KONINKLIJKE MILITAIRE SCHOOL
169^{ste} Promotie Polytechniek
Eregeneraal-majoor NAESSENS de LONCIN

Academiejaar 2018-2019
2^{de} Master

Optimalisatie van een algoritme voor de detectie van Advanced Persistent Threats

door Onderluitenant Kandidaat Beroepsofficier
Jordy CANSSE

Masterproef Departement Communication, Information, Systems and Sensors (CISS)
voorgelegd tot het behalen van het diploma
van master in de Ingenieurswetenschappen
onder leiding van Commandant v/h Vlw Dr. Ir. Thibault DEBATTY
Brussel, 2019

KONINKLIJKE MILITAIRE SCHOOL
Departement Communication, Information,
Systems and Sensors (CISS)

Versie 1.0
28 mei 2019

Optimalisatie van een algoritme voor de detectie van Advanced Persistent Threats

Door
Officier-leerling Jordy CANSSE

Voorwoord

Voor u ligt het summum van mijn vijf jaar durende studie Polytechniek aan de Koninklijke Militaire School. Deze studie heb ik aangevat als een nieuwsgierige jongen die een zekere zin in avontuur had en die geboeid was door wetenschappen en computers. Het avontuur was steeds terug te vinden tijdens de kampperiodes die ik blijkbaar toch niet altijd zo interessant vond. Wat mij wel steeds aansprak waren de lessen van het Departement CISS. De manier waarop Kolonel Dr. Ir. Bart Scheers les gaf, heeft zeker bijgedragen tot mijn wapenkeuze AIR CISS. Toch waren het de lessen van Professor Dr. Ir. Wim MEES en Commandant v/h Vlw Dr. Ir. Thibault DEBATTY die me het meest konden boeien.

De keuze om mijn masterproef in het kader van cyberveiligheid uit te voeren is dus vanzelfsprekend. Het onderwerp, zoals ik het uitleg aan mijn naaste familieleden, is een opvolgend werk over een systeem dat instaat voor de detectie van geavanceerde virussen. Het werk was toch iets complexer dan hier is voorgesteld. Tijdens het werk heb ik leren programmeren in Java. Eerst door zelf veel code te lezen en uiteindelijk door ze zelf te schrijven. Ik kan besluiten dat het werk een leerrijke periode voor mij is geweest.

Allereerst zou ik mijn promotor Commandant v/h Vlw Dr. Ir. Thibault DEBATTY en tweede lezer Professor Dr. Ir. Wim MEES willen bedanken voor de mogelijkheid die ze mij hebben gegeven en alle tijd die ze voor mij hebben vrijgemaakt om mij te begeleiden tijdens dit werk. Alsook zou ik Lt. Thomas GILON willen bedanken, die het oorspronkelijke systeem heeft ontwikkeld, om zijn hulp bij het proberen oplossen van een bug.

Ik zou graag de 169ste Promotie Polytechniek bedanken voor hun kameraadschap. Wij zijn vrienden voor het leven geworden. Samen vonden we steeds de weg of hebben we er een gemaakt! Voor de onvoorwaardelijke steun van mijn ouders en familie zou ik ook hen graag bedanken. Phara wil ik in het bijzonder bedanken. Zonder haar zou de opleiding een pak moeilijker geweest zijn. Bedankt voor je steun!

Jordy CANSSE

Inhoudsopgave

Lijst van figuren

Afkortingen

APT Advanced Persistent Threat

AUC Area Under Curve

CSOC Cyber Security Operations Centre

C2 Command and Control

DMZ DeMilitarized Zone

DNS Domain Name System

GA Genetisch Algoritme

HTTP HyperText Transfer Protocol

MASFAD Military Multi-Agent System For APT Detection

NIST National Institute of Standards and Technology

OWA Ordered Weighted Average

PHP PHP: Hypertext Preprocessor

PSO Particle Swarm Optimization

ROC Receiver Operating Characteristic

RUCD Research Unit Cyber Defense

RWS Roulette Wheel Selection

SMTP Simple Mail Transfer Protocol

TLD Top Level Domain

URL Uniform Resource Locator

TOR The Onion Router

TOS TOurnament Selection

WOWA Weighted Ordered Weighted Average

1. Inleiding

Vandaag de dag moeten organisaties zowel in de privé als in de publieke sector zich wapenen tegen de opkomende cyberdreiging. Defensie beloopt hetzelfde pad, dat blijkt uit de strategische visie voor Defensie:

> "De cyberomgeving is een essentiële omgeving voor het welslagen van de militaire operationele inzet. Het versterken van onze eigen militaire cybercapaciteit is noodzakelijk [3, p. 52]."
>
> "Een militaire cybercapaciteit draagt bij tot het verdedigen van de voor Defensie cruciale communicatie- en wapensystemen, maar moet evenzeer een bijdrage kunnen leveren aan offensieve acties binnen de expeditionaire inzet [3, p. 53]."
>
> "De cyberdimensie werd toegevoegd aan de dimensies waarin de NAVO aan collectieve defensie doet, vanwege de realiteit dat sommige cyberaanvallen eenzelfde uitwerking kunnen hebben op de territoriale integriteit van de NAVO-landen als aanvallen met conventionele wapens [3, p. 53]."

