Het Verhoeff algoritme

Bachelor scriptie, Radboud Universiteit, Nijmegen

Sipho Kemkes
Begeleider: Sep Thijssen
Tweede lezer: Wieb Bosma

1 december 2023

Inhoudsopgave

1 Introductie

Iedereen maakt wel eens een foutje bij het opschrijven of intypen van een getal. Zelfs bij computers gaat het soms fout waardoor een getal wordt veranderd. Hoe groot de gevolgen van zo'n fout zijn kan verschillen. Bij het maken van huiswerk is het jammer, maar valt de impact mee. Als je je IBAN ergens moet invullen en je maakt een foutje, kunnen de gevolgen veel groter zijn, iemand anders kan je prijs van de loterij krijgen bijvoorbeeld. Om een fout te detecteren, heeft je IBAN gelukkig twee controlecijfers toegevoegd zodat kleine foutjes gedetecteerd worden [4]. In een IBAN zijn dit de twee cijfers die direct na de landcode staan. Deze cijfers zijn extra toegevoegd om te helpen detecteren of er een fout gemaakt is, bij het intypen bijvoorbeeld of bij communicatie tussen computers.

Het kunnen detecteren van deze fouten is iets waar al langere tijd mensen mee bezig zijn. In de jaren '60 dachten een aantal hiervan dat het niet mogelijk was om alle enkele fouten (een enkel cijfer veranderen) en alle transpositiefouten (twee cijfers naast elkaar omwisselen) te kunnen detecteren in een decimaal systeem met maar een enkel controlecijfer en niet twee zoals bij de IBAN. Met een decimaal systeem wordt bedoeld dat het gaat om een controlecijfer te bereken over een getal in decimale notatie. Een algoritme dat erg dichtbij komt is het Luhn algoritme [5], maar deze mist helaas verwisselingen van 0 en 9. Ondanks dit wordt het Luhn algoritme wel wat gebruikt door de simpelheid van het algoritme. De vraag 'Bestaat er een decimale controlecijfercode die zowel alle enkele fouten detecteert als alle transpositiefouten detecteert?' is hiermee nog niet definitief beantwoord.

Het blijkt dat het antwoord op die vraag ja is. In 1969 publiceerde J. Verhoeff een algoritme in [6, p. 95] dat op decimale getallen werkt en alle enkele fouten en alle transpositiefouten detecteert. De manier waarop ligt niet voor de hand, gezien het de niet abelse groep D_5, de dihedrale groep van orde 10, gebruikt. Recenter in 2004 is het Damm algoritme [2, H7] [3, H10.7.6] gepubliceerd, deze kan net als het Verhoeff algoritme alle enkele fouten en transpositiefouten detecteren. Dit algoritme maakt gebruik van een quasigroep, wat dat is en hoe dat werkt is niet iets waar we hier op ingaan.

De reden dat D_5 wordt gebruikt, komt door de problemen die je hebt met rekenen modulo 10. Dit betekent werken in de ring $\mathbb{Z}/10\mathbb{Z}$ waar je nuldelers hebt omdat 10 geen priemgetal is en beide operaties commutatief zijn. De beperkingen die je krijgt door de nuldelers zorgen ervoor dat je niet genoeg opties overhoudt om ook nog voldoende rekening te kunnen houden met het feit dat alles commutatief is. Dit is iets wat bewezen gaat worden.

Codes zoals het Verhoeff algoritme en die gebruikt wordt voor je IBAN zijn fout detecterende codes, dit betekent dat ze gemaakt zijn om fouten te detecteren, maar ze geven niet aan waar het fout zit en wat het zou moeten zijn. Codes die dit verbeteren wel kunnen zijn fout verbeterende codes, dit is een klasse van codes waar we hier niet naar gaan kijken.

In deze scriptie staat de volgende, al eerder genoemde, vraag centraal:

"Bestaat er een decimale controlecijfercode die zowel alle enkele fouten detecteert als alle transpositiefouten detecteert?"

Om een antwoord te krijgen op deze vraag gaan we als eerste kijken naar een algemene definitie van wat een controlecijfercode is in Sectie 2. Hierna kijken we

in Sectie 3 naar welke soorten fouten er zijn er die je kan maken en wanneer je alle fouten van een soort kan detecteren. Daarna gaan we in Sectie 4 dieper in op codes gebaseerd op $\mathbb{Z}/10\,\mathbb{Z}$ en kijken we waarom deze niet goed genoeg zijn om alle enkele fouten en transpositiefouten te detecteren. Dit wordt gevolgd door codes gebaseerd op de dihedrale groep in Sectie 5. Hierbij kijken we waarom een simpele code hiervan ook nog niet genoeg is. Ten slotte wordt in Sectie 6 het Verhoeff algoritme uitgelegd en bewijzen we waarom deze wel alle enkele fouten en alle transpositiefouten kan detecteren.

2 Controlecijfers

Definitie 2.1 (Blokcode, alfabet, boodschap(-lengte), bloklengte, codewoord)**.** Een *blokcode* is een injectieve afbeelding $C\colon A^n \to A^m$, waarbij A een eindige verzameling, en n, m natuurlijke getallen. In deze context noemen we A het *alfabet*, een element van A^n een *boodschap*, n de *boodschaplengte*, en m de *bloklengte*. De verzameling van *codewoorden* wordt gegeven door het beeld $C(A^n)$. We zien A^n hier als het product en elementen hiervan dan als rijtjes van n elementen van A.

Merk op dat de lengte van een codewoord niet hetzelfde hoeft te zijn als de lengte van de boodschap voordat deze wordt gecodeerd. Wel is het zo dat alle codewoorden dezelfde lengte hebben.

Voorbeeld 2.2. De IBAN is een voorbeeld van een blokcode, waarbij $m = n+2$. De waarde van n verschilt tussen verschillende landen, maar is constant binnen een land [4]. Het alfabet bestaat uit de cijfers 0 tot en met 9 en de letters A tot en met Z. De injectieve afbeelding C is de rest modulo 97 van de bankcode, rekeningnummer en de landcode aan het einde, hierbij worden de letters door twee cijfers vervangen om de rest te kunnen nemen. Dit volgens A = 10 tot en met Z = 35. Het beeld is dan eerst de landcode, dan de rest, dan de bankcode en als laatste het rekeningnummer.

Stel je voor dat je hier in Nederland een bankrekening hebt bij de ING Bank met rekeningnummer 0123456789. Dan krijgen we:

$$INGB0123456789NL \mod 97 = 18231611012345678923 21 \mod 97 = 69$$

De volledige IBAN is dan NL69INGB0123456789.

2.1 Controlecijfercode

Het idee van een controlecijfer code is om een extra cijfer toe te voegen aan het einde (of begin) van het getal om te zorgen dat het resultaat aan bepaalde voorwaarden voldoet.

Definitie 2.3 (Controlecijfercode, controlecijfer)**.** Zij A een alfabet. Een *controlecijfercode* is een blokcode van de vorm $A^n \to A^{n+1}, w \mapsto (w, f(w))$, waarbij $f(w) : A^n \to A$ het *controlecijfer* is van de controlecijfercode.

Een controlecijfercode is dus een blokcode met $m = n + 1$. In het Engels wordt een controlecijfer een *check digit* genoemd.

Notatie 2.4. In de rest van dit document, hebben de volgende symbolen een vaste betekenis, tenzij anders aangegeven:

- A is het alfabet,
- q is het aantal elementen van A,
- n is de boodschaplengte,
- $C\colon A^n \to A^m$ is een blokcode,
- $C(A^n)$ de verzameling van alle codewoorden van een code,
- $f\colon A^n \to A$ is een controlecijfer met $w \mapsto (w, f(w))$ de bijbehorende controlecijfercode.

