt es ein Alphabet, einen öffentlichen und einen geheimen Schlüssel, sowie ein Verschlüsselungsverfahren:

- Das Alphabet ist ein endlicher Körper $\mathbb{F}$. Wäre $\mathbb{F}= \mathbb{F}_2$, so wäre die Nachricht immer ein einzelnes Bit.
- Der öffentliche Schlüssel von Alice ist eine endliche Menge $B = \{q_j\}$von Polynomen in $\mathbb{F}[T]$, wobei $T = \{t_i\}_{i=1}^{n}$ eine Menge aus Variabeln darstellt.
- Der geheime Schlüssel von Alice ist ein Vektor $z \in \mathbb{F}^n$, wobei für alle $q \in B$ gilt $q(z) = 0$.
- Möchte nun Bob an Alice eine Nachricht $m \in \mathbb{F}$ schicken, so wendet er folgendes Verschlüsselungsverfahren an: er erzeugt ein Element p aus dem Ideal $J \subset \mathbb{F}[T]$ welches durch B erzeugt wird:

$$p = \sum_{j=1}^{r} h_j p_j, \ \textit{wobei}\ h_j \in \mathbb{F}[T]$$

Er verschlüsselt die Nachricht m, indem er $c = p + m$ bildet.
Erhält Alice die verschlüsselte Nachricht c, so muss sie sie nur an der Stelle z auswerten und erhält die Originalnachricht m:

$$c(z) = p(z) + m = \sum_{j=1}^{r} h_j(z) q_j(z) + m = \sum_{j=1}^{r} h_j(z) 0 + m = m \tag{1.1}$$

1.2.2 "Polly Cracker" und NP-vollständige kombinatorische Probleme

Hierfür nimmt man spezielle Polynome, die genau dort Nullstellen haben, wo sie mit Hilfe der Lösung des kombinatorischen Problems ausgewertet werden. Die gemeinsamen Nullstellen dieser Polynome zu kennen, ist dann äquivalent dazu, die Lösung des Problems zu kennen. Es ist also ebenso schwer diese Nullstellen zu finden, wie eine Lösung für das Problem. Dies wird dann ausserordentlich schwer, wenn wir den Polynomen eine schwere Instanz eines NP-vollständigen Problems zu Grunde legen.
Wir verdeutlichen dies hier am Beispiel des NP-vollständigen Problems der '3-Färbung von Graphen'.

1.2.2.1 3-Färbung von Graphen

- öffentlicher Schlüssel: Ein Graph G(V,E).
- geheimer Schlüssel: eine gültige 3-Färbung für G, also eine Funktion $f : V(G) \to \{1, 2, 3\}$ und $\forall uv \in E(G)$ gilt $f(u) \neq f(v)$.
- Eine Basis $B = B_1 \cup B_2 \cup B_3$ in den Variablenmenge $T = \{t_{vi} \mid v \in V(G), 1 \leq i \leq 3\}$ ist gegeben durch:
  - $B_1 = \{t_{v1} + t_{v2} + t_{v3} - 1 \mid v \in V(G)\}$.
  - $B_2 = \{t_{vi}t_{vj} \mid v \in V(G), 1 \leq i < j \leq 3\}$.
  - $B_3 = \{t_{ui}t_{vi} \mid uv \in E(G), 1 \leq i \leq 3\}$.

Wird $t_{vi} = 1$, $v \in V(G)$ $i \in \{1, 2, 3\}$, gesetzt bedeutet dies Knoten v bekommt Farbe i.

Proposition. *Es gibt eine gültige Färbung $f : V(G) \to \{1, 2, 3\} \Leftrightarrow$ Es gibt eine Belegungsfunktion $g : K[T] \to K$, so dass für alle $P \in \langle B \rangle$ gilt $g(P) = 0$*

Beweis: $\Rightarrow$
Sei $G = (V, E)$ Graph, $B = B_1 \cup B_2 \cup B_3$, f eine gültige Färbung für G.

Man definiert nun eine Belegungsfunktion $g : K[T] \to \mathcal{K}$:

$$g(p) = \begin{cases} \sum_{i=1}^{n} a_i \; g(M_i) & : \quad p = \sum_{i=1}^{n} a_i M_i, \;\; M_i \; normiertes \; Monom, \;\; a_i \in K \\ \prod_{i=1}^{n} g(m_i) & : \quad m_i \in T, \;\; p = \prod_{i=1}^{n} m_i \\ 1 & : \quad p = t_{vi}, \;\; t_{vi} \in T, \;\; f(v) = i \\ 0 & : \quad sonst \end{cases}$$

Man prüft leicht nach:

$$\forall \; p, q \in K[T] \; g(p)g(q) = g(pq) \quad (\star) \tag{1.2}$$

Sei nun $p \in \langle B \rangle$, dann hat p folgende Darstellung:

$$p = \overbrace{\sum_{i=1}^{n_1} x_i b_{1i}}^{p_1} + \overbrace{\sum_{j=1}^{n_2} y_j b_{2j}}^{p_2} + \overbrace{\sum_{l=1}^{n_3} z_l b_{3l}}^{p_3}$$

wobei $b_{1i} \in B_1$, $b_{2j} \in B_2$, $b_{3l} \in B_3$ und $x_i, y_j, z_l \in K[T]$.

Betrachten wir nun die Polynome in B genauer:

1. Sei $z \in B_1$:
   Also $z = t_{v1} + t_{v2} + t_{v3} - 1$. Da f eine gültige Färbung ist, gilt $f(v) = i$ nur für ein $i \in \{1, 2, 3\}$
   $\Rightarrow g(z) = 0$, mit $(\star)$ folgt $g(p_1) = 0$.

2. Sei $z \in B_2$:
   Also $z = t_{vi} t_{vj}$, $i, j \in \{1, 2, 3\}$. Da f eine gültige Färbung ist, existiert nur ein $z \in \{1, 2, 3\}$ mit $g(t_{vz}) = 1$. O.B.d.A. gilt dann $g(t_{vi}) = 0$. Dann folgt $g(t_{vi} t_{vj}) = g(t_{vi})g(t_{vj}) = 0$, mit $(\star)$ folgt $g(p_2) = 0$.

3. Sei $z \in B_3$:
   Also $z = t_{ui} t_{vi}$, wobei für die entsprechenden Knoten u, v gilt $\{u.v\} \in E(G)$. Da f eine gültige Färbung ist, gilt $f(u) \neq f(v)$. Hieraus folgt: $g(t_{ui}) = 0$ und $g(t_{vi}) = 0$ oder O.B.d.A. $g(t_{ui}) = 1$ und $g(t_{vi}) = 0$. Aus beiden Fällen folgt $g(z) = 0$ und mit $(\star)$ folgt $g(p_3) = 0$.

Aus 1,2,3 folgt dann $g(p) = 0$. Also ist g Belegungsfunktion mit den gesuchten Eigenschaften.

$\Leftarrow$
Sei $g : K[T] \to K$ eine Belegungsfunktion, so dass für alle $p \in \langle B \rangle$ gilt $g(p) = 0$. Insbesondere für alle $z \in B$ gilt $g(z) = 0$:

1. Sei $z \in B_2$:
   Also $z = t_{vi}t_{vj}$. Aus $g(t_{vi}t_{vj}) = 0$ folgt: $g(t_{vi}) = 0$ und $g(t_{vj}) = 0$ oder O.B.d.A. $g(t_{vi}) = 0$ und $g(t_{vj}) = c$, $c \in N$.

2. Sei $z \in B_1$:
   Also $z = t_{v1} + t_{v2} + t_{v3} - 1$.
   Aus 1. wissen wir, dass für nur ein $l \in \{1, 2, 3\}$ gilt $g(t_{vl}) \neq 0$. Gäbe es k_n, k_m mit $g(t_{vk_n}) = c_1$ und $g(t_{vk_m}) = c_2$, so wäre $g(t_{vn}t_{tm}) \neq 0$, ein Widerspruch.
   O.B.d.A. $g(t_{v3}) \neq 0 \Rightarrow g(t_{v1}+t_{v2}+t_{v3}-1) = g(t_{v1})+g(t_{v2})+g(t_{v3})-1 = g(t_{v3}) - 1 = 0 \Rightarrow g(t_{v3}) = 1$.

$\Rightarrow \forall d \in T\ g(d) = 1 \vee g(d) = 0$.

Sei nun $f : V(G) \to \{1, 2, 3\}$ wie folgt definiert:

$$f(v) = \begin{cases} 1 & : \quad g(t_{v1}) = 1 \\ 2 & : \quad g(t_{v2}) = 1 \\ 3 & : \quad g(t_{v3}) = 1 \end{cases}$$

Diese Funktion ist wohldefiniert, denn aus 2. folgt, dass $g(t_{vi}) = 1$ für genau ein $i \in \{1, 2, 3\}$ gilt.

3. Sei $z \in B_3$:
   Also $z = t_{vi}t_{ui}$. Aus 1 und 2 folgt: $g(t_{vi}) = 0$ und $g(t_{vj}) = 0$ oder O.B.d.A. $g(t_{vi}) = 0$ und $g(t_{vj}) = 1$.

   Für alle $\{u, v\} \in E(G)$ gilt dann $f(u) \neq f(v)$.
   Angenommen es gilt $f(u) = f(v)$: $O.B.d.A. g(t_{v1}) = g(t_{u1}) = 1 \Rightarrow g(t_{v1}t_{u1}) = 1$, ein Widerspruch.

Also folgt, dass f eine gültige Färbung für G ist.

□

1.2.3 Schwachstellen von "Polly Cracker"

Hier soll ein kurzer, exemplarischer Einblick in die "Verletzlichkeit" von "Polly Cracker" gegeben werden. Dieses Thema ist hiermit bei weitem noch nicht erschöpft und der interessierte Leser sei auf [5] und [6] verwiesen.

