Project CodeTheorie

Verslag door Jeroen Verstraelen, Evert Heylen en Stijn Janssens

PlayFair

Hill-Climbing

Stijn Janssens heeft playfair gekraakt en dit deel van het verslag geschreven. Voor het kraken van playfair heb ik me gebasseerd op het bekende hill-climbing probleem. Concreet voor het playfair probleem gaat dit als volgt te werk:

- Neem een random sleutel
- Ontcijfer de code adhv playfair
- Geef het resultaat een bepaalde score
- Onthoud het beste resultaat
- Maak kleine aanpassingen aan de sleutel (swap 2 letters)
- Herhaal vanaf stap 2

We gingen ervan uit dat deze methode uiteindelijk wel resultaat zou opleveren. Dit idee kreeg ik omdat een gedecrypteerde playfair tekst met een sleutel die net niet correct is, behoorlijk lijkt op de correcte gedecrypteerde tekst, in tegenstelling tot bijvoorbeeld enigma. Echter deze methode bleek geen correcte resultaten te geven. De verklaring hiervoor ligt bij lokale maxima. De sleutel bereikt op een gegeven moment een min of meer degelijke tekst (volgens de computer, want vaak is deze niet leesbaar) en zit dan vast bij deze lokale maximale score. Er bestaan nog veel betere scores maar hier komt het algoritme niet.

Simulated Annealing

Om dit probleem op te lossen moest ik een aangepaste versie van hill-climbing gebruiken: Simulated Annealing Dit algoritme werkt hetzelfde als hill-climbing met als enige verschil dat het soms (random gekozen) verder werkt met een slechtere score dan voorheen. Hierdoor kan men wanneer men in een lokaal maximum zit, door verder te gaan met een slechtere score eruit geraken en naar een beter globaal maximum werken. Door dit algoritme 1,5 tot 2,5 uur te laten runnen verkregen we het correcte resultaat.

Score

Een gedecrypteerde tekst een score geven van hoe goed hij gedecrypteerd is doen we door te kijken naar quadgrammen. Deze methode werkt met de log10 kansen van het voorkomen van bepaalde quadgrammen in een correcte tekst. Vervolgens telt hij de kansen van de effectief voorkomende quadgrammen op en hoe hoger het resultaat hoe logischer het voorkomen van de quadgrammen dus hoe beter de tekst. Bij het decrypteren van playfair zijn we naief uitgegaan van het feit dat deze uit het engels was geencrypteerd, dit was eerder gelukkig toeval.

Resultaat

Voor het herschrijven van de code

Het eerste resultaat dat simulated annealing ons gaf was volgende key: ypharkwotegblinmqcdfxzsuv Wanneer we dit oorspronkelijk zagen dachten we dat we de oplossing hadden gevonden, de tekst was perfect leesbaar. Echter na betere inspectie bleek dat er toch nog foute zaken inzaten. (zie hierboven, een semi-juiste key geeft een semi-juiste tekst) Hierna ben ik overgeschakeld op manueel werk: de sleutel die we hadden was de volgende:

Y	P	H	A	R
K	W	O	T	E
G	B	L	I	N
M	Q	C	D	F
X	Z	S	U	V

maar ik wist (vanuit de les) dat de sleutel "logisch" was opgebouwd, i.e. een woord en vervolgens de resterende letters uit het alfabet. Deze sleutel ben ik dan gaan aanpassen met behulp van kolomtransposities wat ons de logischere sleutel:

H	A	R	Y	P
O	T	E	K	W
L	I	N	G	B
C	D	F	M	Q
S	U	V	X	Z

gaf. We waren er echter nog steeds niet. Onze tekst was een grote stap dichter bij perfectie gekomen maar er vormden zich problemen bij de dubbele letters. Ik

vermoedde een fout in mijn eigen decodering van playfair met betrekking tot 2 dezelfde letters naast elkaar etc. maar kon deze niet vinden. Dan kwam het verlossende bericht:

Na het herschrijven van de code

Blijkbaar was er een fout geslopen in de encodering van de tekst. Een decryptie met bovenstaande sleutel gaf echter niet zo'n goed resultaat als we gehoopt hadden. Logisch nadenken gaf ons echter de meest logische sleutel: Harry Potter J. K. Rowling, als we daar alle herhaalde letters uithalen en i = j stellen, bekomen we HARYPOTEIKWLNG, hetgeen subtiel verschillend is van bovenstaande key. Deze sleutel gaf ons het correcte resultaat: de introductie van Harry Potter and the Philosopher's stone door J.K. Rowling