Voorbeeld 2.5 (ISBN-13 [1])**.** ISBN staat voor International Standard Book Number en ISBN-13 is de variant die sinds het begin van 2007 in gebruik is. Er geldt $A = \mathbb{Z}/10\,\mathbb{Z}$ en $n = 12$, waarmee de codewoorden 13 cijfers hebben. De cijfers van de boodschap worden afwisselend vermenigvuldigd met 1 en 3. Het resultaat hiervan wordt bij elkaar opgeteld en daarna modulo 10 genomen. Dit geeft een cijfer tussen 0 en 9, deze wordt van 10 afgehaald om het controlecijfer te krijgen. Als het controlecijfer hierdoor 10 zou worden, wordt deze naar 0 gezet. In een formule krijg je dan:

$$f(w_1, \ldots, w_{12}) = 10 - ((w_1 + 3w_2 + w_3 + 3w_4 + \cdots + 3w_{12}) \mod 10) \mod 10.$$

Zeker als $q \leq 10$ wordt vaak de link gelegd tussen de elementen van A en de cijfers $0, 1, .., q-1$. Dit betekent dat er ook een link kan worden gelegd naar $\mathbb{Z}/q\,\mathbb{Z}$ of als q priem $\mathbb{F}_q$. Dit betekent dat het controlecijfer modulo q of in $\mathbb{Z}/q\,\mathbb{Z}$ wordt berekend. Veel voorkomende waardes voor q zijn 2 en 10. Het geval $q = 2$ zie je veel terugkomen bij communicatie tussen computers en over het internet. Het geval $q = 10$ komt vooral voor bij getallen die wij als mensen ook gebruiken, zoals bijvoorbeeld bij de ISNB-13 code. Als $q = 2$ wordt het controlecijfer ook wel een *controlebit* genoemd, of in het Engels *check bit*. Als $q = 10$ wordt de code ook wel een decimale controlecijfercode genoemd.

Er zijn ook codes die meer dan één controlecijfer toevoegen, zoals bij de IBAN, maar gezien een van de eisen van Verhoeff is dat er maar één cijfer wordt toegevoegd, kijken we niet naar deze mogelijkheid.

Voorbeeld 2.6 (Pariteitscontrole)**.** Een voorbeeld van een controlebitcode is de zogenoemde pariteitscontrole. Hierbij geldt

$$q = 2,\ A = \mathbb{Z}/2\,\mathbb{Z},\ f(w) = \sum_{i=1}^{n} w_i.$$

De resulteerde code genereert codewoorden met een even aantal keer een 1.

Definitie 2.7 (Lineariteit)**.** Een controlecijfer(code) heet *lineair* als $A = \mathbb{Z}/q\,\mathbb{Z}$ en $f(w_1, w_2, .., w_n) = \sum_{i=1}^{n} x_i w_i$ voor zekere $x_i \in \mathbb{Z}/q\,\mathbb{Z}$.

Opmerking 2.8. De eis $A = \mathbb{Z}/q\,\mathbb{Z}$ is in toepassingen te omzeilen met een bijectie $g : A \leftrightarrow \mathbb{Z}/q\,\mathbb{Z}$. Dan wordt $f(w_1, w_2, .., w_n) = \sum_{i=1}^{n} x_i g(w_i)$

De pariteitscontrole hierboven is een lineaire code en in 3.4 en 3.5 staan ook enkele voorbeelden van lineaire codes.

2.2 Doel van de codes

Het doel van een controlecijfer code is om te detecteren of er een fout gemaakt is. Iemand met slechte bedoelingen zou het hele getal aan kunnen passen, waardoor een controlecijfer code niet geschikt is voor beveiliging voor aanvallen van buitenaf. Wel zorgt zo'n code voor beveiliging tegen typefouten en mogelijke storingen bij het versturen van een getal, waarbij meestal niet het hele getal verandert. Er bestaan ook codes die fouten niet alleen detecteren, maar ook kunnen verbeteren, maar dat is niet waar we hier in geïnteresseerd zijn.

Simpele fouten zijn enkele fouten, waarbij één cijfer verandert is in een ander cijfer, en transposities, waarbij twee cijfers naast elkaar zijn verwisseld.

Het doel van het Verhoeff-algoritme is om alle enkele fouten én alle transpositie fouten te detecteren, samen met maar één controlecijfer toevoegen en niet meerdere. Hoe 'fouten' precies gedefinieerd zijn staat in de volgende Sectie.

3 Foutdetectie

De standaard theorie van foutdetecterende codes kijkt vooral enkele fouten en soms naar het geval dat er meerdere enkele fouten worden gemaakt. Hier gaan we kijken naar ook een aantal anderen foutsoorten zoals transpositiefouten en dubbele fouten.

3.1 Fouten

Hoe dit foutieve codewoord tot stand komt, kan erg verschillen. Meestal komt het door ruis bij het versturen van een boodschap, maar ook iets als een typefout kan een fout veroorzaken.

Het doel van een foutdetecterende code, zoals een controle cijfercode, is om te zorgen dat zoveel mogelijk fouten gedetecteerd worden.

Definitie 3.1 (Foutsoorten)**.**

1. Een *enkele fout op plek* j, waarbij $j \in \{1, \ldots, m\}$, is een $(w_1, \ldots, w_m) \in A^m$ waarvoor geldt dat er een $(c_1, \ldots, c_m) \in C(A^n)$ is met $c_j \neq w_j$ en $c_i = w_i$ als $i \neq j$. De verzameling van alle enkele fouten wordt aangeduid met $F_e(C)$.

   We geven dit ook wel kortweg aan met $a \to b$, één van de cijfers (a) verandert in een ander cijfer (b).

2. Een *transpositiefout op plek* j *en* $j+1$, waarbij $j \in \{1, \ldots, m-1\}$, is een $(w_1, \ldots, w_m) \in A^m$ waarvoor geldt dat er een $(c_1, \ldots, c_m) \in C(A^n)$ is met $c_j = w_{j+1}$ en $c_{j+1} = w_j$ en $c_j \neq c_{j+1}$ en $c_i = w_i$ als $i \neq j$ en $i \neq j+1$. De verzameling van alle enkele fouten wordt aangeduid met $F_t(C)$.

   We geven dit ook wel kortweg aan met $ab \to ba$, een paar cijfers (ab) dat omdraait (ba).

3. Een *dubbele fout op plek* j *en* $j+1$, waarbij $j \in \{1, \ldots, m-1\}$, is een $(w_1, \ldots, w_m) \in A^m$ waarvoor geldt dat er een $(c_1, \ldots, c_m) \in C(A^n)$ is met $c_j \neq w_j$ en $c_{j+1} \neq w_{j+1}$ en $c_j = c_{j+1}$ en $w_j = w_{j+1}$ en $c_i = w_i$ als $i \neq j$ en $i \neq j+1$. De verzameling van alle enkele fouten wordt aangeduid met $F_d(C)$.

We geven dit ook wel kortweg aan met $aa \to bb$, een paar dezelfde cijfers (aa) dat verandert in een ander paar dezelfde cijfers (bb).

4. Een *fonetische fout op plek j en $j+1$*, alleen gedefinieerd als $q = 10$, waarbij $j \in \{1, \ldots, m-1\}$, is een $(w_1, \ldots, w_m) \in A^m$ waarvoor geldt dat er een $(c_1, \ldots, c_m) \in C(A^n)$ is met een van de volgende twee opties:

   - $c_j = w_{j+1}$ en $c_{j+1} = 0$ en $w_j = 1$ en $c_i = w_i$ als $i \neq j$ en $i \neq j+1$.
   - $c_{j+1} = w_j$ en $c_j = 1$ en $w_{j+1} = 0$ en $c_i = w_i$ als $i \neq j$ en $i \neq j+1$.