Het strategisch plan voor Defensie [3] voorziet op korte termijn bijkomende middelen en personeelsleden voor de cybercapaciteit. Deze cybercapaciteit is het *Cyber Security Operations Centre* (CSOC) en staat in voor de cyberdefensie van de communicatie- en informatiesystemen van de strijdkrachten. Eveneens voorziet de strategische visie de nodige investeringsbudgetten en bijkomend personeel voor de opbouw van een bijkomende cybercapaciteit, namelijk offensieve cyber.

De *Advanced Persistent Threat* (APT) is bij uitstek de meest geavanceerde cyberaanval. Een Advanced Persistent Threat is een gesofisticeerde, doelgerichte cyberaanval uitgevoerd over een lange periode door ontzettend bekwame aanvallers. Een bekende APT is *Sofacy*, ook gekend als APT28, *Fancy Bear* of *Tsar Team* [4]. Deze APT is actief sinds 2011 en heeft al militaire en diplomatieke organisaties aangevallen van diverse landen waaronder België, Frankrijk, Duitsland, Italië, het Verenigd Koninkrijk en de Verenigde Staten.

De Research Unit Cyber Defense (RUCD) van de Koninklijke Militaire School (KMS) is bezig met de ontwikkeling van een systeem dat APTs detecteert, genaamd MASFAD (*Military multi-Agent System For APT Detection*) [5]. Dit systeem is opgebouwd uit verschillende *agents* die elk op een andere wijze APTs detecteren. Uiteindelijk worden alle resultaten van de verschillende *agents* gebundeld zodanig dat de kans op detectie verhoogt.

De detectie is gebaseerd op de specifieke eigenschappen van een APT. Elke *agent* mikt op de detectie van een verschillende eigenschap. APTs communiceren met hun aanvallers voor

het doorgeven van informatie of het verkrijgen van nieuwe instructies. In een vorig werk is een detectie-algoritme ontwikkeld die berust op deze eigenschap. Dit vorig werk is uitgevoerd door Thomas Gilon en heeft geleid tot een werkend algoritme dat APTs detecteert aan de hand van grafen gebaseerd op de logs van een proxy [1]. Het algoritme beschikt over een waaier aan parameters waarop kan worden ingespeeld zodanig de kans op detectie van APTs gemaximaliseerd kan worden. Een eerste beperkte studie van de invloed van deze parameters op het algoritme is reeds uitgevoerd. De analist, de gebruiker van het algoritme, wordt voorzien van advies over de parameters en een combinatie van parameters die dienen als startpunt voor de detectie van APTs.

In deze masterproef is het de bedoeling om een optimale combinatie van parameters te vinden meer bepaald een combinatie van parameters die de kans op detectie van APTs maximaliseert. Dit wordt gerealiseerd door een literatuurstudie uit te voeren naar de mogelijke optimalisatiemethoden die bestaan om het detectie-algoritme te optimaliseren. Eenmaal de bruikbare optimalisatiemethoden bepaald zijn, worden deze geïmplementeerd en geïntegreerd in het detectie-algoritme. Ten slotte worden de optimalisatie-algoritmen uitgevoerd zodanig de optimale parameters bekomen worden.

De masterproef is opgebouwd uit zes onderdelen. Het eerste deel omvat deze inleiding. In het tweede deel wordt de nadruk gelegd op het voorgaande werk. Het idee achter het detectie-algoritme, de werking van het detectie-algoritme en de belangrijkste besluiten uit voorgaand werk worden besproken. Vervolgens wordt het theoretisch kader uiteengezet. Hierin worden enkele begrippen over optimalisatie uitgelegd evenals de optimalisatiemethoden die gebruikt zullen worden. In het vierde deel worden de implementatie en integratie van de optimalisatiemethoden verduidelijkt. De resultaten van de optimalisatie worden weergegeven en besproken in het daaropvolgend deel. Het zesde en laatste deel bevat de conclusie van de masterproef, de eigen bijdragen aan het detectie-algoritme en de aanbevelingen voor opvolgend werk.

2. Detectie-algoritme

2.1 Inleiding

Het detectie-algoritme ligt aan de basis van dit werk. Het algoritme is ontworpen door Thomas Gilon. Dit hoofdstuk is volledig gebaseerd op zijn masterthesis [1] en heeft als doel het idee achter het algoritme te verduidelijken, de werking van het algoritme uit te leggen en de bevindingen uit zijn werk samen te vatten. Het is niet de bedoeling om alle details over het algoritme te verklaren maar eerder om de lezer vertrouwd te maken met het algoritme en de belangrijkste resultaten uit het werk van Gilon over te brengen.