Sei nun $c = p + \lambda$ das Chiffrat, welches Bob an Alice schickt. Wertet man es an der Stelle Null aus, so gilt:

$$c(0) = p(0) + \lambda = \sum_{j=1}^{r} h_i(0) q_i(0) + \lambda$$

Ziel soll es sein, die skalaren Anteile $h_i(0)$ der h_i zu bestimmen, um dann auf den Klartext m des Chiffrats c schliessen zu können.

$$Sei\ M(t) = \{a \mid a\ Monom\ in\ Polynom\ t\}$$

Wir fordern nun:

$$\forall m_{h_i} \in M(h_i) \wedge m_{h_i} \neq 1 \wedge \forall m_{q_i} \in M(q_i)$$

gelte:

$$m_{h_i} m_{q_i} \notin \bigcup_{j=1}^{r} M(q_i)$$

Dies bedeutet also, dass die Multiplikation eines Monoms $m_{h_i} \in M(h_i)$ mit dem zugehörigen Monom $m_{q_i} \in M(q_i)$ nicht zu einem Monom in einem beliebigen q_j führt.
Im weiteren setzten wir o.B.d.A voraus, dass die Menge der öffentlichen Polynome $B = \{q_1 .... q_r\}$ linear unabhängig ist. Sonst reduzieren wir B um die benötigte Anzahl an abhängigen Elementen.
Richten wir nun unser Augenmerk auf ein beliebiges nichtkonstantes Monom:

$$m \in \bigcup_{i=1}^{r} M(q_i),\ m \neq 1$$

Mit obigen Voraussetzungen gilt dann:

$$Koeff_c(m) = \sum_{i=1}^{r} Koeff_{q_i}(m) h_i(0)$$

wobei $Koeff_c(m)$ den Koeffizienten von m bezüglich c darstellt. Stellt man so eine Gleichung für jedes $m \in \bigcup_{i=1}^{r} M(q_i) = \{m_1....m_t\}$ auf, so ergibt sich ein Gleichungssystem:

$$\underbrace{\begin{pmatrix} Koeff_{q_1}(m_1) & Koeff_{q_2}(m_1) & \cdots & Koeff_{q_r}(m_1) \\ Koeff_{q_1}(m_2) & Koeff_{q_2}(m_2)) & \cdots & Koeff_{q_r}(m_2) \\ \vdots & \vdots & \ddots & \vdots \\ Koeff_{q_1}(m_t) & Koeff_{q_2}(m_t) & \cdots & Koeff_{q_r}(m_t) \end{pmatrix}}_{A} \cdot \begin{pmatrix} h_1(0) \\ h_2(0) \\ \vdots \\ h_r(0) \end{pmatrix}$$

$$= \begin{pmatrix} Koeff_c(m_1) \\ Koeff_c(m_2) \\ \vdots \\ Koeff_(m_t) \end{pmatrix}$$

Es muss nun die Frage beantwortet werden, ob die Spalten der Matrix A linear unabhängig sind. Denn dann wäre das Gleichungssystem eindeutig lösbar.
Zunächst beobachtet man für jedes q_i das folgendes gilt:

$$q_i = q_i(0) + \sum_{j=1}^{t} Koeff_{q_i}(m_j) \cdot m_j \Leftrightarrow \sum_{i=1}^{r} q_i - q_i(0) = \sum_{i=1}^{r} \sum_{j=1}^{t} Koeff_{q_i}(m_j) \cdot m_j$$

Besitzen die Spalten von A nun eine nichttriviale Nulllinearkombination, so würde gelten:

$$\sum_{i=1}^{r} \omega_i \cdot (q_i - q_i(0)) = 0$$

wobei die ω_i die entsprechenden Koeffizienten der Nulllinearkombination wären.
Diesen Ausdruck wertet man nun an der, nach Voraussetzung existenten, gemeinsamen Nullstelle der q_i aus und es folgt:

$$0 = \sum_{i=1}^{r} \omega_i \cdot (q_i - q_i(0)) = -\sum_{i=1}^{r} \omega_i \cdot q_i(0)$$

$$\Leftrightarrow \sum_{i=1}^{r} \omega_i \cdot q_i = \left(\sum_{i=1}^{r} \omega_i \cdot (q_i - q_i(0)) \right) + \left(\sum_{i=1}^{r} \omega_i \cdot q_i(0) \right) = 0$$

Dies wäre aber eine nichttriviale Nulllinearkombination der q_i, welches im Widerspruch zu ihrer linearen Unabhängigkeit stehen würde. Andererseits sind in $\bigcup_{i=1}^{r} M(q_i)$ auch mindestens r Monome enthalten da sonst die q_i linear abhängig wären.

Es kommt also, bezüglich Sicherheit, in hohem Masse darauf an, wie die h_i gebaut sind. In [5],[6] gibt es noch weitere Attacken, die auf einer unvorteilhaften Auswahl von h_i basieren. Es sollte also keines Falls dem Zufall überlassen werden die Polynome h_j auszuwählen.

Kapitel 2

Lateinische Quadrate

Definition. *Ein lateinisches Quadrat der Grösse n ist eine $n \times n$-Matrix, in der in jeder Zeile und jeder Spalte jedes Element aus $\{1....n\}$ exakt einmal vorkommt. Ein partielles lateinisches Quadrat ist ein lateinisches Quadrat mit der Ausnahme, dass es einen Eintrag 0 gibt der obige Eigenschaft verletzen darf. Dieser Eintrag kann sinngemäss als noch nicht besetzt betrachtet werden. Wenn wir im folgenden von Farben sprechen so meinen wir Elemente aus $\{1...n\}$.*

Das Problem, ein partielles lateinisches Quadrat zu komplettieren, ist NP-vollständig. Im weiteren Verlauf steht KLQ genau für dieses Problem. Wir wollen hier nun eine Codierung in SAT-Formeln entwickeln und einen kurzen Überblick bezüglich des NP-Vollständigkeitsnachweisses geben.

2.1 Codierung von KLQ in SAT-Formeln

Wir bemühen uns im folgenden Abschnitt eine Formulierung des Problems der Komplettierung lateinischer Quadrate in SAT-Formeln aufzustellen.

Sei $L = \{(i,j) \mid 1 \leq i,j \leq n\}$ ein partielles lateinisches Quadrat. Sei $C = \{1.....n\}$ die zur Verfügung stehenden Farben. Sei $f : L \to C \cup \{0\}$ die durch das lateinische Quadrat definierte partielle Färbungsfunktion.
Sei $F = \{(i,j) \mid f((i,j)) = 0\}$.
Sei $ZF(i) = \{f((i,p)) \mid 1 \leq p \leq n \wedge f((i,p)) \neq 0\}$.
Sei $FZF(i) = \mathcal{C}$ ZF(i), also die Farben, die in der Zeile i nicht vorkommen.
Sei $SF(j) = \{f((p,j)) \mid 1 \leq p \leq n \wedge f((p,j)) \neq 0\}$.

Sei $FSF(j) = \mathcal{C}$ SF(j), also die Farben, die in der Spalte j nicht vorkommen.

Wir wollen nun eine SAT-Formel in konjunktiver Normalform generieren. Wir erzeugen die Klauseln der SAT-Formel, die zu L korrespondieren soll, wie folgt:

- Für jedes $a_{te} \in F$ fügt man folgenden Klauseln hinzu:
  Sei $FF((t,e)) = FZF(t) \cap FSF(e) = \{z_1 ... z_m\}$.
  Füge nun die Klausel $(x_{tez_1} \vee ... \vee x_{tez_m})$ hinzu.
  Wird $x_{tez_k} = 1$ gesetzt, so bedeutet dies, dass $f((t,e)) = z_k$, also die Zelle a_{te} Farbe z_k zugewiesen bekommt. a_{te} kann also alle noch nicht benutzen Farben, bzgl. Zeile und Spalte, zugewiesen bekommen.

- Für alle $(t,e) \in F$, $u \in FZF(t)$, $(t,s) \in F$, $e \neq s$, füge hinzu:
  $(x_{teu} \Rightarrow \neg x_{tsu}) = (\neg x_{teu} \vee \neg x_{tsu})$
  Wird also $x_{teu} = 1$ gesetzt, so können alle anderen freien Zellen in der gleichen Zeile diese Farbe nicht mehr zugewiesen bekommen.

- Für alle $(t,e) \in F$, $u \in FSF(e)$, $(r,e) \in F$, $r \neq t$, füge hinzu:
  $(x_{teu} \Rightarrow \neg x_{rsu}) = (\neg x_{teu} \vee \neg x_{rsu})$
  Wird also $x_{teu} = 1$ gesetzt, so können alle anderen freien Zellen in der gleichen Spalte diese Farbe nicht mehr zugewiesen bekommen.

Dies definiert uns eine Reduktion von KLQ nach SAT.

2.2 Transformation in Triangulierungsproblem

Wir transformieren nun die KLQ in das Finden einer Triangelpartition für einen Graphen $G(V,E)$. Eine Triangelpartition ist eine Zerlegung von E in disjunkte Kantenmengen, die jeweils einen K_3 bzw. ein Dreieck formen.

Sei $L = \{(i,j) \mid 1 \leq i,j \leq n\}$ ein partiell gefärbtes lateinisches Quadrat.
$V(G)$ wird nun so definiert:

Sei $R = \{r_i \mid Zeile\ i\ enthält\ freie\ Zelle\}$
Sei $C = \{c_i \mid Spalte\ i\ enthält\ eine\ freie\ Zelle\}$
Sei $E = \{e_k \mid Farbe\ k\ taucht\ weniger\ als\ n-mal\ auf\}$
Sei $V(G) = R \cup C \cup E$.

E(G) wird folgendermassen definiert:

1. $(i,j) \in F \Leftrightarrow \{r_i, c_j\}$ in $E(G)$
2. In Zeile i kommt Farbe k nicht vor $\Leftrightarrow \{r_i, e_k\} \in E(G)$
3. In Spalte j kommt Farbe k nicht vor $\Leftrightarrow \{c_j, e_k\} \in E(G)$

Dieser Graph ist der *defect* eines lateinischen Quadrates L. Im folgenden sei mit $\triangle(a,b,c)$ das Dreieck $\{\{a,b\}\{b,c\}\{c,a\}\}$ bezeichnet, welches durch die Knotenmenge $\{a,b,c\}$ definiert wird.

Proposition. *Ein lateinisches Quadrat L besitzt eine gültige Komplettierung* $g : F \to \{1...n\} \Leftrightarrow$ *der defect von L besitzt eine Triangelpartition in E.*

Beweis: $\Rightarrow$
Sei $TP = \{\triangle(r_i, c_j, e_{g(a_{ij})}) \mid (i,j) \in F\}$. Es gilt für alle $a, b \in TP$, dass $a \cap b = \emptyset$. Denn sonst: Sei O.B.d.A. $a = \triangle(r_i, c_j, e_{g((i,j))}), b = (r_i, c_p, e_{g((i,p))})$. Aus $a \cap b \neq \emptyset$ folgt dann $g((i,j)) = g((i,p))$. Widerspruch.
Ebenfalls gibt es für alle $e \in E(G)$ genau ein $c \in TR$, so dass $e \in c$. Angenommen es gäbe ein $e \in E(G)$, welches in keinem Dreieck aus TP vorkommt:

1. $e = \{r_i, c_j\} \Rightarrow (i,j) \notin F$. Widerspruch
2. $e = \{r_i, e_k\} \Rightarrow$ Für alle j, $1 \leq j \leq n$, gilt $\triangle(r_i, c_j, e_k) \notin TP$.
   (a) $\{r_i, c_j\}$ ist keine Kante. Dann $(i,j) \notin F$ und somit ist a_{ij} mit einer Farbe ungleich k gefärbt.
   (b) $\{r_i, c_j\}$ ist Kante. Dann $g((i,j)) \neq k$, da $\triangle(r_i, c_j, e_k) \notin TP$.