   De verzameling van alle enkele fouten wordt aangeduid met $F_f(C)$.

   We geven dit ook wel kortweg aan met $a0 \leftrightarrow 1a$, een paar cijfers ($a0$) dat verandert in een ander paar cijfers ($1a$) of vice versa.

5. Een *sprong-transpositiefout op plek j en $j+2$*, waarbij $j \in \{1, \ldots, m-2\}$, is een $(w_1, \ldots, w_m) \in A^m$ waarvoor geldt dat er een $(c_1, \ldots, c_m) \in C(A^n)$ is met $c_j = w_{j+2}$ en $c_{j+2} = w_j$ en $c_j \neq c_{j+2}$ $c_i = w_i$ als $i \neq j$ en $i \neq j+2$. De verzameling van alle enkele fouten wordt aangeduid met $F_s t(C)$.

   We geven dit ook wel kortweg aan met $abc \to cba$, een drietal cijfers (abc) dat omdraait (cba).

6. Een *sprong-dubbele fout op plek j en $j+2$*, waarbij $j \in \{1, \ldots, m-2\}$, is een $(w_1, \ldots, w_m) \in A^m$ waarvoor geldt dat er een $(c_1, \ldots, c_m) \in C(A^n)$ is met $c_j \neq w_j$ en $c_{j+2} \neq w_{j+2}$ en $c_j = c_{j+2}$ en $w_j = w_{j+2}$ en $c_i = w_i$ als $i \neq j$ en $i \neq j+2$. De verzameling van alle enkele fouten wordt aangeduid met $F_s d(C)$.

   We geven dit ook wel kortweg aan met $aba \to cbc$, een drietal cijfers (aba) dat verandert in een ander drietal cijfers (cbc).

Voorbeeld 3.2. Dan nu van elke foutsoort enkele voorbeelden met als correct codewoord $c = (7, 7, 5, 0, 5)$:

1. enkele fout: (3,7,5,0,5), (7,9,5,0,5) en (7,7,5,1,5),
2. transpositiefout: (7,5,7,0,5) en (7,7,0,5,5),
3. dubbele fout: (9,9,5,0,5),
4. fonetische fout: (7,7,1,5,5),
5. sprong-transpositiefout: (5,7,7,0,5) en (7,0,5,7,5),
6. sprong-dubbele fout: (7,7,8,0,8).

Een fonetische fout lijkt op het eerste gezicht een beetje willekeurig, maar deze foutsoort komt voort uit de taal. Mensen willen nog wel eens bijvoorbeeld 'vijftig' en 'vijftien' door elkaar halen.

Verhoeff heeft onderzoek gedaan naar de verdeling van de fouten [6, H0.5]. Hiervoor heeft Verhoeff gekeken naar literatuur die toen al beschikbaar was. Ook heeft hij gekeken naar voorbeelden uit de praktijk die beschikbaar zijn gesteld rekenkamers van de Nederlandse post en van de gemeente Amsterdam. De meest voorkomende is de enkele fout, met 60% tot 95% van de gemaakte fouten. Daarna volgen de fouten waar een cijfer is weggelaten of toegevoegd

met een percentage van 10% tot 20%, waarbij het weglaten van een cijfer de meerderheid heeft. Dit is een type fout die nog niet is benoemd en die verder ook niet zal worden toegelicht. Het cijfer 0 is het cijfer dat het vaakst wordt weggelaten. Ook zijn meerdere opeenvolgende cijfers een zwakke plek.

Transpositiefouten zijn de meest gemaakte fouten waarbij twee cijfers veranderen, en de andere fouten met twee cijfers zijn niet veel voorkomend, met $0,5\%$ tot $1,5\%$, maar hebben wel de mogelijkheid om detectie volledig te omzeilen, dus het is handig om hier nog steeds wel naar te kijken. De laatste groep fouten zijn 'overige' fouten, waarbij er geen duidelijke relatie te vinden is tussen het correcte getal en het foutieve. Dit heeft een voordeel en een nadeel, als eerste is het zo dat alle pogingen om fouten te detecteren helpt met het detecteren van deze willekeurige fouten, maar dit heeft als nadeel dat elke code ongeveer even goed is in het detecteren van deze willekeurige fouten.

Definitie 3.3 (Foutsoort detectie)**.** Zij C een blokcode, dan heet deze:

1. enkele-fout detecterend als $F_e \cap C(A^n) = \emptyset$,
2. transpositiefout detecterend als $F_t \cap C(A^n) = \emptyset$,
3. dubbele-fout detecterend als $F_d \cap C(A^n) = \emptyset$,
4. fonetische-fout detecterend als $F_f \cap C(A^n) = \emptyset$,
5. sprong-transpositiefout detecterend als $F_{st} \cap C(A^n) = \emptyset$,
6. sprong-dubbele-fout detecterend als $F_{sd} \cap C(A^n) = \emptyset$.

Voorbeeld 3.4. Een voorbeeld van een lineaire controlecijfercode is de code met als controlecijfer $f(w) = \sum_{i=1}^{n} w_i$. Deze kan alle enkele fouten detecteren, maar geen enkele transpositiefout.

Voorbeeld 3.5. Een ander voorbeeld is om als lineair controlecijfer met $q = 10$ te nemen $f(w) = \sum_{i=1}^{n} x_i w_i$ waar $x_i = 1$ als i oneven en $x_i = 3$ als i is even. Deze resulterende code kan nog steeds alle enkele fouten detecteren, zie Gevolg 3.8. Een deel van de transpositie fouten wordt ook gedetecteerd, maar niet allemaal. Zo wordt $12 \to 21$ wel gedetecteerd, maar $50 \to 05$ niet.

Opmerking 3.6. In het geval dat een code een deel van een foutsoort wel detecteert, bijvoorbeeld de code hierboven in Voorbeeld 3.5 detecteert transpositiefouten $ab \to ba$ niet als het verschil tussen a en b 5 is. Dit zijn tien van de negentig mogelijke transpositiefouten, en dat zeg je ook wel dat deze code $\frac{80}{90} \approx 88,9\%$ van de transpositiefouten detecteert.

Stelling 3.7. *Voor een controlecijfercode met controlecijfer f geldt dat deze enkele-fout detecterend is, dan en slechts dan als voor alle $i = 1, 2, \ldots, n$ de functie*
$g : A \to A$ gegeven door

$$g_i(a) = f(w_1, \ldots, w_{i-1}, a, w_{i+1}, \ldots, w_n)$$

injectief is voor alle $w_1, \ldots, w_{i-1}, w_{i+1}, w_n \in A$.