Het doel van het algoritme is om een graaf van domeinnamen te maken door de analyse van de logs van een HTTP-proxyserver. Door een correcte modellering van het netwerkverkeer uit te voeren, zou het Commando- en Controlekanaal van de APT als geïsoleerd domein in de graaf moeten verschijnen. Er wordt geijverd naar een maximale detectie van APTs met een minimaal aantal aan valse alarmen.

Het hoofdstuk is opgebouwd uit 6 delen. Deze inleiding is het eerste deel. Het tweede deel leidt enkele begrippen in die nodig zijn om het algoritme te begrijpen. In het derde deel wordt er gefocust op het model dat wordt toegepast om de logbestanden te modelleren. Het idee van het model is om de log voor te stellen als een verzameling van webpagina's. Eens het model gekend is, wordt er ingegaan op de implementatie van het algoritme. Het algoritme is geschreven in de programmeertaal Java en bestaat om praktische redenen uit drie delen: de Core, de Batch Processor en de Server. Om de interactiviteit met de analist te verhogen beschikt het algoritme over een grafische interface. Vervolgens wordt een combinatie van parameters gezocht die kunnen dienen als startpunt voor het algoritme. Eveneens wordt de invloed van elke parameter op het algoritme besproken. Het geheel wordt afgesloten met een conclusie.

2.2 Inleidende begrippen

De ***Advanced Persistent Threat*** (APT) is een veelbesproken onderwerp in de Cyberwereld, waardoor er verschillende definities circuleren. Om een duidelijk beeld te scheppen, wordt de definitie van het NIST[1] aangehaald. Het NIST beschrijft een APT als volgt [7]:

> "An adversary that possesses sophisticated levels of expertise and significant resources which allow it to create opportunities to achieve its objectives by using multiple attack vectors (e.g., cyber, physical, and deception). These objectives typically

[1] Het *National Institute of Standards and Technology* (NIST) maakt deel uit van het *U.S. Department of Commerce* en houdt zich voornamelijk bezig met het stimuleren van innovatie, het industrieel concurrentievermogen en de levenskwaliteit [6].

> include establishing and extending footholds within the information technology infrastructure of the targeted organizations for purposes of exfiltrating information, undermining or impeding critical aspects of a mission, program, or organization; or positioning itself to carry out these objectives in the future. The advanced persistent threat: (i) pursues its objectives repeatedly over an extended period of time; (ii) adapts to defenders' efforts to resist it; and (iii) is determined to maintain the level of interaction needed to execute its objectives."

Uit de definitie blijkt dat de term APT zorgvuldig is gekozen daar deze inderdaad geavanceerd, aanhoudend en bovenal een bedreiging is. De APT wordt uitgevoerd door goed georganiseerde, vakkundige aanvallers die over zowel de nodige technische als financiële middelen beschikken [8]. De aanvallers gebruiken geavanceerde technieken zoals *zero-day* aanvallen of *social engineering*. Een zero-day aanval maakt gebruik van kwetsbaarheden in computerprogramma's die publiek ongekend zijn. Bijgevolg is het onmogelijk om dit type aanval te detecteren met de traditionele detectiemethoden die worden gebruikt in antivirussoftwares of *Intrusion Detection Systems*[1]. Social engineering is een techniek welke uitgaat van de psychologische manipulatie van mensen om zo doelen te bereiken die wel of niet in het belang van het doelwit zijn [8].

Een ander kenmerk van de APT is dat deze een duidelijk doel en een specifiek doelwit heeft, wat ervoor zorgt dat de aanvallers hun technieken en tactieken aanpassen aan het doelwit. Mogelijke doelwitten zijn grote organisaties zoals Defensie, overheidsinstellingen, multinationals. De eigenlijke aanval wordt voorafgegaan door een periode van planning en voorbereiding waarin zoveel mogelijk informatie wordt verzameld over het doelwit en een afgetekend aanvalsplan wordt opgesteld met duidelijke doelen en aangepaste technieken. Als de aanval mislukt, worden de tactieken en technieken aangepast en wordt een nieuwe poging ondernomen. Het doel van de APT is om onopgemerkt het netwerk van het doelwit binnen te dringen en zo lang mogelijk verborgen te blijven. Deze periode waarin de APT onopgemerkt blijft kan een paar maanden duren en er zijn zelfs gevallen bekend waarin dit enkele jaren is [10].

