1 Der Uncovering-by-bases-Algorithmus

1.1 Definition (Der Algorithmus)
Sei G eine Gruppe, $\mathcal{U}$ ein Uncovering durch Basen und $w = w_1 \dots w_n$ ein empfangenes Wort. Dann funktioniert der Uncovering-by-bases-Algorithmus wie folgt:
Man nehme eine Basis $B_1 \in \mathcal{U}$ und betrachte die Stellen i in w, die durch die Einträge in B_1 gegeben sind. Sind die Symbole an diesen Stellen in w alle verschieden, so suche in der Tabelle der Gruppenelemente nach einem $g \in G$, das mit w in den betrachteten Positionen übereinstimmt. Wenn ein solches g exisiert, überprüfe die Hammingdistanz zwischen g und dem empfangenen Wort w. Ist diese kleiner oder gleich der Korrekturkapazität r, stoppt der Algorithmus. g war das gesendete Wort.
Andernfalls nimm die nächste Basis $B_2 \in \mathcal{U}$ und wiederhole den Vorgang für B_2.
Iteriere.

1.2 Bemerkung (Unterschiede zum Uncovering-Algorithmus)
Der Uncovering-by-bases-Algorithmus unterscheidet sich nur in zwei kleinen Details vom Uncovering-Algorithmus.

- Die Basen B eines Uncoverings durch Basen können unterschiedliche Größe haben. Die Zeilen eines $(n, k, r)-$Uncoverings hatten hingegen immer feste Länge k.
- Weil die Existenz eines Gruppenelements, das mit dem empfangenen Wort in den durch eine Basis vorgegebenen Stellen übereinstimmt, nicht mehr garantiert ist, beinhaltet der Uncovering-by-bases-Algorithmus zusätzlich einen Test für die Nicht- Existenz.

2 Der Rekonstruktions-Algorithmus

Um die Komplexität des Uncovering-by-bases-Algorithmus verstehen und berechnen zu können, bedarf es eines Hilfs- Algorithmus, der einem genaueren Aufschluss über die Abläufe während der Rekonstruktion eines Gruppenelements gibt.
Um den sogenannten „Rekonstruktions-Algorithmus“ beschreiben zu können, braucht man folgende Voraussetzungen:
Angenommen, man hat eine Gruppe G, die auf einer Menge Ω mit $|\Omega| = n$ operiert. Sei weiter $B = (x_1, \dots, x_b)$ eine Basis von G bzgl. Ω und
$G_{x_1,\dots,x_i} = \{g \in G \,|\forall x_i \in B : x_i^g = x_i\}$ der punktweise Stabilisator von $(x_1, \dots, x_i)$ in G.

2.1 Definition (Stabilisatorkette)
Die *Stabilisatorkette* zu einer Basis $B = (x_1, \dots, x_b)$ ist die Kette der Untergruppen

$$G \geq G_{x_1} \geq G_{x_1,x_2} \geq \dots \geq G_{x_1,\dots,x_{b-1}} \geq G_{x_1,\dots,x_b} = \{id\}\,.$$

2.2 Definition (Starke Erzeugermenge, [Sims, 1970/71])
Es sei $B = (x_1, \ldots, x_b)$ eine Basis der Gruppe G. Eine Menge S von Erzeugern von G heißt *starke Erzeugermenge* bzgl. der Basis B, wenn für $1 \leq i \leq b$ gilt:

(i) $G = \langle S \rangle$

(ii) $G_{x_1,\ldots,x_i} = \langle S \cap G_{x_1,\ldots,x_i} \rangle$

S enthält also Erzeuger für jede Untergruppe der Stabilisatorkette. Deshalb kann man S auch schreiben als $S = S_1 \cup \ldots \cup S_b$, wobei S_i die Menge der Nebenklassenrepräsentanten von $G_{x_1,\ldots,x_{i-1}}$ in $G_{x_1,\ldots,x_i}$ ist.