   Also taucht Farbe k nicht in Zeile i auf. Widerspruch.
3. $e = \{c_j, e_k\} \Rightarrow$ Für alle i, $1 \leq i \leq n$ gilt $\triangle(r_i, c_j, e_k) \notin TP$.
   (a) $\{r_i, c_j\}$ ist keine Kante. Dann $(i,j) \notin F$ und somit ist a_{ij} mit einer Farbe ungleich k gefärbt.
   (b) $\{r_i, c_j\}$ ist Kante. Dann $g((i,j)) \neq k$, da $\triangle(r_i, c_j, e_k) \notin TP$.

   Also taucht Farbe k nicht in Spalte i auf. Widerspruch.

$\Leftarrow$
Sei also T eine Triangelpartition. Sei $d = \triangle(a, b, c) \in T$. Nach Konstruktion kann ein Dreieck nur aus 3 Knoten bestehen welche jeweils aus R, C und E sind. Sei O.B.d.A. $d = \triangle(r_k, c_j, e_p)$. Jedes Dreieck symbolisiert also eine Belegung für ein Loch.
Es folgt: Für alle $r_i, c_j \in V(G)$ gibt es $\triangle(r_i, c_j, e_k) \in T$. Setzte dann $g((i, j)) = k$. Angenommen g wäre eine ungültige Komplettierung, dann gibt es O.B.d.A. (i, j), (i, p), so dass $g((i, j)) = g((i, p)) = l \Rightarrow$ es gibt $\triangle(r_i, c_j, e_l), \triangle(r_i, c_p, e_l) \in T$, so dass $\triangle(r_i, c_j, e_l) \cap \triangle(r_i, c_p, e_l) \neq \emptyset$. Widerspruch.

□

2.3 NP-Vollständigkeit

Der Nachweis der NP-Vollständigkeit für KLQ erstreckt sich über zwei Arbeiten. Es werden deshalb nur die Verweise genannt. In [9] wird nachgewiesen, dass $Triangelpartition$ NP-vollständig ist, indem es auf das Erfüllbarkeitsproblem SAT reduziert wird. [8] zeigt, dass jeder uniforme tripartite Graph der $defect$ eines partiellen lateinischen Quadrats ist.

Ein tripartiter Graph $G = (V_1 \cup V_2 \cup V_3, E)$ ist uniform, falls jeder Knoten folgende Eigenschaft besitzt: Ein Knoten aus V_1 (oder V_2, oder V_3) besitzt die gleiche Anzahl an Nachbarn in V_2 und V_3 (oder V_1 und V_3, oder V_1 und V_2).

Jeder Graph der ein Triangelpartition der Kanten besitzt ist auch uniform. Denn dann ist er trianguliert. Färben wir nun bei einem Dreieck die Knoten mit jeweils 3 Farben, so lässt sich diese 3 Färbung auf den gesamten Graphen ausdehnen. Sei nun O.B.d.A. $v \in V_1$. Dieser Knoten gehört zu einer bestimmten Anzahl von Dreiecken. Die weiteren Knoten in diesen Dreiecken sind alle aus $V_2 \cup V_3$. Jedes Dreieck fügt der Nachbarschaft von v jeweils einen Konten aus V_2 und einen aus V_3 hinzu. Also ist die Anzahl der Knoten in der Nachbarschaft von v aus V_2 und V_3 gleich.
Man kann also jeden Graph mit Triangelpartition auf ein partielles lateinisches Quadrat reduzieren. Also ist KLQ NP-vollständig.

2.4 Realisierung von "Polly Cracker" mit KLQ

Damit ein kombinatorisches Problem kryptologisch interessant wird, benötigt man Polynome, die genau dort Nullstellen haben, wo sie mit Hilfe der Lösung des Problems ausgewertet werden. Wir haben bereits gesehen wie diese Polynome im Falle der 3-Färbung von Graphen beschaffen sein müssen. Analog werden wir nun die Polynomcodierung von KLQ in "Polly Cracker" entwickeln.

2.4.1 Direkte Formulierung

Sei $T = \{t_{ijk} \mid (i,j) \in F \wedge k \in FF((i,j))\}$ Variablenmenge. $t_{ijk} = 1$ bedeutet wieder, dass t_{ij} mit Farbe k gefärbt wird. Es gelten ebenfalls die Begriffsbildungen des letzten Paragraphen.

- $B_1 = \{-(\sum_{k \in FF((i,j))} t_{ijk}) + 1 \mid (i,j) \in F\}$
- $B_2 = \{t_{iju}t_{ijp} \mid (i,j) \in F \; u,p \in FF((i,j))\}$
- $B_3 = \{t_{iju}t_{izu} \mid (i,z),(i,j) \in F \; u \in FF((i,j)) \wedge u \in FF((i,z))\}$
- $B_4 = \{t_{iju}t_{pju} \mid (i,j),(p,j) \in F \; u \in FF((i,j)) \wedge u \in FF((p,j))\}$

$B = B_1 \cup B_2 \cup B_3 \cup B_4$.

Definition. *Sei* $L = \{(i,j) \mid 1 \leq i,j \leq n\}$ *und* $t : L \to \{0,1...n\}$ *die, durch das partiell gefärbte lateinische Quadrat definierte, partielle Färbungsfunktion. Eine Funktion* $f : F \to \{1...n\}$ *ist dann eine gültige Komplettierung, falls für*

$$\mathcal{H}((i,j)) = \begin{cases} f((i,j)) & : \quad t((i,j)) = 0 \\ t((i,j)) & : \quad sonst \end{cases}$$

$$\forall (i,j) \in L \; \forall (i,z) \in L, \;\; j \neq z, \quad \mathcal{H}((i,j)) \neq \mathcal{H}((i,z)) \qquad (\spadesuit)$$

$$\forall (i,j) \in L \; \forall (l,j) \in L, \;\; i \neq l, \quad \mathcal{H}((i,j)) \neq \mathcal{H}((l,j)) \qquad (\spadesuit\spadesuit)$$

gilt.

Proposition. *Es gibt eine gültige Komplettierung* $f : F \to \{1...n\} \Leftrightarrow$ *Es gibt eine Belegungsfunktion* $g : K[T] \to K$, *so dass für alle* $P \in \langle B \rangle$ *gilt* $g(P) = 0$.

Beweis: $\Rightarrow$
Sei $L = \{(i,j) \mid 1 \leq i,j \leq n\}$ ein partiell gefärbtes Quadrat und
$f : F \to \{1...n\}$ eine gültige Komplettierung. Man definiert nun eine Belegungsfunktion $g : K[T] \to K$:

$$g(p) = \begin{cases} \sum_{i=1}^{n} a_i \; g(M_i) & : \; p = \sum_{i=1}^{n} a_i M_i, \; M_i \; normiertes \; Monom, \; a_i \in K \\ \prod_{i=1}^{n} g(m_i) & : \; m_i \in T, \; p = \prod_{i=1}^{n} m_i \\ 1 & : \; p = t_{iju}, \; t_{iju} \in T, \; f((i,j)) = u \\ 0 & : \; sonst \end{cases}$$

Man prüft leicht nach:

$$\forall \; p, q \in K[T] \; g(p)g(q) = g(pq) \qquad (\diamondsuit)$$

Sei nun $p \in \langle B \rangle$ dann hat p folgende Darstellung:

$$p = \overbrace{\sum_{i=1}^{n_1} x_i b_{1i}}^{p_1} + \overbrace{\sum_{j=1}^{n_2} y_j b_{2j}}^{p_2} + \overbrace{\sum_{l=1}^{n_3} z_l b_{3l}}^{p_3} + \overbrace{\sum_{k=1}^{n_4} v_k b_{4k}}^{p_4}$$

wobei $b_{1i} \in B_1$, $b_{2j} \in B_2$, $b_{3l} \in B_3$, $b_{4k} \in B_4$ und $x_i, y_j, z_l v_k \in K[T]$.
Betrachten wir nun die Polynome in B genauer:

1. Sei $z \in B_1$:
   Also $z = -(\sum_{k \in FF((i,j))} t_{ijk}) + 1$.
   Für alle $(i,j) \in L$ gibt es $k_d \in FF((i,j))$, so dass gilt $f((i,j)) = k_d$
   $\Rightarrow g(t_{ijk_d}) = 1$ und $\forall k_u \in FF((i,j)), u \neq d, g(t_{ijk_u}) = 0$
   $\Rightarrow g(z) = 0 \Rightarrow g(p_1) = 0$

2. Sei $z \in B_2$:
   Also $z = t_{iju} t_{ijp}$
   Für alle $(i,j) \in L$ gibt es $k_d \in FF((i,j))$, so dass wieder gilt
   $f((i,j)) = k_d$
   $\Rightarrow g(t_{iju}) = 0$ und $g(t_{ijp}) = 0$ oder O.B.d.A. $g(t_{iju}) = 1$ und $g(t_{ijp}) = 0$
   $\Rightarrow g(z) = 0 \Rightarrow g(p_2) = 0$

3. Sei $z \in B_3$:
   Also $z = t_{iju}t_{izu}$
   Da gilt: $f((i,j)) \neq f((i,z)) \Rightarrow g(t_{iju}) = 0$ und $g(t_{izu}) = 0$ oder O.B.d.A. $g(t_{iju}) = 1$ und $g(t_{izu}) = 0 \Rightarrow g(z) = 0 \Rightarrow g(p_3) = 0$

4. Sei $z \in B_4$:
   Also $z = t_{iju}t_{pju}$
   Da gilt: $f((i,j)) \neq f((p,j)) \Rightarrow g(t_{iju}) = 0$ und $g(t_{pju}) = 0$ oder O.B.d.A. $g(t_{iju}) = 1$ und $g(t_{pju}) = 0 \Rightarrow g(z) = 0 \Rightarrow g(p_4) = 0$

Aus 1,2,3,4 folgt $g(p) = 0$. g ist also eine Belegungsfunktion mit den gewünschten Eigenschaften.

$\Rightarrow$
Sei $g : K[T] \rightarrow K$ eine Belegungsfunktion, so dass für alle $p \in \langle B \rangle$ gilt $g(p) = 0$. Insbesondere für alle $z \in B$ gilt $g(z) = 0$:

1. Sei $z \in B_2$: $\quad (\dagger)$
   Also $z = t_{iju}t_{ijp}$
   Da $g(z) = 0 \Rightarrow g(t_{iju}) = 0$ und $g(t_{ijp}) = 0$ oder O.B.d.A. $g(t_{iju}) = 0$ und $g(t_{ijp}) = c,\ c \in N$

2. Sei $z \in B_1$: $\quad (\dagger\dagger)$
   Also $z = -(\sum_{k \in FF((i,j))} t_{ijk}) + 1$. Sei $FF((i,j)) = \{k_1 ... k_g\}$.
   Es existiert exakt nur ein $k_s \in FF((i,j))$ mit $g(t_{ijk_s}) \neq 0$. Gäbe es k_n, k_m mit $g(t_{ijk_n}) = c_1$ und $g(t_{ijk_m}) = c_2$, so folgt $g(t_{ijk_n}t_{ijk_m}) \neq 0$, ein Widerspruch. O.B.d.A. $g(t_{ij1}) \neq 0 \Rightarrow g(-(\sum_{k \in FF((i,j))} t_{ijk}) + 1) = -(\sum_{k \in FF((i,j))} g(t_{ijk})) + 1 = -g(t_{ij1}) + 1 = 0 \Rightarrow g(t_{ij1}) = 1$.