Bewijs. Stel dat een g_i niet injectief is voor zekere i, dan heb je een $a \neq b$ met $g_i(a) = g_i(b)$, dus

$$f(w_1, \ldots, w_{i-1}, a, w_{i+1}, \ldots, w_n) = f(w_1, \ldots, w_{i-1}, b, w_{i+1}, \ldots, w_n),$$

voor zekere $w_1, \ldots, w_{i-1}, w_{i+1}, w_n$. Dit betekent dat de enkele fout waarbij a wordt verwisseld met b niet gedetecteerd wordt, want per definitie van de code geldt

$$c_1 = (w_1, \ldots, w_{i-1}, a, w_{i+1}, w_n, g_i(a)) \in C(A^n).$$

Dus als we één positie veranderen dan krijgen we een enkele fout:

$$e = (w_1, \ldots, w_{i-1}, b, w_{i+1}, w_n, g_i(a)) \in F_e.$$

Omdat $g_i(a) = g_i(b)$, krijgen we ook

$$e = (w_1, \ldots, w_{i-1}, b, w_{i+1}, w_n, g_i(b)) \in C(A^n).$$

Dit betekent dat $e \in C(A^n) \cap F_e$, en die is dus niet leeg, en dus is f niet enkele fout detecterend. Dus g_i is injectief voor alle i

Dan nu het bewijs voor de andere kant op. Stel dat f niet enkele-fout detecterend is, dan zijn er een i, en een paar $a \neq b$ met

$$f(w_1, \ldots, w_{i-1}, a, w_{i+1}, \ldots, w_n) = f(w_1, \ldots, w_{i-1}, b, w_{i+1}, \ldots, w_n),$$

voor zekere $w_1, \ldots, w_{i-1}, w_{i+1}, w_n$. Dit betekent dat $g_i(a) = g_i(b)$, wat een tegenspraak is. Dus moet gelden dan f enkele-fout detecterend is. □

Gevolg 3.8. *Zij $f(w) = \sum_{i=1}^{n} x_i w_i$ een lineair controlecijfer, dan is de bijbehorende code enkele-fout detecterend $\Leftrightarrow$ $\gcd(x_i, n) = 1$ voor alle i.*

Bewijs. ($\Rightarrow$) Stel $\gcd(x_i, n) > 1$ voor een i. Dan is er een $a \in \mathbb{Z}/q\mathbb{Z}$ met $a \neq 0$ en $x_i \cdot a = 0$. Dit betekent dat de enkele fout waarbij a wordt verwisseld met 0 dus niet te detecteren is. Dus is in dit geval de code niet enkele-fout detecterend. Dit is in tegenspraak met de aanname dat f wel enkele-fout detecterend is en dus moet gelden $\gcd(x_i, n) = 1$ voor alle i.

($\Leftarrow$) Gezien we weten dat $\gcd(x_i, n) = 1$, geldt dus dat de vermenigvuldiging met x_i dus injectief is. Dit geldt voor iedere i en dus kunnen we 3.7 toepassen die geeft dat f dus enkele-fout detecterend moet zijn. □

Verhoeff heeft een soortgelijke stelling bewezen [6, Thm 3.0]. In deze stelling wordt uitgegaan van $f = \sum_{i=1}^{n} f_i(w_i)$ met $A = \mathbb{Z}/10\mathbb{Z}$ en de stelling zegt dat de code hiervan enkele fout detecterend is als iedere f_i een permutatie is.

Stelling 3.9. *Zij f een controlecijfer, zij de code die bij deze f hoort enkele-fout detecterend. Zij $\sigma \in Sym(A)$, dan geldt dat de code horende bij het controlecijfer $\sigma \circ f$ ook enkele-fout detecterend is.*

Bewijs. Zij $e \in F_e$, dan is er een codewoord c zodat e een enkele fout is van c, $a \to b$ ($a \neq b$). Zij i de locatie van de enkele fout, dan zijn er twee opties, $i = n + 1$ of $i \leq n$.

Het geval $i \leq n$:

$$f(w_1, \ldots, w_{i-1}, a, w_{i+1}, \ldots, w_n) \neq f(w_1, \ldots, w_{i-1}, b, w_{i+1}, \ldots, w_n).$$

Dit omdat de code die hoort bij f enkele-fout detecterend is. Als we nu kijken naar wat er gebeurt als deze fout wordt gemaakt als we als controlecijfer $\sigma \circ f$ hebben, dan krijgen we:

$$\sigma \circ f(w_1, \ldots, w_{i-1}, a, w_{i+1}, \ldots, w_n) \neq \sigma \circ f(w_1, \ldots, w_{i-1}, b, w_{i+1}, \ldots, w_n).$$

En hiermee zijn in ieder geval enkele fouten die niet het controlecijfer veranderen te detecteren met de code die hoort bij het controlecijfer $\sigma \circ f$.

Dan voor het geval $i = n + 1$:

$$(\sigma \circ f)(w_1, \ldots, w_n) = a \neq b.$$

Dit betekent dat het direct te detecteren is als de enkele fout op deze positie plaatsvindt. Daarmee is dus bewezen dat ook de code die hoort bij $\sigma \circ f$ enkele-fout detecterend is. □

Lemma 3.10. *Zij f een controlecijfer met de bijbehorende code. Schrijf een woord w als $[v, a]$, met $v \in A^{n-1}$. Dan detecteert de code transpositiefouten op de laatste plaats als $a = f([v, a])$ de enige oplossing van $a = f([v, f([v, a])])$ is voor alle v en a.*

Bewijs. Neem v en a willekeurig. Laat als eerste duidelijk zijn dat $a = f([v, a])$ altijd een oplossing is:

$$f([v, f([v, a])]) = f([v, a]) = a.$$

Als dit inderdaad de enige oplossing is, betekent dit dat alle transpositiefouten op de laatste plek gedetecteerd worden, gezien voor een transpositiefout $ab \to ba$ moet gelden $a \neq b$ en in dit geval geldt juist $a = b$. □

4 Decimale controlecijfercodes

We gaan nu kijken naar controlecijfercodes gebaseerd op $\mathbb{Z}/q\mathbb{Z}$ en de rekenkundige operaties daarin. In het bijzonder zijn we geïnteresseerd in het geval $q = 10$.

Notatie 4.1. In Sectie 4 geldt $A = \mathbb{Z}/q\mathbb{Z} = \{0, 1, \ldots, q-1\}$ voor zekere $q > 1$. Als we rekenen met elementen van A, dan is dat altijd modulo q.

Als eerste kijken we waarom lineaire controlecijfercodes niet werken om een decimale controlecijfercode te krijgen die alle enkele fouten en alle transpositie fouten detecteert.

Stelling 4.2. *Voor q even is er geen lineaire controlecijfercode is die alle enkele fouten én alle transpositiefouten detecteert.*

Bewijs. Zij $f(w) = \sum_{i=1}^{n} x_i w_i$ en stel dat de bijbehorende code alle enkele fouten én alle transpositiefouten kan detecteren.

Voor enkele fouten kijken we naar Gevolg 3.8 om te zien dat x_i oneven moet zijn voor alle i, gezien de $\text{ggd}(x_i, n) = 1$ als we alle enkele fouten willen kunnen detecteren.

Dan kijken we de detectie van transpositiefouten. Omdat x_i en x_{i+1} oneven zijn, hebben we een k zodat $x_{i+1} = x_i + 2k$ Neem $a = 0$ en $b = \frac{q}{2}$, dan is de transpositiefout $ab \to ba$ niet te detecteren. Dit omdat:

$$x_i \cdot 0 + x_{i+1} \cdot \frac{q}{2} = (x_i + 2k)\frac{q}{2} = x_i \cdot \frac{q}{2} + x_{i+1} \cdot 0.$$

Gezien deze transpositiefout niet te detecteren is, is de code dus niet transpositiefout detecterend.

□

Stelling 4.3. *Een lineaire controlecijfercode met $q \geq 3$ en q oneven en x_i afwisselend 1 en 2, zodat $x_n = 1$. Deze code detecteert alle enkele fouten en alle transpositiefouten.*

Bewijs. Als eerste de enkele fouten: omdat $\text{ggd}(1, q) = 1$ en $\text{ggd}(2, q) = 1$ is Gevolg 3.8 direct toepasbaar en is de code dus enkele-fout detecterend.