2.3 Satz [Sims, 1970/71]
Sei G eine Gruppe, $B = (x_1, \ldots, x_b)$ eine Basis von G und $S = S_1 \cup \ldots \cup S_b$ ein Menge von starken Erzeugern bezüglich B.
Dann kann jedes Gruppenelement $g \in G$ eindeutig als Produkt $s_b s_{b-1} \cdots s_1$ geschrieben werden, wobei $s_i \in S_i$.

Beweis: - siehe Cameron: *Permutation Groups*, Section 1.13

Mit dieser Vorbereitung lässt sich nun der Rekonstruktions-Algorithmus formulieren.

2.4 Definition (Der Rekonstruktions-Algorithmus, [Bailey])
Sei G eine Gruppe, die als fehlerkorrigierender Code benutzt wird. Sei weiter $B = (x_1, \ldots, x_b)$ eine Basis von G und S eine zugehörige Menge starker Erzeuger. Man empfange das Wort w, das die Symbole $(y_1, \ldots, y_b)$ an den Stellen hat, die durch die Basis B gegeben sind.
Der Algorithmus soll folgende Frage beantworten:
Gibt es $\forall i$ ein $g \in G$, so dass $x_i^g = y_i$ und wenn ja, wie sieht dieses g aus?
Die Rekonstruktion funktioniert folgendermaßen:
Betrachte im ersten Schritt die Menge $\{x_1^s \,|\, s \in S_1\}$. Das ist die Menge aller Elemente, in die x_1 durch Anwendung von $s \in S_1$ überführt werden kann. Nun überprüft man, ob y_1 in dieser Menge enthalten ist.
Ist $y_1 \notin \{x_1^s \,|\, s \in S_1\}$, so kann es nach Satz 2.3 kein $g \in G$ mit $x_i^g = y_i$ geben und der Algorithmus stoppt.
Wenn y_1 in der obigen Menge enthalten ist, dann sei s_1 das Element aus S_1, das x_1 auf y_1 abbildet. Ersetze nun $(y_1, \ldots, y_b)$ durch $(y_1^{s_1^{-1}}, \ldots, y_b^{s_1^{-1}})$ und iteriere wie folgt:
Prüfe zum Zeitpunkt i, ob es ein $s_i \in S_i$ gibt, so dass $x_i^{s_i} = y_i^{s_1^{-1} \cdots s_{i-1}^{-1}}$ ist. Wenn es ein solches s_i nicht gibt, kann es nach Satz 2.3 kein $g \in G$ mit der gewünschten Eigenschaft geben und der Algorithmus stoppt. Existiert ein solches Element s_i, so ersetze $(y_1^{s_1^{-1} \cdots s_{i-1}^{-1}}, \ldots, y_b^{s_1^{-1} \cdots s_{i-1}^{-1}})$ durch $(y_1^{s_1^{-1} \cdots s_i^{-1}}, \ldots, y_b^{s_1^{-1} \cdots s_i^{-1}})$ und fahre fort.
Ist man bei $i = b$ angekommen und existiert ein $s_b \in S_b$ wie oben beschrieben, so ist

man fertig und das Element $s_b \cdot ... \cdot s_1$ ist das gesuchte Gruppenelement g, denn $\forall i$ gilt:

$$\begin{aligned} x_i^g &= x_i^{s_b \cdots s_1} \\ &= x_i^{s_i \cdots s_1} \qquad \text{(weil } s_b, s_{b-1}, \ldots, s_{i+1} \text{ alle in } G_{x_1,\ldots,x_i} \text{ liegen)} \\ &= (y_i^{s_1^{-1} \cdots s_{i-1}^{-1}})^{s_{i-1} \cdots s_1} \\ &= y_i \end{aligned}$$

Jetzt kann man also nachvollziehen, wie algorithmisch ein Gruppenelement zurückgewonnen werden kann. Deshalb kann man sich nun der Frage nach der Komplexität der eingeführten Algorithmen widmen.