Eenmaal binnengedrongen in het netwerk, probeert de APT een communicatiekanaal tot stand te brengen met een ***Command and Control server*** (C2). Dit kanaal wordt gebruikt voor het exfiltreren van informatie of het verkrijgen van nieuwe instructies. Om het kanaal tot stand te brengen kan eender welk protocol worden gebruikt dat is toegelaten door de beveiligingssystemen van de aangevallen organisatie, voorbeelden zijn HTTP, DNS en SMTP. Verschillende methodes worden gebruikt om de C2 en de gegevens die worden verzonden te verbergen, zoals het vercijferen van de gegevens, het gebruik van een *anonymity network* (bv. TOR[2]), het gebruik van sociale media.

Eveneens varieert de manier waarop APTs verbinding maken de C2. Enerzijds bestaan er periodieke APTs die, zoals de naam reeds doet vermoeden, steeds na een bepaalde periode verbinding maken. Deze APTs zijn weinig tot niet verborgen aangezien de APT eventueel

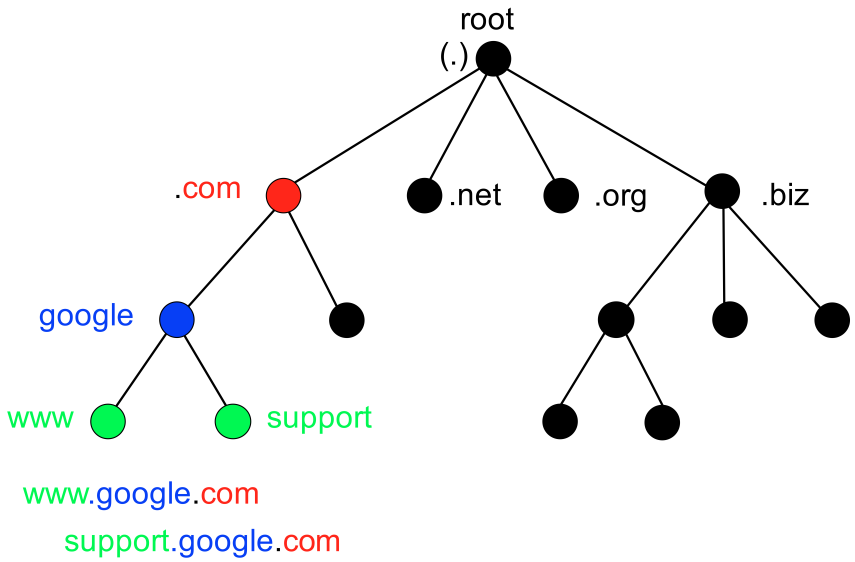
[1] Een *Intrusion Detection System* (IDS) is een software- en/of hardwaresysteem dat zelfstandig gebeurtenissen in computersystemen of netwerken controleert op zoek naar indicaties van veiligheid gerelateerde problemen [9].

[2] *The Onion Router* (TOR) heeft als doel om een internet met zoveel mogelijk privacy te realiseren door het netwerkverkeer door verschillende servers te routen en elke stap te vercijferen [11].

requests[1] verstuurt als de gebruiker niet actief is. Anderzijds, zijn er gegevensstroom gebaseerde APTs. Dit type APT maakt enkel een verbinding met de C2 als de gebruiker actief is. Deze methode zorgt ervoor dat de requests van de APT verborgen zijn tussen deze van de gebruiker.

In de meeste bedrijfsnetwerken, in ieder geval in elk cyberveiligheidsbewust bedrijf, wordt er gebruik gemaakt van een **HTTP-proxyserver**. De HTTP-proxyserver bevindt zich in de *De-Militarized Zone* (DMZ)[2] en dient als verplicht doorgangspunt voor elke request tussen een gebruiker en het internet. De HTTP-proxyserver houdt een logbestand bij van alle requests en alle acties die de server heeft uitgevoerd. Deze logbestanden worden gebruikt als startpunt van het algoritme. Het algoritme kan twee types van logbestanden behandelen, namelijk *Squid* en *JSON*.

Een **domeinnaam** is een naam die een netwerkdomein identificeert. Domeinnamen hebben een boomstructuur waarin elke knoop een tekstlabel voorstelt, behalve de *root* (wortel van de boom) die wordt voorgesteld door het label *null* (een leeg label) (Fig. 2.1). De labels het dichts bij de root zijn de *Top Level Domains* (TLDs) (bv. `.com`). Een domeinnaam is een opeenvolging van textlabels gescheiden door punten die leiden tot een bepaalde knoop in de boom (bv. `www.google.com`).