Bronvermelding

Voor het vinden van deze methodes heb ik gebruik gemaakt van de cursus en volgende bronnen:

- https://crypto.stackexchange.com/questions/22522/how-does-cryptanalysis-of-the-playfair-cipher-work
- http://practicalcryptography.com/cryptanalysis/stochastic-searching/cryptanalysis-playfair/

Vigenère

Helaas wordt er niet enkel Vigenère gebruikt, maar ook een kolomtranspositie. Dit zorgt ervoor dat we niet zomaar de simpele frequentieanalyse kunnen gebruiken.

We hebben hiervoor geen trucs gevonden, dus we hebben eerst gewoon bruteforce geprobeerd. Dit gaat als volgt: we gaan alle mogelijkheden van de kolomtranspositie af. Daarna proberen we zo goed mogelijk een frequentieanalyse te doen, gebaseerd op de methode van Simon Singh. 'Zo goed mogelijk' is gedefinieerd door de som van de verschillen in frequentie voor alle letters. We hebben hier ook varianten van getest maar deze gaf het beste resultaat.

Om deze methode te verifiëren, hebben we deze (met succes!) geprobeerd op een aantal bestanden die we zelf gegenereerd hebben. Het ging wel wat te traag, dus hebben we het hele geval herschreven in C++, waarna het 10 tot 100 keer sneller ging.

Maar helaas, voor alle 4 mogelijke talen die overbleven na Playfair (Duits, Spaans, Nederlands, Frans) werd er geen goede oplossing gevonden (met lengte sleutel

<= 11 voor de kolomtranspositie en lengte sleutel <= 20 voor de Vigenère sleutel). Als we het gedrag van het programma bij onze voorbeeldbestanden mogen geloven is het enige wat we kunnen zeggen dat er vrij veel S'en in de Vigenère sleutel zitten.

De volgende stap zou zijn om de score functie uit te breiden en het hele geval om te vormen naar een volwaardige hill-climbing implementatie.

Enigma

Enigma Simulator

We hebben ons allemaal samen geworpen op enigma, helaas zonder resultaat. Voor het kraken van enigma zijn we begonnen met het schrijven van een implementatie van de enigma machine. Deze hebben we geverifieerd met voorbeelden uit de cursus en zelf opgestelde voorbeelden, zodat we zeker waren dat ze correct functioneerde.

Methode 1

Aangezien methode 1 uit de cursus een min of meer simpele oplossing geeft tot het kraken van enigma, hebben we besloten om deze eerst te testen. Hiervoor hebben we een crib graph uitgetekend.

graph.jpg

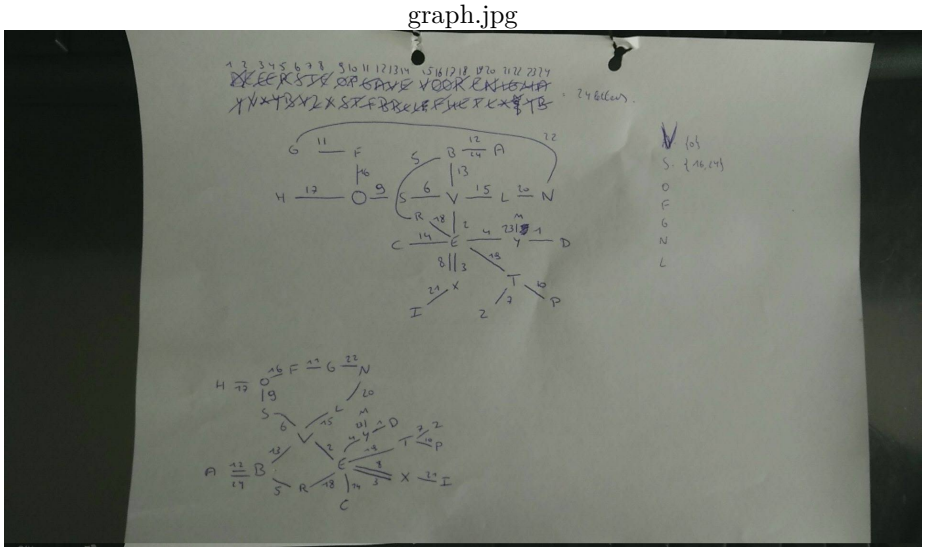

Figure 1: Crib graph

Er zijn echter niet voldoende gesloten paden op deze crib graph om enigma volledig mee te kraken, deze methode had dus in geen geval een volledig sluitende oplossing kunnen geven. Ook het uitwerken van de methode voor verschillende gesloten paden gaf tegenstrijdige oplossingen dus we hebben uiteindelijk besloten om ze te laten varen.