3 Die Komplexität der Algorithmen

Neben der Berechnung der reinen Komplexität (Anzahl der Elementaroperationen) soll auch der benötigte Speicherplatz für die Durchführung der Algorithmen bestimmt werden.
Aus diesem Grund trifft man zunächst folgende Vereinbarung:

- Das Bild eines Punktes unter einer Permutation zu finden kostet *eine* Elementaroperation
- Die Komposition zweier Permutationen der Länge n kostet n Elementaroperationen
- Das Abspeichern *eines* Symbols kostet *eine* Speichereinheit

Mit dieser Vereinbarung soll nun zunächst die Komplexität und danach der Speicherplatzbedarf des Rekonstruktions-Algorithmus berechnet werden.

3.1 Lemma (Komplexität des Rekonstruktions-Algorithmus)
Die Komplexität des Rekonstruktions-Algorithmus ist $O(bn)$, wobei b die Anzahl der Stellen einer Basis und n die Mächtigkeit von Ω ist.

Beweis: Im i-ten Schritt des Algorithmus durchsucht man die Menge S_i um zu sehen, ob das Element s_i mit $x_i^{s_i} = y_i^{s_1^{-1} \cdots s_{i-1}^{-1}}$ darin vorkommt. Da S_i die Menge aller Nebenklassenrepräsentanten von $G_{x_1,\ldots,x_{i-1}}$ in $G_{x_1,\ldots,x_i}$ ist, gilt $|S_i| = \left|G_{x_1,\ldots,x_{i-1}} : G_{x_1,\ldots,x_i}\right| \leq n$. Es wird also höchstens n mal das Bild von x_i unter s_i bestimmt, das anschließend noch mit $y_i^{s_1^{-1} \cdots s_{i-1}^{-1}}$ verglichen wird. So ergeben sich höchstens $2n$ Elementaroperationen für diese Aktion. Hat man das Element in S_i gefunden, so muss man insgesamt b Punkte durch deren Bilder unter s_i^{-1} ersetzen. Das sind nach obiger Vereinbarung gerade b Operationen. Man macht also in jedem Schritt des Algorithmus höchstens $(2n+b)$ Operationen. Insgesamt gibt es natürlich höchstens b dieser Schritte, also höchstens $b(2n+b)$ Operationen.

Waren alle b Schritte erfolgreich, so müssen $b-1$ Permutationen hintereinander ausgeführt werden, um das Gruppenelement zu rekonstruieren. Nach der Vereinbarung sind das $(b-1)n$ Operationen. Insgesamt erhält man also für die maximale Anzahl an Operationen $b(2n+b)+(b-1)n = 3bn+b^2-n \in O(bn)$, da $b < n$. □

3.2 Lemma (Speicherplatzbedarf des Rekonstruktions-Algorithmus)
Der benötigte Speicherplatz des Rekonstruktions-Algorithmus ist $O(bn^2)$ und der Platz (Zwischenspeicher), der gebraucht wird um den Algorithmus durchzuführen, ist $O(n)$.

Beweis: Eine Suchtabelle enthält eine starke Erzeugermenge und die zugehörige Menge der inversen Elemente. Eine solche starke Erzeugermenge enthält b Mengen S_i, die ihrerseits jeweils höchstens n Elemente enthalten. Jedes dieser Elemente ist n Symbole lang, so dass für die starke Erzeugermenge höchstens bn^2 Speichereinheiten benötigt werden. Da außerdem das Inverse zu jedem Element gespeichert werden muss, benötigt man insgesamt also $2bn^2 \in O(bn^2)$ Speichereinheiten.
Während der Durchführung des Algorithmus muss in jedem Schritt i die Position des Elements s_i in der Suchtabelle gespeichert werden, was eine Speichereinheit kostet. Bei insgesamt b Schritten braucht es also auch b Speichereinheiten. Zudem werden für die Berechung der $s_b \cdots s_1$ zur Rückgewinnung des Gruppenelementes weitere n Speichereinheiten benötigt, so dass letztlich $b+n \in O(n)$ Speichereinheiten zur Verfügung stehen müssen. □