Voor de transpositiefouten kijken we naar de vraag of de vergelijking $x_i w_i + x_{i+1} w_{i+1} = x_i w_{i+1} + x_{i+1} w_i$ een andere oplossing heeft dat $w_i = w_{i+1}$.

$$\begin{aligned} x_i w_i + x_{i+1} w_{i+1} &= x_i w_{i+1} + x_{i+1} w_i \\ (x_i - x_{i+1}) w_i &= (x_i - x_{i+1}) w_{i+1} \\ w_i &= w_{i+1} \end{aligned}$$

Gezien de enige oplossing $w_i = w_{i+1}$ is, hoeven we alleen nog te kijken naar transpositiefouten op de laatste plaats. Hiervoor kijken we naar Lemma 3.10 en is te zien dat we willen kijken naar de vergelijking $f([v + f([v, a])]) = a$. Noem $\sum_{i=1}^{n-1} x_i v_i = b$ Dit uitwerken geeft:

$$f([v + f([v, a])]) = b + x_n f([v, a]) = b + x_n(b + x_n a) = 2b + a = a.$$

Om dit op te lossen moet gelden $2b = 0$, ofwel $b = 0$. Dit is precies de oplossing $f([v, a]) = a$, want $f([v, a]) = b + a = a$. Hiermee geeft Lemma 3.10 dat deze code dus ook alle transpositiefouten aan het einde detecteert en daarmee dus transpositiefout detecterend is. □

Stelling 4.4. *Zij $A = \mathbb{Z}/10\,\mathbb{Z}$, dan is de controlecijfercode gegeven door*

$$f(w) = \sum_{i=1}^{n} x_i w_i^2 + y_i w_i$$

met $x_i \neq 0$ voor alle i enkele-fout detecterend dan en slechts dan als $x_i = 5$ voor alle i en als $y_i \in \{2, 4, 6, 8\}$ voor alle i. Verder is zo'n code nooit zowel enkele-fout detecterend als transpositiefout detecterend.

Bewijs. We nemen aan dat f alle enkele fouten kan detecteren. Dit geeft de volgende beperkingen op x_i en y_i met gebruik van Stelling 3.7:

Er geldt $y_i \neq 0$. Gezien $1^2 = 9^2 \mod 10$ geldt, geldt dus ook $x_i \cdot 1^2 = x_i \cdot 9^2 \mod 10$. Dus moet gelden $y_i \neq 0$.

Er geldt x_i of y_i oneven. Als x_i en y_i beide even zijn krijg je:

$$x_i \cdot 5^2 + y_i \cdot 5 = 0 + 0 = 0 = x_i \cdot 0^2 + y_i \cdot 0$$

en is de fout $5 \to 0$ niet te detecteren.

Er geldt x_i of y_i even. Als x_i en y_i beide oneven zijn, wordt het:

$$x_i \cdot 5^2 + y_i \cdot 5 = 5 + 5 = 0 = x_i \cdot 0^2 + y_i \cdot 0$$

en is de fout $5 \to 0$ niet te detecteren.

Er geldt dat een van x_i en y_i even is en de ander oneven. Dit is een combinatie van de twee punten hierboven.

Er geldt: $x_i \neq 1$. Voor $x_i = 1$ en y_i even is $(10 - y_i) \to 0$ niet te detecteren:

$$(10 - y_i)^2 + y_i \cdot (10 - y_i) = 0 - 0 + y_i^2 + 0 - y_i^2 = 0 = 0^2 + y_i \cdot 0.$$

Er geldt: $x_i \neq 9$. Voor $x_i = 9$ en y_i even is $y_i \to 0$ niet te detecteren:

$$9 \cdot y_i^2 + y_i \cdot y_i = 9y_i^2 + y_i^2 = 10y_i^2 = 0 = 0^2 + y_i \cdot 0.$$

Er geldt: $x_i \neq 3$ en $x_i \neq 7$. Voor $x_i \in \{3, 7\}$ en y_i even is $(x_i \cdot y_i) \to 0$ niet te detecteren:

$$3 \cdot (3y_i)^2 + y_i \cdot 3y_i = 3^3 y_i^2 + 3y_i^2 = 7y_i^2 + 3y_i^2 = 10y_i^2 = 0 = 0^2 + y_i \cdot 0,$$

$$7 \cdot (7y_i)^2 + y_i \cdot 7y_i = 7^3 y_i^2 + 7y_i^2 = 3y_i^2 + 7y_i^2 = 10y_i^2 = 0 = 0^2 + y_i \cdot 0.$$

Er geldt: $y_i \neq 5$. Voor x_i even en $y_i = 5$ is $4 \to 6$ niet te detecteren:

$$x_i \cdot 4^2 + 5 \cdot 4 = x_i \cdot 6 + 0 = x_i \cdot 6^2 + 5 \cdot 6.$$

Er geldt: $x_i \neq 2$ en $x_i \neq 8$. Voor $x_i \in \{2, 8\}$ en $y_i \neq 5$ en oneven is $(x_i \cdot y_i) \to 0$ niet te detecteren:

$$\begin{aligned} 2 \cdot (2y_i)^2 + y_i \cdot 2y_i &= 8y_i^2 + 2y_i^2 \\ &= 10y_i^2 = 0 \\ &= 0^2 + y_i \cdot 0 \\ 8 \cdot (8y_i)^2 + y_i \cdot 8y_i &= 2y_i^2 + 8y_i^2 = 10y_i^2 = 0 = 0^2 + y_i \cdot 0. \end{aligned}$$

Er geldt: $x_i \neq 4$ en $x_i \neq 6$. En voor $x_i \in \{4, 6\}$ en $y_i \neq 5$ en oneven is $(10 - x_i y_i) \to 0$ niet te detecteren:

$$4 \cdot (10 - 4y_i)^2 + y_i(10 - 4y_i) = 4y_i^2 - 4y_i^2 = 0 = 0^2 + y_i \cdot 0,$$

$$6 \cdot (10 - 6y_i)^2 + y_i(10 - 6y_i) = 6y_i^2 - 6y_i^2 = 0 = 0^2 + y_i \cdot 0.$$

Het enige wat nu nog overblijft is $x_i = 5$ en $y_i \in \{2, 4, 6, 8\}$. Het blijkt dat voor deze x_i en y_i de enkele fouten wel te detecteren zijn. In $\mathbb{Z}/10\,\mathbb{Z}$ hebben we

$$5 \cdot w_i^2 = \begin{cases} 0 & \text{als } w_i \text{ even,} \\ 5 & \text{als } w_i \text{ oneven.} \end{cases}$$

Dan moeten we ook nog kijken naar het vermenigvuldigen van w_i met $y_i \in \{2, 4, 6, 8\}$. Dit is een functie $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \to \{0, 2, 4, 6, 8\}$ omdat y_i even is. Daarnaast zijn de deelfuncties $\{1, 3, 5, 7, 9\} \to \{0, 2, 4, 6, 8\}$ en $\{0, 2, 4, 6, 8\} \to \{0, 2, 4, 6, 8\}$ beide bijectief. Dit betekent dat het geheel $5w_i^2 + y_i w_i$ met $y_i \in \{2, 4, 6, 8\}$ ook bijectief is en dus enkele fouten nu gedetecteerd kunnen worden.