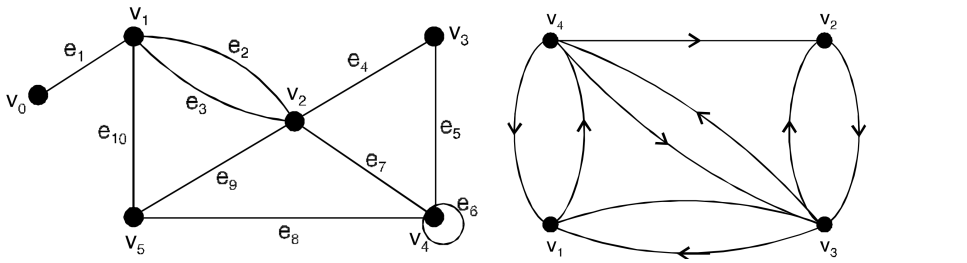
Figuur 2.1: Voorbeeld van de boomstructuur van domeinnamen [1].

Een **graaf** bestaat uit een niet-lege eindige verzameling van knopen en uit een eindige verzameling van verschillende ongeordende paren van knopen zogenaamde bogen. Verschillende types grafen zijn te onderscheiden. Een gerichte graaf is een graaf waarin elke boog een richting én een zin heeft. Een volledige graaf is een graaf die alle bogen bevat zodanig alle knopen onderling verbonden zijn. Een k-reguliere graaf is een graaf waarin elke knoop k bogen met andere knopen heeft. Aan bogen kan ook een waarde worden toegekend, welke de overeenkomst weergeeft tussen de desbetreffende knopen. Een k-NN (*k-Nearest Neighbors*) graaf is een k-reguliere graaf

[1] Een mogelijk manier waarop computers met elkaar communiceren is via het *request-response* mechanisme. Als een gebruiker een bepaalde webpagina op een server wil bekijken, verstuurt de computer een *request* (verzoek) om de webpagina door te sturen. De server beantwoordt het verzoek dan met een *response* (antwoord) die de data van de webpagina bevat.

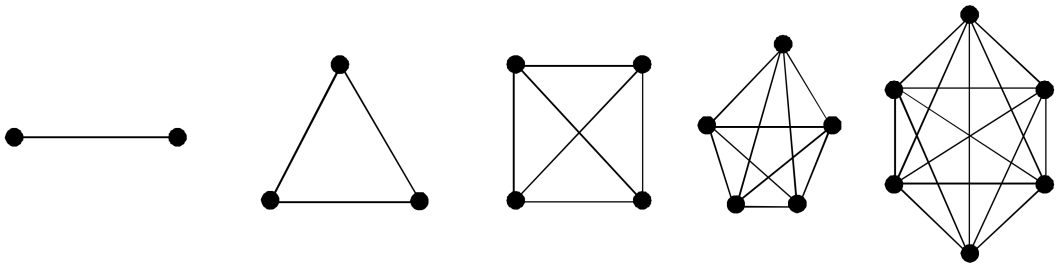
[2] Lezers die meer informatie wensen over de DMZ kunnen volgende website raadplegen `http://www.tech-faq.com/dmz.html`.

waarin elke knoop gelinkt is aan de k knopen met de grootste gelijkenis. Het algoritme maakt gebruik van gerichte grafen waaronder sommige k-NN grafen zijn.



(a) Graaf met $v_{i\in[0,5]}$, de verzameling knopen en $e_{i\in[1,10]}$, de verzameling bogen.

(b) Gerichte graaf.



(c) Volledige grafen.

Figuur 2.2: Weergaven van verschillende grafen [2].

Pruning is de actie waarbij alle bogen worden verwijderd die een lagere waarde dan een bepaalde drempel hebben, namelijk de pruningdrempel. *Clustering* is de actie welke verborgen structuren in grafen zoekt. Clustering bestaat uit drie fases:

1. bepalen van de overeekomsten tussen de verschillende knopen,
2. groeperen van de knopen a.d.h.v. deze overeenkomsten,
3. beoordelen van de bekomen oplossing.

2.3 Modellering

Het doel van het algoritme is om een graaf van domeinnamen te creëren a.d.h.v. de logs van een HTTP-proxyserver waarin de APT als geïsoleerde knoop verschijnt. Om dit te realiseren is het noodzakelijk dat het netwerkverkeer op een correcte manier wordt behandeld t.t.z. volgens een bepaald model. Uiteraard gaat dit model gepaard met enkele beperkingen die voorzien worden van de nodige maatregelen.

2.3.1 Model

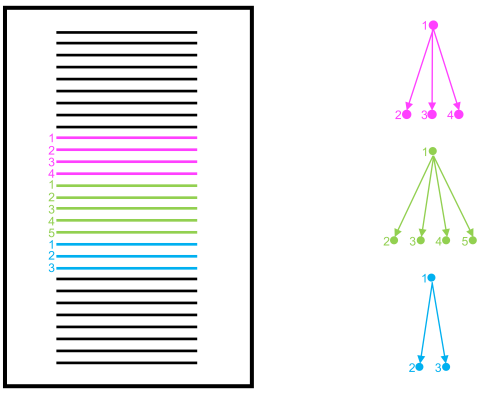
Het idee van het model is om het netwerkverkeer voor te stellen als een verzameling webpagina's. Webpagina's kunnen op verschillende manieren worden geopend, zoals het gebruik van een bladwijzer, zoekmachine of het direct ingeven van de URL in de webbrowser. Een *Uniform Resource Locator* (URL) is een *string* die naar een bron op het internet verwijst [12]. De domeinnaam is een onderdeel van de URL. Om een volledige webpagina te laden, moeten er