$\Rightarrow \forall d \in T\ g(d) = 1 \vee g(d) = 0$.

Sei nun $f : F \rightarrow \{1...n\}$ wie folgt definiert:

$$f((i,j)) = \begin{cases} k_1 & : \quad g(t_{ijk_1}) = 1 \wedge k_1 \in FF((i,j)) \\ k_2 & : \quad g(t_{ijk_2}) = 1 \wedge k_2 \in FF((i,j)) \\ \vdots & : \quad \vdots \\ k_g & : \quad g(t_{ijk_g}) = 1 \wedge k_g \in FF((i,j)) \end{cases}$$

Diese Funktion ist wohldefiniert, denn aus 2 folgt dass $g(t_{ijk_u}) = 1$ für genau ein $k_u \in FF((i,j))$ gilt.

3. Sei $z \in B_3$:
   Also $z = t_{iju}t_{izu}$
   Aus 1 und 2 folgt mit $g(z) = 0$ entweder $g(t_{iju}) = 0$ und $g(t_{izu}) = 0$ oder O.B.d.A. $g(t_{iju}) = 1$ und $g(t_{izu}) = 0$.

$\Rightarrow$ Für alle $(i,j),(i,z) \in F$ gilt $f((i,j)) \neq f((i,z))$.

4. Sei $z \in B_4$:
   Also $z = t_{iju}t_{pju}$
   Aus 1 und 2 folgt mit $g(z) = 0$ entweder $g(t_{iju}) = 0$ und $g(t_{pju}) = 0$ oder O.B.d.A. $g(t_{iju}) = 1$ und $g(t_{pju}) = 0$.

$\Rightarrow \forall (i,j),(p,j) \in F$ gilt $f((i,j)) \neq f((p,j))$.

Sei $t : L \rightarrow \{0,1...n\}$ die, durch das partiell gefärbte lateinische Quadrat definierte, Färbungsfunktion. Sei nun $\mathcal{H}$ folgende Funktion:

$$\mathcal{H}((i,j)) = \begin{cases} f((i,j)) & : \quad t(i,j)) = 0 \\ t(i,j)) & : \quad sonst \end{cases}$$

Die Eigenschaften (♠) und (♠♠) sind innerhalb des "eingeschränkten" Definitionsbereich $\mathcal{C}F = \{(i,j) \mid t((i,j)) \neq 0\}$ von t und im Definitonsbereich F von f erfüllt. Für t durch das gegebene lateinische Quadrat, für f durch die Folgerungen der Punkte 3 und 4. Betrachten wir jetzt jeweils zwei Einträge aus den zwei verschiedenen Definitionsbereichen:

Sei $(i,j) \in \mathcal{C}F, (i,z) \in F$.
Angenommen: $t((i,j)) = f((i,z)) = k$.
- $t((i,j)) = k \Rightarrow k \notin FF((i,z))$
- $f((i,z)) = k \Rightarrow g(t_{izk}) = 1 \Rightarrow k \in FF((i,z))$
$\Rightarrow$ Widerspruch

Sei $(i,j) \in \mathcal{C}F, (l,j) \in F$.
Angenommen: $t((i,j)) = f((l,j)) = k$.
- $t((i,j)) = k \Rightarrow k \notin FF((l,j))$
- $f((l,j)) = k \Rightarrow g(t_{ljk}) = 1 \Rightarrow k \in FF((l,j))$
$\Rightarrow$ Widerspruch

$\Rightarrow$ Damit erfüllt $\mathcal{H}$ (♠) und (♠♠), und somit ist f eine gültige Komplettierung.

□

2.4.2 Indirekte Formulierung

In diesem Abschnitt wird das äquivalente Problem, eine Triangelpartition im $defect$ des partiellen lateinischen Quadrat L zu finden, in Polynome übersetzt. Wir konstruieren zuerst den $defect$ G von L.
Danach explorieren wir in Laufzeitkomplexität $O(n^4)$ alle Dreiecke die G enthält.
Sei $D = \{Dreiecke\ \{a,b,c\}\ in\ G\}$ und $T = \{t_{ia} \mid i \in D \wedge a \in E(G) \wedge i = \{a,b,c\}\}$

- $B_1 = \{-(\sum_{i \in D \wedge i = \{a,b,c\}} t_{ia}) + 1 \mid a \in E(G)\}$
- $B_2 = \{t_{ia}t_{ja} \mid i,j \in D \wedge i = \{a,b,c\} \wedge j = \{a,p,q\}\}$
- $B_3 = \{2t_{ia} - t_{ib} - t_{ic} \mid i \in D \wedge i = \{a,b,c\}\}$

Sei $B = B_1 \cup B_2 \cup B_3$.

Proposition. *G hat eine Triangelpartition $\Leftrightarrow$ Es existiert eine Belegungsfunktion $g : K[T] \to K$, so dass für alle $P \in \langle B \rangle$ gilt $g(P) = 0$*

Beweis: $\Rightarrow$
Sei TR eine Triangelpartition von G
Man definiert nun eine Belegungsfunktion:

$$g(p) = \begin{cases} \sum_{i=1}^{n} a_i\ g(M_i) & : \quad p = \sum_{i=1}^{n} a_i M_i,\ M_i\ normiertes\ Monom,\ a_i \in K \\ \prod_{i=1}^{n} g(m_i) & : \quad m_i \in T,\ p = \prod_{i=1}^{n} m_i \\ 1 & : \quad p = t_{ia},\ t_{ia} \in T \\ 0 & : \quad sonst \end{cases}$$

Man prüft leicht nach:

$$\forall\ p,q \in K[T]\ g(p)g(q) = g(pq) \quad (\star)$$

Sei nun $p \in \langle B \rangle$, dann hat p folgende Darstellung:

$$p = \overbrace{\sum_{i=1}^{n_1} x_i b_{1i}}^{p_1} + \overbrace{\sum_{j=1}^{n_2} y_j b_{2j}}^{p_2} + \overbrace{\sum_{l=1}^{n_3} z_l b_{3l}}^{p_3},$$

wobei $b_{1i} \in B_1$, $b_{2j} \in B_2$, $b_{3l} \in B_3$ und $x_i, y_j, z_l \in K[T]$.

1. Sei $z \in B_1$
   Also $z = -(\sum_{i \in D \wedge i = \{a,b,c\}} t_{ia}) + 1$
   Da TR eine gültige Triangelpartition ist, wird jede Kante $a \in E(G)$ nur einem Dreieck zugesprochen. Also $g(t_{sa}) = 1$ nur für exakt ein $s \in D$ $\Rightarrow g(z) = 0$, mit $(\star)$ folgt $g(p_1) = 0$.

2. Sei $z \in B_2$
   Also $z = t_{ia}t_{ja}$
   Da nur exakt ein $s \in D$ mit $g(t_{sa}) = 1$ existiert, sei O.B.d.A. $g(t_{ja}) = 0$.
   Dann folgt $g(t_{ia}t_{ja}) = g(t_{ia})g(t_{ja}) = 0$, mit $(\star)$ folgt $g(p_2) = 0$.

3. Sei $z \in B_3$
   Also $z = 2t_{ia} - t_{ib} - t_{ic}$

   (a) Sei O.B.d.A. $g(t_{ia}) = 1 \Rightarrow g(t_{ib}) = g(t_{ic}) = 1$, da TR Triangelpartition. Damit $g(z) = 0$, also mit $(\star)$ folgt $g(p3) = 0$.

   (b) Sei O.B.d.A. $g(t_{ia}) = 0 \Rightarrow g(t_{ib}) = g(t_{ic}) = 0$, da TR Triangelpartition. Damit $g(z) = 0$, also mit $(\star)$ folgt $g(p3) = 0$.

Also $g(p) = 0$. Also ist g eine Belegungsfunktion mit den gewünschten Eigenschaften.
$\Leftarrow$
Sei $g : K[T] \to K$ eine Belegungsfunktion, so dass für alle $p \in \langle B \rangle$ gilt $g(p) = 0$. Insbesondere für alle $z \in B$ gilt $g(z) = 0$. Wie in $(\dagger)$ und $(\dagger\dagger)$ des letzten Beweises ergibt sich, dass $g : K[T] \to \{0, 1\}$ und es existiert exakt nur ein $k \in D$ und $a \in E(G)$ mit $g(t_{ka}) = 1$.
Definiere nun:

$$TR = \{\{a, b, c\} \mid g(t_{ia}) = 1, g(t_{ib}) = 1, g(t_{ic}) = 1, i \in D\}$$

Wir behaupten nun dies ist eine Triangelpartition von G. Für alle $i = \{a, b, c\}, j = \{d, e, f\} \in TR$ gilt $\{a, b, c\} \cap \{d, e, f\} = \emptyset$. Sonst würde O.B.d.A. gelten $a = d$ und somit $g(t_{ia}) = g(t_{jd}) = 1$.Widerspruch.
Ebenfalls gilt für alle $z = \{a, b, c\} \in TR$, dass z ein Dreieck formt. Angenommen o.B.d.A. c würde nicht zum Dreieck z gehören, dann gilt
$g(2t_{za} - t_{zb} - t_{tc}) = g(2t_{za}) - g(t_{zb}) - g(t_{zc}) = 2 - 1 - 0 = 1$.Widerspruch.

Also ist TR eine gültige Triangelpartition $\square$

2.5 Erzeugung lateinischer Quadrate

Im folgenden Abschnitt befassen wir uns damit, aus einem gegebenen lateinischen Quadrat L_1 weitere zu generieren. Zur Erzeugung von KLQ-Instanzen wollen wir möglichst viele, im Optimalfall sogar alle, lateinischen Quadrate der entsprechenden Größe als Basis wählen können. Denn würde man dann versuchen "Polly Cracker" mittels Lösung der zugrundeliegenden KLQ-Instanz zu brechen, so könnte man keine speziellen Eigenschaften einer Unterklasse von KLQ-Instanzen nutzen. Wir stellen einen Algorithmus vor der auf Mark T. Jacobson und Peter Matthews [10] zurück geht. Er basiert auf einer Markovkette welche jedes lateinische Quadrat L_2 von einem Ausgangsquadrat L_1 mit gleicher Wahrscheinlichkeit erreicht.