Nu we weten welke x_i en y_i ervoor zorgen dat enkele fouten gedetecteerd worden, moet er nog gekeken worden naar de transpositiefouten met deze x_i en y_i. Hiervoor kijken we naar $5 \to 0$. Dan:

$$5{\cdot}5^2{+}y_i{\cdot}5{+}5{\cdot}0^2{+}y_{i+1}{\cdot}0 = 5{+}0{+}0{+}0 = 0{+}0{+}5{+}0 = 5{\cdot}0^2{+}y_i{\cdot}0{+}5{\cdot}5^2{+}y_{i+1}{\cdot}5.$$

Gezien deze transpositiefout niet te detecteren is voor elke mogelijke y_i en y_{i+1}, betekent dit dat deze code niet transpositiefout detecterend is. □

Het is wel mogelijk om een code te construeren die alle enkele fouten detecteert en ongeveer 97,8% van de mogelijke transpositiefouten detecteert, zie [6, p. 78]. Dit betekent dat 88 van de 90 mogelijke transpositiefouten gedetecteerd worden, $\frac{88}{90} \approx 97,8\%$. Deze code is van de vorm $f(w) = \sum_{i=1}^{n} g_i(w_i)$ waar elke g_i een permutatie van $\mathbb{Z}/10\,\mathbb{Z}$ is. Deze code detecteert ongeveer 95,6% van de sprong-transpositiefouten en sprong-dubbele fouten en ongeveer 97,9% van de fonetische fouten. De constructie geeft Verhoeff in [6, H3.5 en H3.6]. Verhoeff zegt hierbij niks over of de percentages anders zijn als de fout op de laatste plek plaatsvindt. Ikzelf denk dat de percentages niet veel zullen veranderen als hier wel op gelet zou zijn, maar ik heb hier geen concreet bewijs voor en dat zal ook niet gegeven worden. Dit omdat deze groep van controlecijfercodes niet voldoende is en we in Sectie 6 enkele codes zullen zien die wel alle transpositiefouten kunnen detecteren.

Gezien het erop lijkt dat er iets anders nodig is om aan de eisen van Verhoeff te voldoen, is Verhoeff gaan kijken naar codes die niet gebaseerd zijn op de optelling in $\mathbb{Z}/10\,\mathbb{Z}$.

5 Dihedrale controlecijfercodes

Waar in de vorige sectie de link is gelegd tussen A en $\mathbb{Z}/q\,\mathbb{Z}$, kan ook een link gelegd worden met een dihedrale groep. Dit kan alleen als q even is, gezien een dihedrale groep altijd een even aantal elementen heeft.

Een gevolg van deze verandering om niet meer naar $\mathbb{Z}/q\,\mathbb{Z}$ te kijken, maar naar $\mathrm{D}_{q/2}$, is dat we nog maar één operatie hebben op de elementen. Waar we in $\mathbb{Z}/q\,\mathbb{Z}$ optelling en vermenigvuldiging hadden, hebben we in $\mathrm{D}_{q/2}$ alleen een vermenigvuldiging. Dit vermindert het aantal wiskundige bewerking binnen deze groep voor controlecijfercodes, maar deze vermenigvuldiging heeft wel betere eigenschappen met het oog op foutdetectie dan de optelling of vermenigvuldiging van $\mathbb{Z}/q\,\mathbb{Z}$. Zo is een vermenigvuldiging in het algemeen in een dihedrale

groep niet commutatief, terwijl deze vermenigvuldiging met een groepselement wel injectief is.

Definitie 5.1 (dihedrale controlecijfercode, enkelvoudig)**.** Een controlecijfer code is een *dihedrale* controlecijfercode als $A = \mathrm{D}_{q/2}$. Een dihedrale controlecijfercode heet *enkelvoudig* als

$$f(a_1, a_2, .., a_n) = \prod_{i=1}^{n} x_i a_i,$$

waarbij $x_i \in A$.

Merk op dat de definitie voor enkelvoudige dihedrale controlecijfercodes een analoog is voor de definitie voor lineaire codes in $\mathbb{Z}/q\,\mathbb{Z}$.

Een dihedrale groep D_k is een groep met $2k$ elementen en wordt multiplicatief genoteerd. Deze elementen zijn rotaties en spiegelingen van een regelmatige k-hoek, deze worden genoteerd als $r^0, r, .., r^{k-1}$ en $s, rs.., r^{k-1}s$ respectievelijk. Het eenheidselement van de groep is r^0, deze komt overeen met een rotatie van $0°$. Het eenheidselement wordt ook wel genoteerd met e. Daarnaast gelden $r^k = r^0$ en $s^2 = r^0 = (r^a s)^2$. Verder gelden $r^a r^b = r^{a+b}$ en $sr^a = r^{k-a}s$.

Voorbeeld 5.2 ($A = \mathrm{D}_1$)**.** De kleinste dihedrale groep is D_1, deze is isomorf met $\mathbb{Z}/2\,\mathbb{Z}$, waardoor deze dezelfde problemen heeft als je een controlecijfer code probeert te maken gebaseerd op deze groep.

Voorbeeld 5.3 ($A = \mathrm{D}_2$)**.** De volgende dihedrale groep D_2 blijkt isomorf te zijn met de viergroep van Klein. Deze is wel Abels, maar niet isomorf met $\mathbb{Z}/4\,\mathbb{Z}$. Met een enkelvoudige code over deze groep kan je altijd de enkele fouten detecteren door de structuur. Dit omdat vermenigvuldiging met een element in een groep een injectieve operatie is en daarmee Stelling 3.7 te gebruiken is. Transpositie fouten zijn niet altijd te detecteren. Zij x_i en x_{i+1} twee opeenvolgende factoren, dan, omdat de groep Abels is en elk element van orde hoogstens 2:

$$(x_i \cdot x_i) \cdot (x_{i+1} \cdot x_{i+1}) = e = (x_i \cdot x_{i+1}) \cdot (x_{i+1} \cdot x_i).$$

Dit betekent dat je dus nooit alle transpositiefouten kan detecteren met een enkelvoudige code over D_2.

Stelling 5.4. *Zij $(G, *)$ een groep met q elementen. Zij $A = G$ en zij $f(w) = x_1 * w_1 * x_2 * w_2 \cdots x_n * w_n$ een controlecijfer met de bijbehorende controlecijfercode. Dan kan deze code niet alle transpositiefouten detecteren.*

Bewijs. Voor een transpositiefout kijken we dus of er voor de volgende vergelijking een oplossing is naar w_i en w_{i+1}, met $w_i \neq w_{i+1}$ voor gegeven x_i en x_{i+1}. Als we hier een oplossing voor hebben, is de transpositiefout $w_i w_{i+1} \to w_{i+1} w_i$ niet te detecteren.

$$x_i * w_i * x_{i+1} * w_{i+1} = x_i w_{i+1} x_{i+1} * w_i$$

Voor het geval $x_{i+1} \neq e$, kies $w_i = x_{i+1}$ en $w_{i+1} = e$, zodat:

$$\begin{aligned} x_i * w_i * x_{i+1} * w_{i+1} &= x_i * x_{i+1} * x_{i+1} * e \\ &= x_i * x_{i+1} * x_{i+1} \\ &= x_i * e * x_{i+1} * x_{i+1} \\ &= x_i * w_{i+1} * x_{i+1} * w_i. \end{aligned}$$

In het geval dat $x_{i+1} = e$ kies $w_{i+1} \neq e$ en $w_i = e$. Dan:

$$\begin{aligned} x_i * w_i * x_{i+1} * w_{i+1} &= x_i * e * e * w_{i+1} \\ &= x_i * w_{i+1} \\ &= x_i * w_{i+1} * e * e \\ &= x_i * w_{i+1} * x_{i+1} * a_i. \end{aligned}$$ □

Gevolg 5.5. *Er is geen enkelvoudige dihedrale controlecijfercode die alle transpositiefouten detecteert.*

Bewijs. Dit is een direct gevolg van Stelling 5.4. □

6 Het Verhoeff algoritme

Dan gaan we nu kijken naar een specifieke dihedrale controlecijfercode, namelijk de Verhoeff code en het bijbehorende Verhoeff algoritme.

Notatie 6.1. In Sectie 6 geldt $A = \mathrm{D}_{q/2}$, waarmee A weer q elementen heeft. Ook wordt hier als vermenigvuldiging de vermenigvuldiging binnen $\mathrm{D}_{q/2}$ gebruikt.