nog verschillende andere URLs worden geladen dan deze van de webpagina zelf. Er bestaat dus een verband tussen deze verschillende URLs. De initiële URL wordt beschouwd als ouder en de andere URLs als kinderen van deze ouder. Zonder ouder zouden deze kinderen niet bestaan, t.t.z. als de initiële URL niet wordt bezocht, worden de andere URLs evenmin bezocht. Elke keer dat dezelfde webpagina wordt geladen, moet dit verband ouder-kind worden versterkt.

Als voorbeeld wordt de homepagina van de krant De Tijd genomen. Na het ingeven van de URL `https:///www.tijd.be` in de webbrowser, worden volgende domeinen bezocht: `api.tijd.be`; `apis.google.com`; `charting.vwdservices.com`; `connect.facebook.net`; `fonts.googleapis.com`; `i.ytimg.com`; `images.tijd.be`; `img.youtube.com`; `platform.twitter.com`; `rum-collector-2.pingdom.net`; `rum-static.pingdom.net`; `s.ytimg.com`; `static.tijd.be`; `staticxx.facebook.com`; `syndication.twitter.com`; `www.google.com`; `www.youtube.com`; `yt3.ggpht.com`.

Het domein `www.tijd.be` wordt beschouwd als de ouder van alle domeinen die zonet in de lijst zijn vernoemd. De domeinen in de voorgaande lijst worden beschouwd als de kinderen van het domein `www.tijd.be` . In de graaf worden aan de knoop die het domein `www.tijd.be` voorstelt gerichte bogen toegevoegd naar alle domeinen in de voorgaande lijst. Alle bogen krijgen een waarde die het verband voorstelt tussen de respectievelijke domeinen. Elke keer dat de homepagina van De Tijd wordt bezocht, moet de waarde van dit verband verhogen.

In figuur 2.3 wordt de modellering van enkele webpagina's schematisch weergeven. Links is er een chronologische lijst te zien van alle requests. De roze graaf stelt een webpagina voor. De roze lijnen in de lijst met alle requests stellen de requests voor die nodig zijn om deze webpagina te laden. Initieel wordt de roze webpagina geladen door de roze request met het nummer 1. Om de webpagina volledig te laden, worden vervolgens de roze requests met het nummer 2, 3 en 4 geladen. Het domein, dat overeenstemt met de eerste roze request, wordt de ouder genoemd en wordt voorgesteld als de knoop met nummer 1. De domeinen, gelinkt aan de roze requests 2, 3 en 4, worden voorgesteld door de knopen met nummers 2, 3 en 4. Deze domeinen worden de kinderen van het eerste domein genoemd. Aangezien de domeinen 2, 3 en 4 het gevolg zijn van domein 1, worden er gerichte bogen toegevoegd van het eerste domein naar domein 2,3 en 4. Aan elke boog wordt eveneens een waarde toegekend die het verband tussen de respectievelijke domeinen voorstelt (het verband ouder-kind).

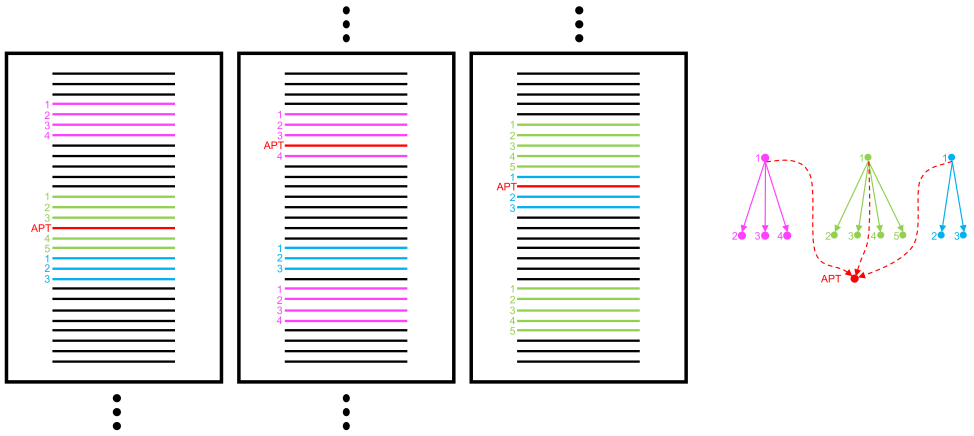


Figuur 2.3: Schematische weergave van verschillende webpaginas in het model [1].