2.5.1 Row-pair-graph

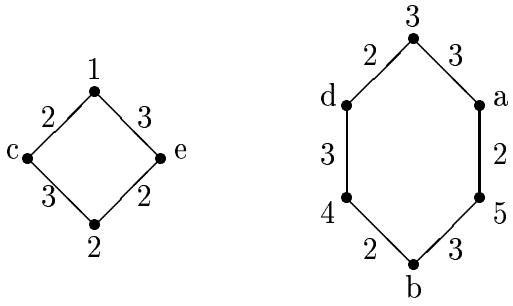
Der *row-pair-graph* zweier Zeilen i, j eines lateinischen Quadrates L wird wie folgt konstruiert:

- $V = \{1_s....n_s\} \cup \{1_v....n_v\}$, wobei i_s für eine Spaltennummer steht und i_v für die Werte in den Zeilen.
- $E = \{\{i_s, j_v\} \mid falls\ Wert\ j\ in\ Spalte\ i\ vorkommt\}$
- $\forall\{i_s, j_v\} \in E\ label(\{i_s, j_v\})$ ="Zeilennummer in der j vorkommt"

Es ist klar, dass $|V| = |\{1_s....n_s\}| + |\{1_v....n_v\}| = n + n = 2n$ und
$\forall g \in V$ gilt $\delta(v) = 2$. Der Graph besteht aus disjunkten Kreisen, welche alle gerade Länge haben und deren *Edgelabel* alternieren. Kein Kreis hat eine Länge kleiner vier.

Ein Beispiel:

$$
\begin{array}{ccccc}
b & a & c & e & d \\
\langle c & e & d & b & a\rangle \\
\langle e & c & a & d & b\rangle \\
d & b & e & a & c \\
a & d & b & c & e
\end{array}
$$



2.5.2 Cycleswap

Wir illustrieren nun mittels *row-pair-graph* eine Operation, welche ein neues lateinisches Quadrat erzeugt. Betrachten wir den Knoten d: Dieser ist inzident zu zwei Kanten mit *label* 2 und 3. Komplementieren wir nun diese beiden *labels*, also 2 wird durch 3 ersetzt und umgekehrt, so bedeutet dies, dass d an den Koordinaten (3,3) und (2,4) zu stehen kommt. Es resultiert ein irreguläres lateinisches Quadrat.

$$\begin{array}{ccccc} b & a & c & e & d \\ \langle c & e & & d,b & a\rangle \\ \langle e & c & d,a & & b\rangle \\ d & b & e & a & c \\ a & d & b & c & e \end{array}$$

Um diesen Missstand zu beseitigen, müssen nur die angrenzenden *labels* vertauscht werden. Dieses Vorgehen lässt sich fortsetzen, bis wieder ein gültiges lateinisches Quadrat entsteht, denn jeder Kreis hat gerade Länge.

$$\begin{array}{ccccc} b & a & c & e & d \\ \langle c & e & a & d & b\rangle \\ \langle e & c & d & b & a\rangle \\ d & b & e & a & c \\ a & d & b & c & e \end{array}$$

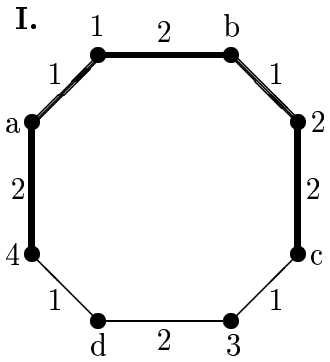
Diese Operation bezeichnen wir als *cycleswap*.

2.5.3 Three-rowed-move

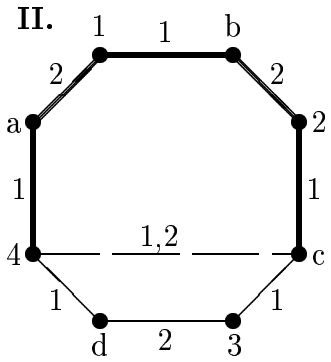
Wir illustrieren nun eine Operation, die 3 Zeilen einbezieht und in ein gültiges lateinisches Quadrat mündet. Wieder nehmen wir den *row-pair-graph* zu Hilfe.

$$\begin{array}{cccc} \langle a & b & c & d\rangle \\ \langle b & c & d & a\rangle \\ d & a & b & c \\ c & d & a & b \end{array}$$

I.



II.

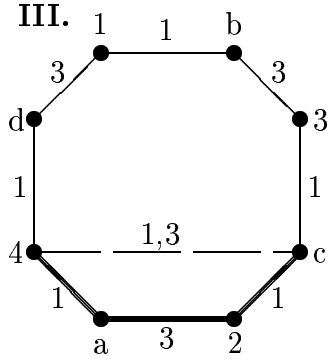


In **I.** haben wir den *row-pair-graph* aufgestellt. In **II.** haben wir die *labels* der 5 herausgehobenen Kanten durch ihr Zeilenkomplement ersetzt. Ebenfalls haben wir eine Linie zwischen c und 4, den beiden Knoten die an unseren herausgehobenen Kantenzug grenzen, eingezeichnet. Diese Linie hat zwei *labels*: 1 und 2. Es resultiert ein ungültiges lateinisches Quadrat.

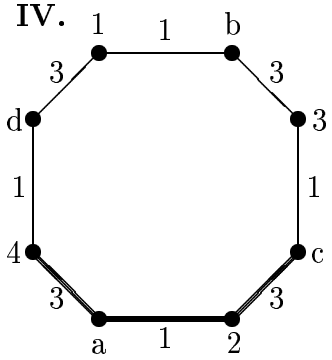
$$\begin{array}{cccc} \langle b & c & c & a, c, d\rangle \\ \langle a & b & d & c\rangle \\ d & a & b & c \\ c & d & a & b \end{array}$$

Es existiert nun eine Zeile, in der es innerhalb dieser keine Kollisionen gibt, in diesem Fall Zeile 2. Allerdings gibt es mit Zeile 3 eine Kollision bezüglich der Spalte. Stellen wir nun den *row-pair-graph* von Zeile 1 und 3 in **III.** auf.

III.



IV.

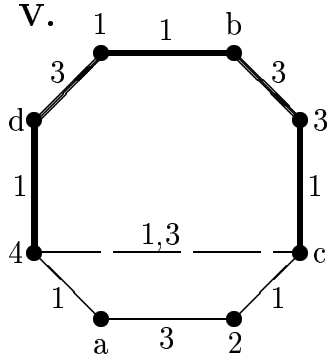


In **IV.** komplementieren wir die *labels* der hervorgehobenen Kanten. Danach können wir Kante $c4$ löschen und ein gültiges lateinisches Quadrat resultiert.

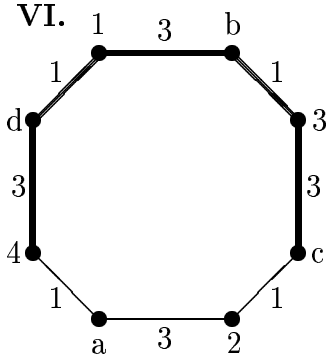
$$\begin{array}{cccc} \langle b & a & c & d\rangle \\ a & b & d & c \\ \langle d & c & b & a\rangle \\ c & d & a & b \end{array}$$

Man hätte natürlich auch den Kantenzug oberhalb der Linie $c4$ hervorheben können. Dann hätte man ein anderes lateinisches Quadrat erreicht.

V.



VI.



$$\begin{array}{cccc} \langle d & c & b & a\rangle \\ a & b & d & c \\ \langle b & a & c & d\rangle \\ c & d & a & b \end{array}$$

2.5.4 Markov-Kette für lateinische Quadrate

Jacobson und Matthews beweisen, dass es für 2 lateinische Quadrate L_1, L_2 eine Sequenz S aus Operationen, bestehend aus *cycleswaps* und *three-rowed-moves*, gibt, so dass L_1 durch S in L_2 überführt werden kann. Ist S die kürzeste solche Sequenz, so ist $4(n-1)^2$ eine obere Grenze.
Sei Y_0 ein beliebiges lateinisches Ausgangsquadrat. Wir konstruieren nun eine Markov-Kette $Y = (Y_0, Y_1, ...)$ indem wir Y_t aus Y_{t-1} $(t > 0)$ wie folgt konstruieren:

1. Wähle ein geordnetes Paar (r, r') gleichverteilt aus den $n(n-1)$ Möglichkeiten, die Y_{n-1} besitzt.
2. Wähle $l \in \{2, 3, ..., n\}$ gleichverteilt.
3. Wähle gleichverteilt einen Knoten v aus dem *row-pair-graph* von (r, r'). Sei 2k die Länge des hierdurch ausgewählten Kreises K.

   - Falls $l \geq k$, führe einen *cycleswap* bezüglich K aus.
   - Falls $l < k$, führe einen *three-rowed-move* aus.

     three-rowed-move:

     (a) Definiere einen hervorgehobenen Kantenzug Ka der Länge $2l-1$, indem wir von v aus über die inzidente r'-$Kante$ weiterschreiten.
     (b) Komplementiere die $labels$ von Ka
     (c) Seien x, y die Endpunkte von Ka. Da Ka ungerade Länge besitzt, ist o.B.d.A. x ein Buchstabe und y eine Zahl. Ergänze r' und r um x an der Spaltenstelle y.
     (d) v kommt in Spalte y in einer Zeile h noch einmal vor. Konstruiere den *r-h-row-pair-graph* U.
     (e) U ist nun ein Kreis mit einer Sehne D. Wähle gleichverteilt einen der beiden Kantenzüge, die D in U definiert, aus. Dieser sei e.
     (f) Komplementiere die $labels$ von e und lösche D.

Jacobs und Matthews zeigen, dass Y eine eindeutige stationäre Gleichverteilung bezüglich der Menge der $n \times n$-lateinischen Quadrate besitzt.