Methode 2

Methode 2 voor het kraken van Enigma is gebasseerd op de Turing Bombe. Deze methode geeft een sluitende oplossing voor zowel de startpositie van de Enigma machine als de configuratie van het steckerbord. Om onze interpretatie en implementatie van de methode te verifieren hebben we een eigen test file gemaakt. Van deze testfile weten we de correcte startposities en plugboard configuraties, zo kunnen we testen of de methode een correct resultaat geeft. Mits deze voorkennis en logische keuze van startwaarden geeft onze methode inderdaad een correcte startpositie en een deels correct steckerbord. Voor het toepassen op de enigma tekst die we moeten kraken heeft de methode ons echter 12 mogelijke startposities gegeven en een zeer onvolledig steckerbord. Concreet toegepast op de crib slagen we erin om maximaal 10 letters juist te decoderen aan de hand van de bekomen resultaten.

ADFGVX

Methode

Bij het kraken van ADFGVX hebben we een aantal methodes doorlopen voordat we bij de juiste uitkwamen. We beginnen natuurlijk altijd met het omzetten van de morse code naar zijn overeenkomstige ADFGVX code. Hierna moeten we de kolomtranspositie kraken om een tekst te verkrijgen die enkel nog vercijferd is via het polybiusvierkant. Het eerste dat we probeerden was een brute force methode die voor een gegeven key lengte alle permutaties voor de kolomtranspositie afgaat. Voor elke permutatie berekenden we de frequentie van de bigrammen (AD, AG, ...) en koppelden we deze in het polybiusvierkant aan een alfanumeriek teken dat ongeveer met dezelfde frequentie in de doeltaal voorkomt. Met het bekomen vierkant konden we dan de tekst decrypteren en berekenen hoe dicht deze bij de doeltaal zit via dezelfde score functie die we in playfair gebruiken. Deze methode had in theorie de oplossing kunnen vinden maar was in de praktijk te traag.

Na lang zoeken naar een manier om deze berekening per permutatie kleiner te maken zijn we uiteindelijk uitgekomen op de index of coincidence. Stel we hebben een tekst verkregen na een kolomtranspositie en vertaling door een polybiusvierkant die ingevuld is via de frequentieanalyse zoals hiervoor. Dan laat deze index ons de kans berekenen dat de resulterende tekst slechts een monoalfabetische substitutie verwijdert is van een bestaande taal. Dit is dus

de kans dat de kolomtranspositie correct is en enkel het berekende polybius vierkant verschilt met die van de oplossing. Talen zoals het Frans, Spaans, Duits en Nederlands hebben een index of coincidence rond de 0.7 en 0.8, bij het Engels is dit iets lager maar deze taal waren we al uitgekomen bij Playfair.

We moesten dus nog enkel de monoalfabetische substitutie oplossen. Omdat het polybiusvierkant is opgesteld met behulp van een frequentie analyse weten we dat het maar met enkele elementen zal verschillen met die van de oplossing. We kunnen dus willekeurig twee elementen in het vierkant verwisselen en dan de score functie gebruiken. Als die ons laat weten dat we dichter bij de doeltaal zitten kunnen we de verandering behouden. Dit kunnen we blijven doen totdat er voor een aantal iteraties geen verbeterde versie meer is voorgekomen.

Oplossing

De tekst die we zijn uitgekomen komt uit het begin van het boek La peste van Albert Camus. De kolomtranspositie had de permutatie 0,4,1,3,5,6,2 als key. De ADFGVX versleuteling gebruikt dit polybiusvierkant als sleutel:

	A	**D**	**F**	**G**	**V**	**X**
A	J		M		F	Y
D		O	C	R	p	
F	L		Z	B		U
G	T	A		V		E
V	H	K	W	D	S	
X		I	Q	X	G	N