De verschillende gebruikers aanwezig in de HTTP-proxylog bezoeken diverse webpagina's. Elke

webpagina wordt geladen door de opeenvolging van het domein van de ouder en verschillende domeinen van de kinderen. Het algoritme berekent eerst de verzameling webpagina's voor elke gebruiker apart om daarna de bekomen resultaten te combineren om zo de gehele HTTP-proxylog te modelleren. Als verschillende gebruikers eenzelfde webpagina bezoeken, moet het overeenkomstige ouder-kindverband eveneens worden versterkt.

Het doel is nog steeds om APTs te detecteren a.d.h.v. de verbindingen die de APT met de C2 maakt. Deze verbinding wordt aanzien als een kind van verschillende ouders. Namelijk elke keer de APT verbinding maakt de C2, wordt deze verbinding aanschouwd als een kind van een bepaalde webpagina die de gebruiker heeft bezocht. Bijgevolg wordt de C2 van een APT beschouwd als een 'illegaal', sporadisch kind dat zwak gelinkt is aan vele ouders (Fig. 2.4).



Figuur 2.4: Schematische weergave van een APT in het model [1].

2.3.2 Beschrijving van de graaf

Zoals reeds besproken, bestaat een graaf uit knopen en bogen. In het model zijn de knopen domeinnamen en stellen de bogen de relatie ouder-kind voor. Het algoritme maakt gebruik van gerichte grafen waarin de richting van ouder naar kind is. Idealiter is een APT een geïsoleerde knoop dus een knoop zonder enige boog.

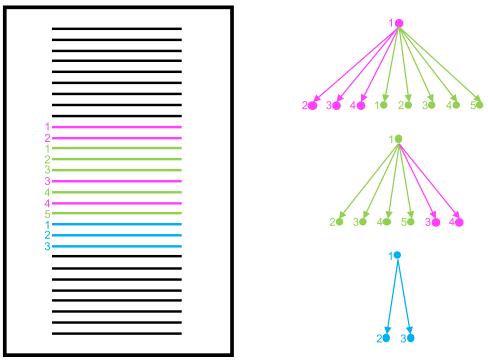
Voor elke gebruiker in de HTTP-proxylog wordt een graaf gemaakt. Idealiter, bestaat deze graaf uit verschillende structuren waarin elke structuur een webpagina voorstelt. Als een gebruiker webpagina's meerdere malen bezoekt, moet alle overeenkomstige verbanden verstrekt worden. Om de gehele log te modelleren, worden vervolgens alle gebruikers samengevoegd. Indien verschillende gebruikers dezelfde webpagina hebben bezocht, moeten opnieuw de overeenkomstige verbanden worden verstrekt. Zo leidt de samenvoeging van de verschillende gebruikers tot een verbetering van de modellering van de webpagina's. Daar een APT gelinkt is aan verschillende ouders, worden de verbanden niet versterkt door herhaling. Door de fusie van de grafen van de verschillende gebruikers, worden de verbanden met de APT evenmin versterkt als er verondersteld wordt dat slechts weinig gebruikers besmet zijn met de APT. Aangezien de APT geen sterke verbanden heeft met een ouder, is het mogelijk de APT te isoleren door alle zwakke verbanden te verbreken.

Het verband tussen de knopen moet zorgvuldig worden gekozen. Een eerste logische aanpak is een tijdsgebonden verband. Hoe dichter de domeinen in tijd (wanneer ze bezocht zijn), hoe

groter het verband. Een andere aanpak is een naamsgebonden verband. Hoe meer labels de domeinnamen gemeenschappelijk hebben, hoe groter het verband tussen beiden.

2.3.3 Beperkingen

Niet altijd worden webpagina's één na één bezocht. Soms worden verschillende webpagina's gelijktijdig geladen, wat resulteert in een foutief model van de verschillende webpagina's (Fig. 2.5). Om alle echte kinderen van een bepaalde ouder te vinden, is het noodzakelijk om in het vervolg van de log een voldoende lange tijdspanne te analyseren. Door meerdere malen een webpagina te bezoeken worden de correcte verbanden versterkt, terwijl de foutieve verbanden zwak blijven waardoor deze automatisch worden uitgesloten. De naamsgebonden aanpak biedt eveneens een uitweg. Het onderbreken van het laden van een webpagina zorgt ervoor dat niet alle kinderen geladen worden. Ook hier bieden herhaling en de naamsgebonden aanpak een uitweg.



Figuur 2.5: Schematische weergave van het gelijktijdig laden van verschillende webpagina's [1].