2.5.5 Implementierung

Wir können den *cycleswap* und den *three-rowed-move* direkt auf der Matrix, welche das lateinische Quadrat repräsentiert, durchführen, ohne den *row-pair-graph* mit Hilfe einer geeigneten Datenstruktur aufstellen zu müssen. Der folgende Algorithmus realisiert dies:

1. Wähle zwei Zeilen r und r', so dass $r \neq r'$.
2. Wähle ein $l \in \{2...n\}$.
3. Wähle ein $s \in \{1...n\}$, $sign = s$.
4. Falls $l \geq 0$:
   (a) c="unmodifizierte Spalte in r' welche $sign$ enthält".
   (b) $sign = a_{rc}$, l=$l - 1$.
   (c) vertausche $a_{r'c}$ mit a_{rc}.
   (d) Falls $sign == s$ ENDE, sonst gehe zu 4.
5. sonst, c="unmodifizierte Spalte in r' welche $sign$ enthält", t="Zeile die in Spalte c s enthält".
6. Vertausche $a_{r'c}$ mit a_{tc}.
7. Falls in r eine Zahl z doppelt vorkommt:
   (a) Falls beide Spalten c_1, c_2 in denen z vorkommt unmodifiziert sind:
      - Setzte mit gleicher Wahrscheinlichkeit $c = c_1$ oder $c = c_2$.
   (b) sonst:
      - Sei c_1 unmodifiziert. $c = c_1$.
   (c) Vertausche a_{rc} mit a_{tc}.
   (d) gehe zu 7.
8. sonst ENDE :

Ein Beispiel mit $r = 1, r' = 2, l = 3$ und $s = c$

$$\begin{array}{ccccc} r & a & b & c & d \\ r' & b & c & d & a \\ t & d & a & b & c \\ & c & d & a & b \end{array} \rightarrow \begin{array}{cccc} a & \overline{c} & c & d \\ b & \underline{b} & d & a \\ d & a & b & c \\ c & d & a & b \end{array} \rightarrow \begin{array}{cccc} \overline{b} & c & c & d \\ \underline{a} & b & d & a \\ d & a & b & c \\ c & d & a & b \end{array} \rightarrow \begin{array}{cccc} b & c & c & d \\ a & b & d & \overline{c} \\ d & a & b & \underline{a} \\ c & d & a & b \end{array}$$

1.Möglichkeit

$$\begin{array}{cccc} b & \overline{a} & c & d \\ a & b & d & c \\ d & \underline{c} & b & a \\ c & d & a & b \end{array}$$

2.Möglichkeit

$$\begin{array}{cccc} b & c & \overline{b} & d \\ a & b & d & c \\ d & a & \underline{c} & a \\ c & d & a & b \end{array} \rightarrow \begin{array}{cccc} \overline{d} & c & b & d \\ a & b & d & c \\ \underline{b} & a & c & a \\ c & d & a & b \end{array} \rightarrow \begin{array}{cccc} d & c & b & \overline{a} \\ a & b & d & c \\ b & a & c & \underline{d} \\ c & d & a & b \end{array}$$

2.6 Algorithmus für KLQ

Um nun eine schwere Instanz von KLQ zu finden versuchen wir im folgenden die Komplexität einer solchen zu messen. Hierzu verfolgen wir eine empirische Herangehensweise indem wir die Komplexität in der Anzahl der Rekursionen des folgenden Algorithmus $SKLQ$ messen.

Sei $L = \{(i, j) \mid 0 \leq i, j \leq n\}$ ein $n \times n$-lateinisches Quadrat.

$SKLQ$:

1. Für alle $(i, j) \in F$ berechne $FZF((i))$ und $FSF((j))$
2. Für alle $(i, j) \in F$ berechne $FF((i, j)) = FZF((i)) \cap FSF((j))$
3. Sei HL die Liste die F enthält. Sortiere dann HL aufsteigend bezüglich $|FF((i, j))|$
4. $solve(HL)$

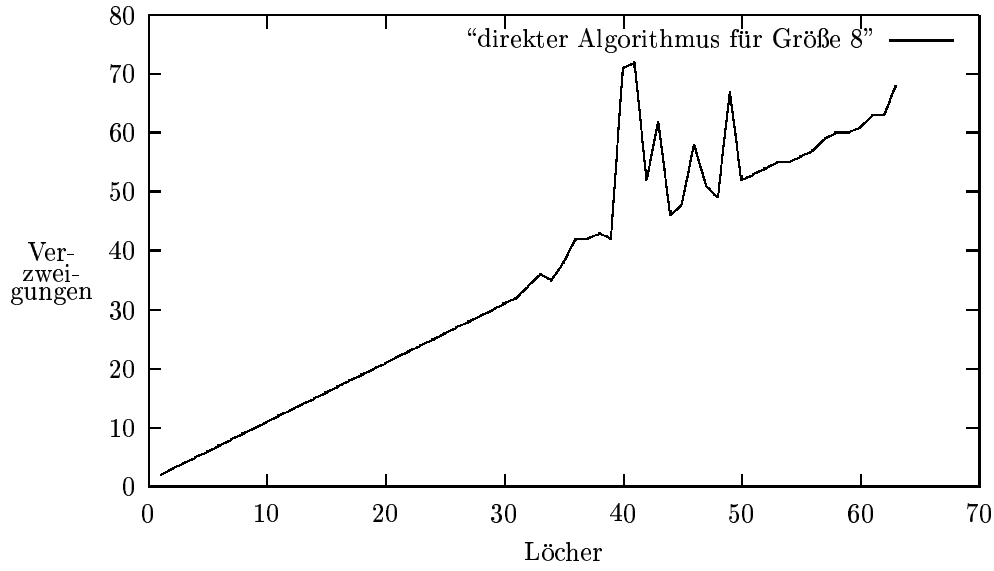
$solve(ListeHL)$:

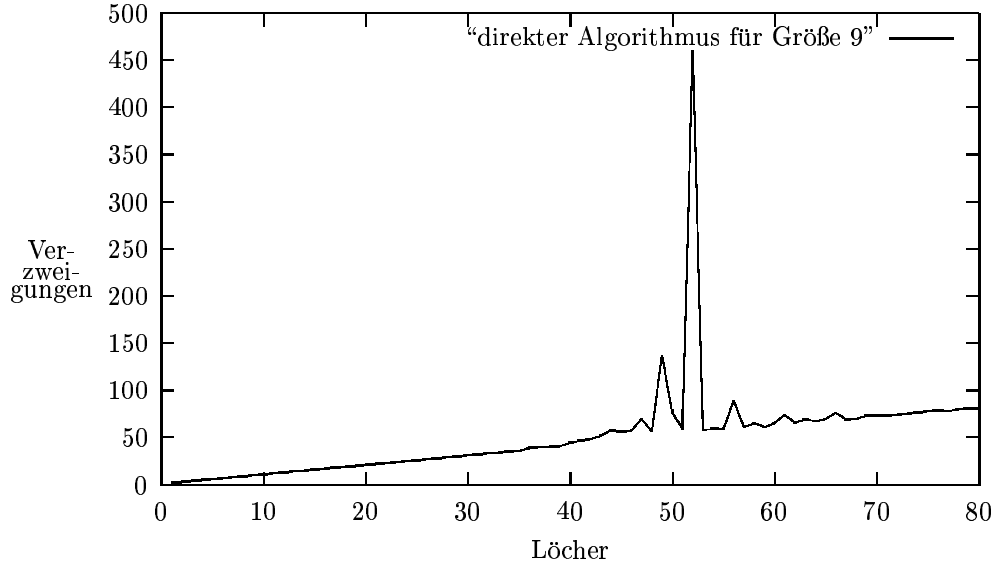
1. Ist HL leer, return true
2. Wähle $act = HL.first$. Ist $FF(act) = \emptyset$ return false.
3. Sei $act = (i, j)$.Wähle $col = FF(act).first$ und setze $L|_{ij} = col$
4. Für alle $z \in HL$, z in der gleichen Spalte oder Zeile wie act, setze $F(z) = F(z) \setminus \{col\}$
5. Sei $NHL = HL \setminus HL.first$
6. Sortiere dann NHL aufsteigend bezüglich $|FF(z)|$
7. Falls $solve(NHL) = true$ return true, sonst $L|_{ij} = 0$
8. gehe zu 2.

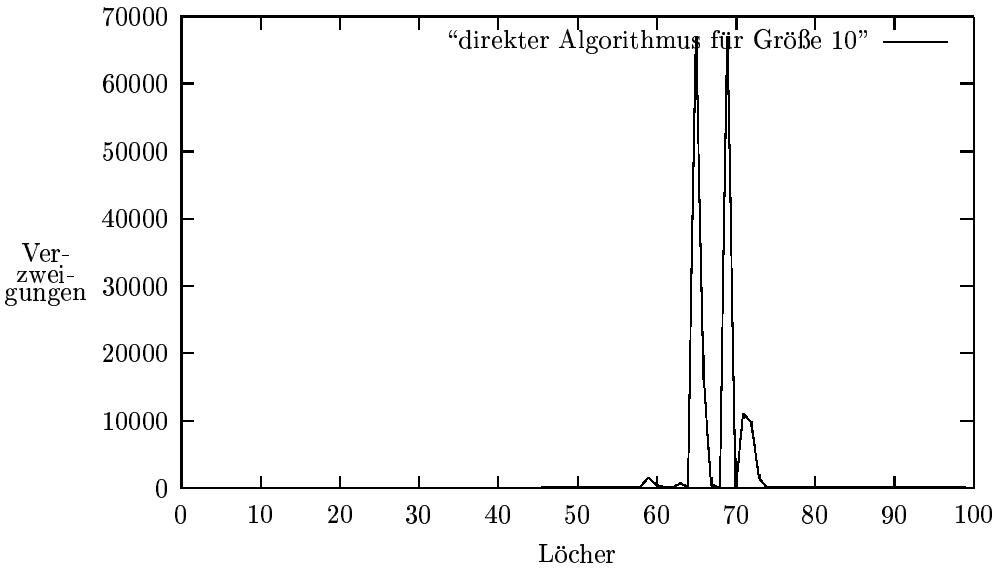
2.7 Empirisches Verhalten

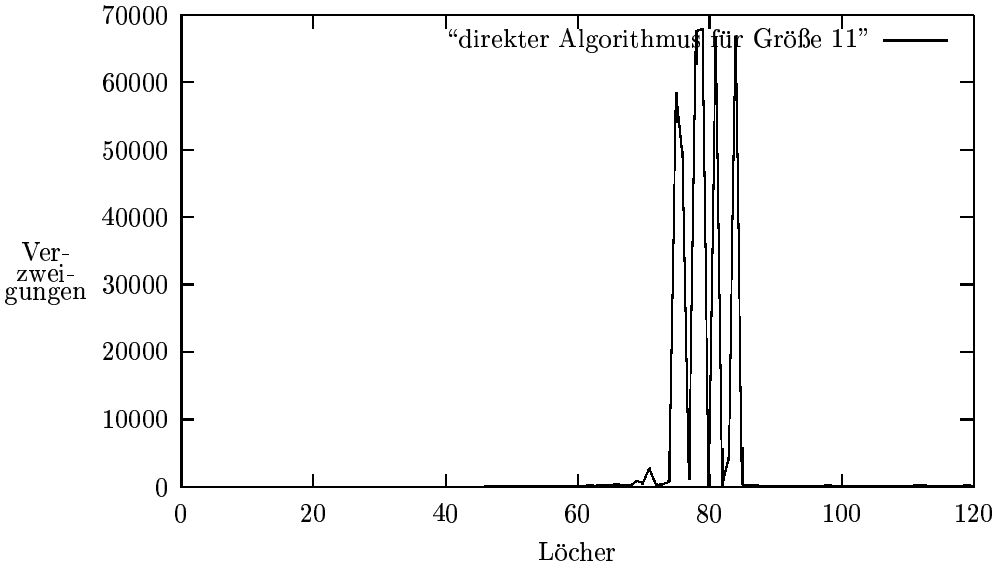
In diesem Abschnitt nähern wir uns KLQ durch zwei empirische Aufwandsabschätzungen. Es werden der direkte Algorithmus $SKLQ$ und ein gewöhnlicher SAT-Beweiser verwendet. Die angewandten Metriken waren beides mal die Verzweigungen im Entscheidungsbaum. Dies ist natürlich nur ein Maß zur Aufwandsabschätzung. Ein weiteres wäre z.B. ein direkter Leistungsvergleich auf dem gleichen Rechner, auf den aufgrund von Zeitgründen verzichtet wurde. Das gewählte Maß kann in gewissen Fällen sogar in die Irre leiten. Dann beispielsweise, wenn die KLQ-Instanzen einer Horn-Formel entspricht. Der SAT-Beweiser erkennt diese als solche und kann sie in polynomieller Zeit mit einer Verzweigung lösen. $SKLQ$ besitzt diese Optimierung nicht. Dennoch erscheint die Wahl des Maßes vernünftig, da gerade bei den interessanten, schwierigen Instanzen solche Optimierungen nur sehr beschränkt möglich sind. Bezüglich jeder hier betrachteten Größe eines lateinischen Quadrates wurden pro Lochanzahl jeweils 30 Instanzen erzeugt. Danach wurde der Aufwand dieser 30 Instanzen gemittelt.