Ruft man sich nun in Erinnerung, dass der eigentliche Uncovering-by-bases-Algorithmus so funktioniert, dass er sich durch die Menge von Basen im Uncovering arbeitet und dabei jedes Mal den Rekonstruktions-Algorithmus wiederholt, bis das Gruppenelement wieder richtig zurückgewonnen wurde, dann geben die folgenden Resultate Aufschluss über die Komplexität des Gesamtalgorithmus.

3.3 Satz (Komplexität des Uncovering-by-bases-Algorithmus)
Sei $\mathcal{U}$ ein Uncovering durch Basen. Dann ist die Komplexität des Uncovering-by-bases-Algorithmus $|\mathcal{U}|\, O(bn)$.

Beweis: Man verwendet für jede Basis aus dem Uncovering den Rekonstruktions- Algorithmus, der nach Lemma 3.1 höchstens $b(2n+b)+(b-1)n$ Elementaroperationen benötigt. Anschließend prüft man, ob die Hammingdistanz zwischen der rekonstruierten Permutation und dem empfangenen Wort kleiner oder gleich der Korrekturkapazität des Codes ist. Für diese Überprüfung benötigt man n Vergleiche. Diesen Vorgang muss man unter Umständen für alle Basen aus dem Uncovering $\mathcal{U}$ wiederholen, so dass man insgesamt auf folgenden Aufwand kommt:

$$|\mathcal{U}|\,(\underbrace{b(2n+b)+(b-1)n}_{\text{Lemma 3.1}}+n) = |\mathcal{U}|\,(3bn+b^2) \in |\mathcal{U}|\,O(bn)$$

□

3.4 Satz (Speicherplatzbedarf des Uncovering-by-bases-Algorithmus)
Der benötigte Speicherplatz des Uncovering-by-bases-Algorithmus ist $|\mathcal{U}|\, O(bn^2)$ *und der benötigte Zwischenspeicher ist* $O(n)$.

Beweis: Zunächst sei bemerkt, dass jede Basis aus einem betrachteten Uncovering $\mathcal{U}$ ihre eigene starke Erzeugermenge besitzt. Nach Lemma 3.2 benötigt man deshalb $2\,|\mathcal{U}|\, bn^2 \in |\mathcal{U}|\, O(bn^2)$ Speichereinheiten für die zugehörigen Suchtabellen.
Um den Algorithmus durchzuführen benötigt man zunächst nur die $b+n$ Speichereinheiten aus Lemma 3.2. Die darin enthaltenen n Einheiten werden außerdem zum Zwischenspeichern des potentiellen Kandidaten für das urprüngliche Gruppenelement benutzt und nach dem Vergleich mit dem empfangenen Wort wieder zur Verfügung gestellt, so dass sich der benötige Zwischenspeicher im Gesamtalgorithmus nicht vergrößert. Aus Lemma 3.2 folgt also ein Speicherplatzbedarf der Größenordnung $O(n)$. □

Es bleibt abschließend festzuhalten, dass sowohl die Komplexität als auch der Speicherplatzbedarf des Uncovering-by-bases-Algorithmus insbesondere von der Größe des zu Grunde liegenden Uncovering durch Basen abhängt.

4 Sekundärliteratur

- Robert Francis Bailey (Queen Mary, University of London): *Permutation Groups, Error-Correcting Codes and Uncoverings*, Thesis submitted to the University of London for the degree of Doctor of Philosophy.
- Markus Grassl (Universität Karlsruhe): *Algorithmen für Gruppen und Codes*, Vorlesungsmitschrift aus dem WS 04/05.