Definitie 6.2. *De Verhoeff code* is een dihedrale controlecijfercode met $q = 10$ en dus $A = \mathrm{D}_5 = \langle r, s \mid r^5 = 1, s^2 = 1, rs = sr^{-1} \rangle$. Verder worden de elementen van D_5 gekoppeld aan de cijfers 0 tot en met 9 op de volgende manier:

$$\begin{array}{lllll} 0 = r^0, & 1 = r, & 2 = r^2, & 3 = r^3, & 4 = r^4, \\ 5 = s, & 6 = rs, & 7 = r^2 s, & 8 = r^3 s, & 9 = r^4 s. \end{array}$$

Zij p de permutatie

$$p = (1\ 5\ 8\ 9\ 4\ 2\ 7\ 0)(3\ 6).$$

Merk op dat p ook de elementen uit A permuteert, vanwege de identificatie van A met de decimale cijfers. Het controlecijfer van Verhoeff voor $w \in A^n$ met $w = (w_1, w_2, \ldots, w_{n-1}, w_n)$ wordt verkregen door eerst een $w_{n+1} = 0$ toe te voegen aan w, het resultaat noemen we w'. Dan is het controlecijfer is gedefinieerd door:

$$f(w') = \left(p^0(w_{n+1}) \cdot p(w_n) \cdot p^2(w_{n-1}) \cdots p^{n-1}(w_2) \cdot p^n(w_1)\right)^{-1}$$

Merk op dat het toevoegen van de w_{n+1} niets verandert aan de waarde van het controlecijfer, gezien $p^0(w_{n+1}) = p^0(0) = 0$ gekoppeld aan het eenheidselement is. De reden voor het toevoegen is dat we deze f nu ook kunnen gebruiken voor het controleren of er een fout is gemaakt.

Voorbeeld 6.3. Neem $w = \{5, 1, 9, 5\}$, dan

$$f(w) = \left(0 \cdot p(5) \cdot p^2(9) \cdot p^3(1) \cdot p^4(5)\right)^{-1} = (0 \cdot 8 \cdot 2 \cdot 9 \cdot 2)^{-1} = 4^{-1} = 1$$

waar f het Verhoeff controlecijfer.

Het nemen van de inverse aan het einde verandert niets aan de foutdetectie. Dit omdat dit een injectieve operatie is en er dus geen informatie verloren gaat. De reden hiervoor is dat het het mogelijk maakt om dezelfde functie te gebruiken voor zowel het berekenen van het controlecijfer als voor het controleren of er een fout is opgetreden. Dit controleren wordt gedaan door een ontvangen getal c als input te gebruiken voor het Verhoeff controlecijfer door te kijken naar $f(c)$. Als het nieuwe controlecijfer niet gelijk is aan 0, is het ontvangen getal geen codewoord en is er ergens een fout gemaakt. Is dit nieuwe controlecijfer wel gelijk aan 0, dan is het ontvangen getal wel een codewoord. De reden hiervoor is dat als w de boodschap is die c geeft als codewoord, dan is c_{n+1}, het controlecijfer, gelijk aan $f(w_1, \ldots, w_n, 0)$. Als je dit uitwerkt krijg je:

$$f(w_1, \ldots, w_n, 0) = (0 \cdot p(w_n) \cdots p^n(w_1))^{-1} = (p(w_n) \cdots p^n(w_1))^{-1} = c_{n+1}.$$

Als we nu kijken naar $f(c)$ krijgen we:

$$f(c) = (c_{n+1} \cdot p(c_n) \cdots p^n(c_1))^{-1} = (c_{n+1} \cdot p(w_n) \cdots p^n(w_1))^{-1} = 0^{-1} = 0.$$

Deze combinatie van het gebruik van het Verhoeff controlecijfer is het Verhoeff algoritme.

Stelling 6.4. *De Verhoeff code is enkele fout detecterend.*

Bewijs. Om dit in te zien willen we Stelling 3.7 gebruiken. Vermenigvuldiging in D_5 en het toepassen van de permutatie zijn beide bijectief. Dit betekent dat elke g_i zoals gedefinieerd in 3.7 dus ook bijectief moet zijn en dus is het Verhoeff algoritme dus enkele fout detecterend. □

Lemma 6.5. *Zij de permutatie p zoals van de Verhoeff code. Dan zijn er geen x en y met $x \neq y$ en $x \cdot p(y) = y \cdot p(x)$.*

Opmerking 6.6. Ikzelf heb helaas geen mooie manier gevonden om dit lemma elegant te bewijzen, waardoor een van de manieren is om alle opties langs te gaan, wat het makkelijkste gaat met behulp van een computer. Een volledige lijst van deze mogelijke transpositiefouten staat in Apendix A. Hieronder een schets om het aantal combinaties voor x en y flink te verminderen, waardoor het aantal opties wat nog gecheckt moet worden met bruteforce een stuk lager uitvalt.

Bewijsschets. Hier in deze bewijsschets worden de elementen aangegeven met de notatie van D_5 en niet als cijfers, dit om de argumentatie duidelijker te maken. Dit betekent dat in deze notatie geldt

$$p = (r \;\; s \;\; r^3 s \;\; r^4 s \;\; r^4 \;\; r^2 \;\; r^2 s \;\; r^0)(r^3 \;\; rs).$$

Het is van belang om in te zien dat $r^a \cdot r^b = r^{a+b}$ en $r^c s \cdot r^d s = r^{c-d}$, wat betekent dat voor de juiste keuzes van a, b, c en d dit gelijk kan zijn. Ook geldt $r^a \cdot r^b s = r^{a+b} s$ en $r^c s \cdot r^d = r^{c-d} s$, wat betekent dat ook hier voor de juiste a, b, c en d dit gelijk kan zijn. Maar dit betekent ook dat $r^a \cdot r^b = r^{a+b}$ en $r^c s \cdot r^d = r^{c-d} s$ dus nooit gelijk kunnen zijn.

Als we kijken naar bijvoorbeeld $rs \to sr$:

$$r \cdot p(s) = r \cdot r^3 s = r^4 s \neq r^0 = s \cdot s = s \cdot p(r).$$

Hier is duidelijk te zien dat de rechterkant een spiegeling s erin heeft zitten, terwijl de linkerkant dit niet heeft en dus gaan ze nooit gelijk zijn. Door alle combinaties van x en y waarbij dit het geval is te elimineren, blijven er nog maar een beperkt aantal opties over voor deze x en y die mogelijk niet te detecteren zijn. Deze zijn makkelijk allemaal na te gaan door het beperkte aantal en het blijkt dat dan ook geldt $x \cdot p(y) \neq y \cdot p(x)$. □

Stelling 6.7. *De Verhoeff code is transpositiefout detecterend.*

Bewijsschets. Hier in het bewijs worden de elementen aangegeven met de notatie van D_5 en niet als cijfers, dit om de argumentatie duidelijker te maken. Dit betekent dat in deze notatie geldt

$$p = (r \;\; s \;\; r^3 s \;\; r^4 s \;\; r^4 \;\; r^2 \;\; r^2 s \;\; r^0)(r^3 \;\; rs).$$

Voor een transpositiefout van de vorm $w_i w_{i+1} \to w_{i+1} w_i$ zijn $p^i(w_i)$ en $p^i(w_{i+1})$ ook weer elementen van D_5 per definitie. Daarmee kan je Lemma 6.5 toepassen op $v_0 = p^i(w_i)$ en $v_1 = p^i(w_{i+1})$. Hiermee is te zien dat ook $p^i(w_i) \cdot p^{i+1}(w_{i+1}) = p^i(w_{i+1}) \cdot p^{i+1}(w_i)$ geen oplossingen heeft voor $w_i \neq w_{i+1}$ en dus de transpositie fout te detecteren is.