2.7.1 Direkter Algorithmus

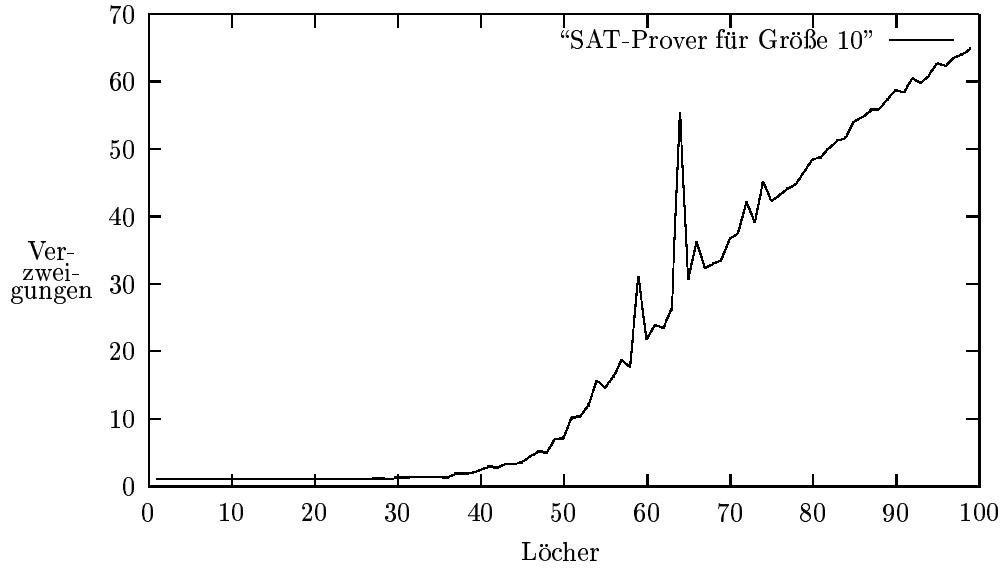


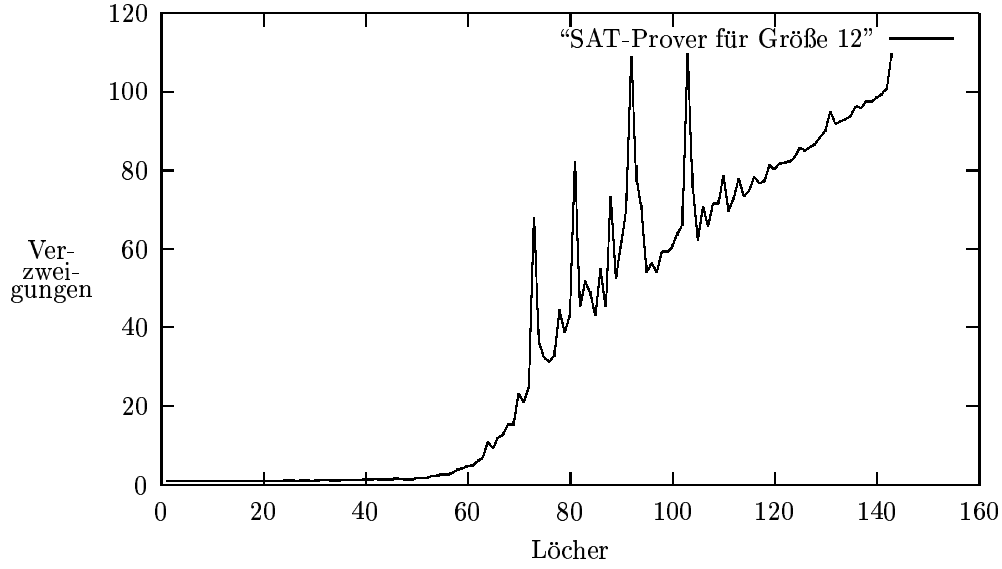


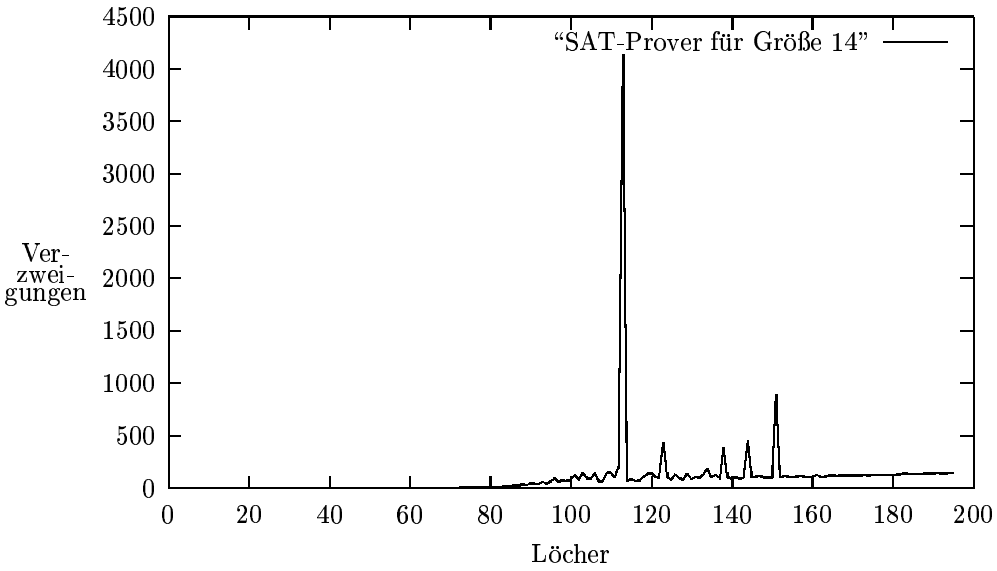

"direkter Algorithmus für Größe 10"
Ver-
zwei-
gungen
70000
60000
50000
40000
30000
20000
10000
0
0
10
20
30
40
50
60
70
80
90
100
Löcher

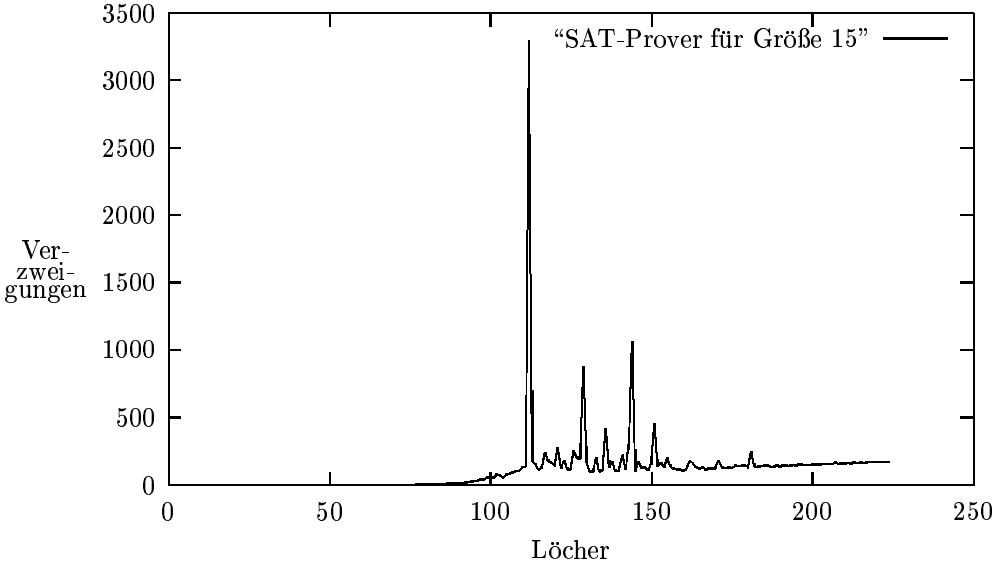

"direkter Algorithmus für Größe 11"
Ver-
zwei-
gungen
70000
60000
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
120
Löcher

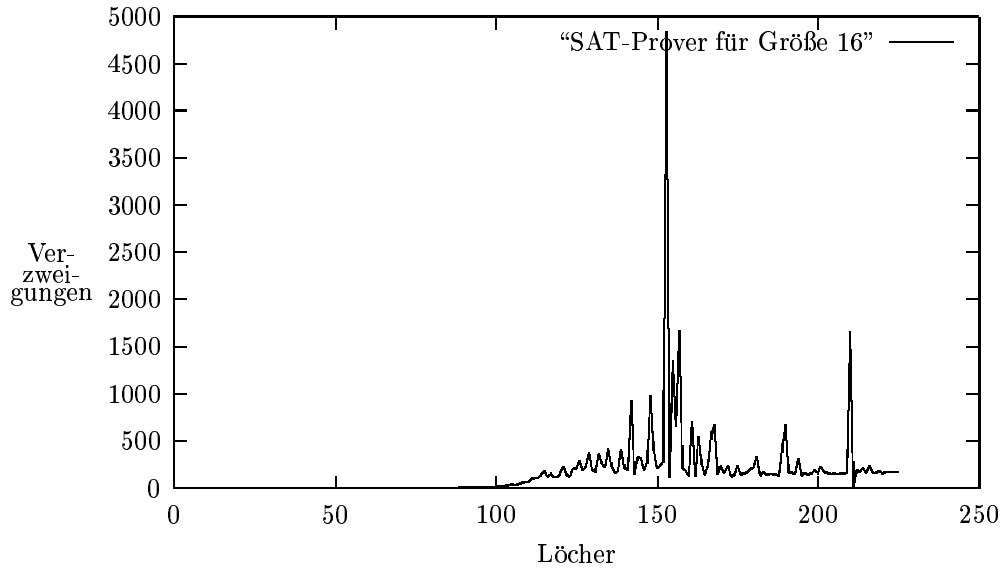
2.7.2 SAT-Beweiser






"SAT-Prover für Größe 14"
Ver-
zwei-
gungen
4500
4000
3500
3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100
120
140
160
180
200
Löcher


"SAT-Prover für Größe 15"
Ver-
zwei-
gungen
3500
3000
2500
2000
1500
1000
500
0
0
50
100
150
200
250
Löcher



2.7.3 Resultat

Bezüglich beider Lösungsansätze bilden sich typische leicht-hart-leicht Aufwandskurven heraus. Die schwierigen Instanzen sind beide Male in einem schwierigen Bereich bei circa $\frac{2}{3}n^2$ Löchern zu finden. Der SAT-Ansatz schneidet bezüglich Aufwand besser ab, doch auch hier ist ein mehr als linearer Anstieg im kritischen Bereich zu beobachten.
Folgende Vermutungen ergeben sich hieraus:

- Für Instanzen mit vielen Löchern findet man aufgrund der wenigen Restriktionen schnell eine Lösung.

- Für Instanzen mit wenigen Löchern werden falsche Entscheidungen aufgrund der vielen Restriktionen schnell entdeckt und korrigiert.

- Für Instanzen aus dem kritischen Bereich werden falsche Entscheidungen, welche sehr früh im Rekursionsbaum getroffen werden, erst sehr viel später entdeckt. Dies zieht den vergleichsweise großen Lösungsaufwand nach sich.

2.8 Resümee

Mit Hilfe von "Polly Cracker" ist man nun in der Lage, für geeignete Instanzen von KLQ mittels deren Polynomcodierung und deren Lösung, ein sicheres Public-Key-Kryptosystem zu konstruieren. "Geeignet" bedeute hier schwer lösbar bezüglich aktueller Algorithmen.
Die Markovkette von Jacobson und Matthews versetzt uns in die Lage, beliebige lateinische Quadrate zu konstruieren. Dies ist vorteilhaft, da man bei einem Angriff auf "Polly Cracker" sich nicht auf eine bestimmte Klasse von KLQ beschränken kann. Um aus einem solchen lateinischen Quadrat eine schwierige Instanz zu erhalten, sollte man die Anzahl der zu "stanzenden" Löcher aus dem kritischen Bereich wählen. Dies ist natürlich keine exakte Lokalisierung einer schwierigen Instanz. Würde dies für einen beliebigen Algorithmus gelingen, so wäre dies auch ein Beweis für $NP \neq P$.
Das bestimmen der Nullstellen der Codierungspolynome, also auch das Entschlüsseln der Nachricht, ist damit gleichwertig die Lösung des entsprechenden KLQ-Problems zu lösen. Wie man exemplarisch an den beiden Lösungsansätzen feststellen konnte, ist der Rechenaufwand beider bei der gleichen Anzahl von Löchern besonders gross.

Anhang A

Appendix: Implementierung der Messalgorithmen

A.1 Klassen

Damit die Messungen des letzten Paragraphen vorgenommen werden konnten, mussten die in diesem Papier vorgestellten Verfahren implementiert werden. Dazu gehört eine Matrixrepräsentation der lateinischen Quadrate, die Markovkette zu deren Erzeugung, das "Stanzen" der Löcher, der Algorithmus $SKLQ$ und das Codieren von KLQ in SAT-Formeln. Folgende $C++$-Klassen wurden zu diesem Zweck geschrieben:

Klasse	implementierte Funktionen
Latinsquare	Markovkette, "Löcherstanzen", $SKLQ$
Satgen	Codieren von KLQ in SAT-Formeln
matrix	Matrixrepräsentation der lateinischen Quadrate
Randint	Erzeugung von Zufallswerten
Latinsquare::holevalues	Datenstruktur welche eine Liste der möglichen Werte für ein Loch (i,j) hält

Es folgt eine Schnittstellenbeschreibung der öffentlichen Funktionen der Klassen, welche sich am höchsten in der Klassenhierarchie befinden.

matrix

matrix(int nn): Der Konstruktor besitzt als Parameter die Dimension der Matrix.

void print(): Ausgabe der Matrix auf dem Terminal.
int& at(int row,int col): Liefert die Werte an der Stelle (row,col).
int search_col(int z,int s): Sucht Element s in Spalte z und gibt Position zurück.

Satgen

Satgen(const list⟨Latinsquare::holevalues⟩, int,string): Konstruktor nimmt eine Liste von holevalues, dann Angabe der Dimension der Matrix und einen Pfad für die SAT-Formel-Dateien.
void Satgen::newFile(int z): Öffnet die zwei Dateien Satlatin$z$.cnf und Satlatin$z$.des. Erklärung des Formats weiter unten.
void create(): Erzeugt die SAT-Formeln in Satlatin$z$.cnf und eine Übersetzung der belegten Variablen in eine Lösung für KLQ in Satlatin$z$.des.
void addholes(const list⟨Latinsquare::holevalues⟩): Eine neue Liste von holevalues als Basis setzen.

Latinsquare

Latinsquare(matrix*,int,bool, bool,string): Konstruktor nimmt als Parameter: Matrixzeiger, maximale Rekursionstiefe für $SKLQ$ (-1 für keine Beschränkung), $SKLQ$ starten, SAT-Formeln generieren, Pfad für SAT-Formel-Dateien
void step(int anz): Führt anz Markovkettenschritte auf der Matrix durch.
void punch_holes(int l): "Stanzt" l Löcher gleichverteilt in die Matrix.
int solve(): Je nachdem wie die Parameter im Konstruktor gesetzt wurden, wird jetzt die KLQ-Instanz, welche die Matrix darstellt, gelöst, die entsprechenden SAT-Formeln generiert oder beides vollzogen. Der Rückgabewert gibt die durchlaufenen Verzweigungen im Rekursionsbaum an.
void set_matrix(matrix* j): Neue Matrix auf der operiert werden soll wird übergeben.

Das folgende Beispielprogramm soll illustrieren wie die Klassen sinnvoll eingesetzt werden können:

```
#include <string>
#include "random.h"
#include "matrix.h"
#include "Latinsquare.h"
```

```
int main(int argc,char *argv[])
{
  //Parameterbergabe ueberprfen
  if(argc < 7)
    {
      cout << "usage:a.out dim times steps upperlimit latsolve satoutput
satpath" << endl;
      exit(1);
    }

  int mil=0;

  int milsize = atoi(argv[4]);

  string buffer;
  int dim,times,steps;
  steps = atoi(argv[3]);
  dim = atoi(argv[1]);
  times = atoi(argv[2]);

  //Matrixinstanz von der Halde holen;
  matrix* m = new matrix(dim);

  double aver=0;

  //Matrix m ausgeben
  m->print();
  //Eine Latinsquareinstanz die auf der Matrix m operiert wird erzeugt
  Latinsquare latin(m,milsize,bool(atoi(argv[5])),bool(atoi(argv[6])),
string(argv[7]));
  //Markovkette steps viele Schritte auf m starten
  latin.step(steps);
  //Eine weitere Matrixinstanz erzeugen
  matrix copy(dim);

  for(int i=1;i< dim*dim;i++)
    {
      cout << i <<" holes" << endl;
```

```
      for(int z=0;z<times;z++)
        {//Original Matrix m speichern
          copy = *m;
          //Loecher in die Matrix m stanzen
          latin.punch_holes(i);
          //KLQ loesen, SAT-Formeln generieren oder beides
          int times2= latin.solve();

          //ist der Rueckgabewert -1 so wurde die maximal Rekursionstiefe
          // erreicht
          if(times2 == -1)
            {
              mil++;
              times2=milsize;
            }
          aver+=times2;
          //Matrix m restaurieren
          *m = copy;
        }
      //Ergebnisse mitteln und ausgeben
      cout <<endl<< "average " << (aver / times) << " aver / holes " <<
((aver /times )/i)<< endl;
      aver = 0;
    }
  cout << mil <<" times over " << milsize << endl;
  return 0;
}
```

Ein Beispielaufruf diese Programms: ./a.out 7 10 1000 10000 0 1 “satout”

Die Dimension ist 7, 10 Instanzen werden pro Lochanzahl gemittelt, auf der Matrix werden 1000 Markovkettenschritte ausgeführt, die maximale Rekursionstiefe ist 10000, die KLQ-Instanzen werden nicht gelöst und es werden SAT-Formel-Dateien in “./satout” erzeugt.

A.2 Format der SAT-Formel-Dateien

Es werden zwei Arten von SAT-Formel-Dateien erzeugt. Sie unterscheiden sich jeweils in ihren Endungen ".cnf" und ".des".

Dateien mit ".cnf"-Endung enthalten die SAT-Codierung der entsprechender KLQ-Instanz. In der ersten Zeile folgt dem Ausdruck "p cnf", er zeigt an, dass die SAT-Formel in konjunktiver Normalform vorliegt, die Variablen- und die Klauselanzahl. In den weiteren Zeilen trifft man auf die Codierung der Klauseln.
Die Variablen werden über Nummern angesprochen. Um eine Variable zu negieren wird ein Minuszeichen vorangestellt. Eine Klausel ist dann eine Zeile, bestehend aus den Variablennummern, gegebenenfalls mit Minus als Präfix, und 0 als Abschlusszeichen. Die Klausel $\{x_1 \vee x_4 \vee \overline{x_7}\}$ hätte dann die Entsprechung **1 4 -7 0**.

Die Dateien mit ".des"-Endung liefern eine Übersetzung von der Variablenmenge der SAT-Codierung in eine gültige Komplettierung der korrespondierenden KLQ-Instanz.
Jede Zeile besitzt das beispielhafte Format **6:5:3=20**. Dies entspricht der Tatsache, dass die Variable x_{20} festlegt ob der Eintrag $a_{6,5}$ mit der Farbe 3 gefärbt wird. Man kann mit Hilfe dieser Übersetzung aus der Variablenbelegung der SAT-Codierung eine gültige Komplettierung erstellen.

Literaturverzeichnis

[1] Albrecht Beutelspacher *Lineare Algebra*,Vieweg Verlag 2000

[2] M.R.Fellows, N.Koblitz *Combinatorial cryptosystmes for children (and adults)*, 1994 Congressus Numerantium 99,9-41

[3] M.R.Fellows, N.Koblitz *Combinatorial cryptosystems galore!* Contemporary Math. 168, 51-61

[4] Uwe Schöning *Theoretische Informatik - kurz gefasst* Spektrum Verlag 1997

[5] Dennis Hofheinz *Angriffe auf das Public-Key-System Polly Cracker* Studienarbeit, Universität Karlsruhe, Fakultät für Informatik, Institut für Algorithmen und kognitive Systeme

[6] N. Koblitz *Algebraic Aspects of Cryptography* Springerverlag 1998

[7] Charles J.Colburn *The complexity of completimg partial latin squares* Discrete Applied Mathematics 8 (1984) 25-30

[8] Ian Holyer *The NP-completness of some edge-partition problems* SIAM J. Comput. Vol. 10, No.4, November 1981

[9] Mark T. Jacobson, Peter Matthews *Generating uniformly distributed random latin squares* Journal of Combinatorial Designs, vol.4., no. 6, (1996) 405-437

[10] Victor Batyrev *Algebra I*

[11] Martin Aigner *Graphentheorie-Skript* Vorlesung an der Freien Universität Berlin SS91

[12] Martin Aigner *Diskrete Mathematik*