Dan moeten we nu nog kijken naar transpositiefouten op de laatste plaats. Hiervoor hebben we Lemma 3.10. Dit Lemma geeft dat we nu alleen hoeven te kijken naar de vergelijking $f([v, f([v, a])]) = a$. Hiervoor schrijven we $p^2(w_{n-1}) \cdots p^n(w_1) = b$. De vergelijking wat uitschrijven geeft nu:

$$f([v, f([v, a])]) = (0 \cdot p(f([v, a])) \cdot b)^{-1} = (p(0 \cdot p(a) \cdot b)^{-1} \cdot b)^{-1} = a$$

Een oplossing is natuurlijk $f([v, a]) = (p(a) \cdot b)^{-1} = a$. Dit betekent dat er nog 90 paren voor a en b overblijven waarvoor het nog niet duidelijk is of de transpositiefout aan het einde te detecteren is. De beste manier is om deze met een computer met bruteforce te checken. Dan blijkt dat geen van deze paren een nieuwe oplossing geeft en daarmee zijn volgens Lemma 3.10 dus ook alle transpositiefouten aan het einde te detecteren.

Het nemen van de inverse is injectief en verandert daarmee welke fouten wel en niet gedetecteerd worden aan het einde. Dit betekent dat het Verhoeff algoritme dus transpositiefout detecterend is. □

Verhoeff heeft ook een computer gebruikt om een groot aantal opties langs te gaan om een permutatie te vinden die werkt, zie hiervoor [6, H4.4 en 4.5]. Verhoeff heeft verder geen aandacht besteed aan het geval van de transpositiefout aan het einde plaatsvindt.

De reden voor de specifieke permutatie p die gebruikt wordt is omdat deze permutatie ook het grootste deel van de fonetische fouten detecteert [6, p95]. Er zijn andere permutaties waarmee je ook alle enkele fouten en alle transpositiefouten kan detecteren, maar die zijn niet beter als het gaat om het detecteren van de fonetische fouten.

Naast de volledige detectie van enkele fouten en transpositiefouten heeft Verhoeff ook gekeken naar de detectie van andere foutsoorten [6, p95]. De Verhoeff code detecteert 95,5% van de dubbele fouten. Dit betekent dat de Verhoeff code 95,5% van de mogelijke dubbele fouten detecteert, dus in dit geval 86 van de 90 mogelijke dubbele fouten, $\frac{86}{90} \approx 95,5\%$. Op een zelfde manier

detecteert de Verhoeff code 94,2% van zowel sprong-dubbele fouten als sprong-transpositiefouten en 95,3% van de fonetische fouten.

Naast de Verhoeff code is er ook een verzameling dihedrale codes met een controlecijfer van de vorm $f(w) = \prod_{i=1}^{n} g_i(w_i)$ waarbij elke g_i een permutatie is van D_5. Er bestaan permutaties g_i waarmee de bijbehorende code alle transpositiefouten kan detecteren. Verhoeff zelf laat zien dat er 34040 permutaties p zijn met $x \cdot p(y) \neq y \cdot p(x)$ voor alle $x \neq y$ in [6, H4.4]. Dit is een handige eigenschap om te hebben voor het kunnen detecteren van transpositiefouten, zoals te zien in Lemma 6.5 de bewijsschets van Stelling 6.7. De permutatie p in de Verhoeff code is een van deze permutaties. Om deze permutaties te vinden heeft Verhoeff een zoekalgoritme [6, H3.4] gebruikt met enkele modificaties om het te laten werken in D_5.

Een deel van deze dihedrale codes kan ook alle fonetische fouten detecteren. Verhoeff geeft in [6, H4.6] een constructie hiervoor. Volgens de claim van Verhoeff detecteren deze alle enkele fouten, alle transpositiefouten, alle fonetische fouten, 95,5% van de dubbele fouten en 94,2% van sprong-dubbele fouten en sprong-transpositiefouten. Hierbij is het niet duidelijk of er ook rekening is gehouden met fouten op de laatste positie.

Referenties

[1] International ISBN Agency. *ISBN International Users Manual.* Seventh Edition, 2017.

[2] Michael H Damm. *Total anti-symmetrische Quasigruppen.* Philipps-Universität Marburg, 2004.

[3] Peter Fenwick. *Introduction to computer data representation.* Bentham Science Publishers, 2014.

[4] ECBS: European Commitee for Banking Standards. Iban: International banking account number. *EBS204 v3.2*, 2003.

[5] Hans P Luhn. *Computer for verifying numbers.* US patent 2950048A, 23-8-1960.

[6] J Verhoeff. *Error detecting decimal codes, vol. 29, Math.* Centre Tracts. Math. Centrum Amsterdam, 1969.

A Tabel 1

De tabel voor alle mogelijkheden voor x en y en de bijbehorende waardes voor $x \cdot p(y)$ en $y \cdot p(x)$, waar p de permutatie van de Verhoeff code.

x	y	$x \cdot p(y)$	$y \cdot p(x)$
r^0	r^1	$r^0 s$	r^2
r^0	r^2	$r^2 s$	r^3
r^0	r^3	$r^1 s$	r^4
r^0	r^4	r^2	r^0
r^0	$r^0 s$	$r^3 s$	$r^4 s$
r^0	$r^1 s$	r^3	$r^0 s$
r^0	$r^2 s$	r^0	$r^1 s$
r^0	$r^3 s$	$r^4 s$	$r^2 s$
r^0	$r^4 s$	r^4	$r^3 s$
r^1	r^2	$r^3 s$	$r^2 s$
r^1	r^3	$r^2 s$	$r^3 s$
r^1	r^4	r^3	$r^4 s$
r^1	$r^0 s$	$r^4 s$	r^0
r^1	$r^1 s$	r^4	r^1
r^1	$r^2 s$	r^1	r^2
r^1	$r^3 s$	$r^0 s$	r^3
r^1	$r^4 s$	r^0	r^4
r^2	r^3	$r^3 s$	$r^0 s$
r^2	r^4	r^4	$r^1 s$
r^2	$r^0 s$	$r^0 s$	r^3
r^2	$r^1 s$	r^0	r^4
r^2	$r^2 s$	r^2	r^0
r^2	$r^3 s$	$r^1 s$	r^1
r^2	$r^4 s$	r^1	r^2
r^3	r^4	r^0	$r^0 s$
r^3	$r^0 s$	$r^1 s$	r^4
r^3	$r^1 s$	r^1	r^0
r^3	$r^2 s$	r^3	r^1
r^3	$r^3 s$	$r^2 s$	r^2
r^3	$r^4 s$	r^2	r^3
r^4	$r^0 s$	$r^2 s$	$r^3 s$
r^4	$r^1 s$	r^2	$r^4 s$
r^4	$r^2 s$	r^4	$r^0 s$
r^4	$r^3 s$	$r^3 s$	$r^1 s$
r^4	$r^4 s$	r^3	$r^2 s$
$r^0 s$	$r^1 s$	$r^2 s$	r^3
$r^0 s$	$r^2 s$	$r^0 s$	r^4
$r^0 s$	$r^3 s$	r^1	r^0
$r^0 s$	$r^4 s$	$r^1 s$	r^1
$r^1 s$	$r^2 s$	$r^1 s$	$r^4 s$
$r^1 s$	$r^3 s$	r^2	$r^0 s$
$r^1 s$	$r^4 s$	$r^2 s$	$r^1 s$
$r^2 s$	$r^3 s$	r^3	$r^3 s$
$r^2 s$	$r^4 s$	$r^3 s$	$r^4 s$
$r^3 s$	$r^4 s$	$r^4 s$	r^0

Tabel 1: Simpele transpositiefout voor de Verhoeff code.