Sommige achtergrondprocessen genereren netwerkverkeer onafhankelijk van het gedrag van de gebruiker. In tegenstelling tot het netwerkverkeer van de APT, is dit netwerkverkeer wel toegestaan. Het netwerkverkeer van achtergrondprocessen heeft gelijkaardige kenmerken aan dat van APTs. Namelijk zwak gelinkt aan verschillende ouders en wordt dus aanschouwd als een 'illegaal', sporadisch kind. Om valse alarmen door dit netwerkverkeer te voorkomen, kan er worden gebruik gemaakt van *whitelisting*.

Veel webpagina's zijn dynamisch opgebouwd m.a.w. de kinderen variëren in de tijd. Als de kinderen snel variëren, is er weinig tijd om de verbanden te versterken door herhaling. Een andere aanpak dan de tijdsgebonden aanpak, namelijk de naamsgebonden aanpak, lost dit probleem op.

Louter door een log te bekijken is het onmogelijk om te achterhalen welke domeinen als ouder en welke als kind moeten worden beschouwd. Praktisch gezien, wordt elk domein als ouder beschouwd en worden verbanden naar alle mogelijke kinderen berekend. Om evidente redenen kunnen slechts een bepaald aantal verbanden worden berekend voor elke ouder.

2.4 Implementatie

In het ideale geval wordt de HTTP-proxylog gemodelleerd door een volledige graaf. Voor het gebruik van een volledige graaf is een grote opslagcapaciteit nodig. Daarom wordt er gebruik gemaakt van een k-NN graaf zodat per knoop enkel de k grootste verbanden worden opgeslagen. De waarde van k is van cruciaal belang omdat die de kwaliteit van het model bepaalt. Doordat het berekenen van k-NN grafen een computationeel intensieve taak is, zijn er verschillende algoritmes ontwikkeld die de ideale k-NN grafen benaderen en de rekentijd verminderen. In het algoritme wordt het *NN-Descent* algoritme gebruikt om de k-NN grafen te berekenen. Afgezien van het feit dat het NN-Descent algoritme de rekentijd aanzienlijk vermindert, blijft de berekening van de grafen een intensieve taak. Om goede resultaten te bekomen met het NN-Descent algoritme is een minimale waarde van 10 voor k noodzakelijk. Indien k groter is dan 50 zijn meer dan 90% van de buren correct.

De bedoeling is dat het algoritme configureerbaar en interactief is. Informatie mag niet op voorhand uit de data worden verwijderd tenzij de analist hier zelf verantwoordelijk voor is. Om de interactiviteit te kunnen garanderen, worden er op voorhand zoveel mogelijk berekeningen uitgevoerd met een minimum aan keuzes die gemaakt moeten worden. Om aan al deze eisen te voldoen, bestaat het algoritme uit drie onderdelen: de Core, de Batch Processor en de Server.

De Core definieert Java-objecten en Java-methoden die door beide andere onderdelen worden gebruikt. De Batch Processor is verantwoordelijk voor de berekening van de grafen voor elk verband en elke gebruiker. Doordat dit proces enige tijd vergt, is het de bedoeling dat dit proces slechts éénmaal wordt doorlopen. De interactiviteit wordt verzorgd door de Server die alle grafen gemaakt door de Batch Processor verwerkt om zo de APT te kunnen detecteren. Ten slotte, is het algoritme voorzien van een grafische gebruikersinterface die de interactiviteit met de analist bevordert.

2.4.1 Core

De Core definieert Java-objecten en Java-methoden die door de Batch Processor en Server worden gebruikt. De verbanden zijn eveneens gedefinieerd in de Core. De tijdsgebonden en naamsgebonden verbanden worden in de volgende alinea's verder uitgelegd. De opbouw van het algoritme maakt het eenvoudig om nieuwe verbanden toe te voegen en te gebruiken.

Tijdsgebonden verband

Het tijdsgebonden verband wordt berekend aan de hand van het tijdsverschil tussen twee requests in de HTTP-proxylog. Het verband wordt gedefinieerd als volgt:

$$\mu_{\Delta t} = \frac{1}{1 + |\Delta t|} \tag{2.1}$$

waarin Δt het tijdsverschil voorstelt tussen de twee *timestamps* van de requests.

Het verband maakt geen onderscheid tussen voorgaande of komende requests. Dat maakt het verband symmetrisch. Er kan niet echt van kinderen worden gesproken maar eerder van buren. Het is aan de Batch Processor om de kinderen te selecteren uit alle buren. Het verband kan variëren tussen 0 ($\mu_{\Delta t} = \infty$) en 1 ($\mu_{\Delta t} = 0$) en is dus genormaliseerd.

De nauwkeurigheid van het tijdsgebonden verband hangt af van de nauwkeurigheid van de timestamp in de log. Meestal is de nauwkeurigheid van de timestamp gelijk aan 1 seconde. Doordat de timestamp een bepaalde nauwkeurigheid heeft, is het tijdsgebonden verband gekwantiseerd.

Naamsgebonden verband

Het naamsgebonden verband wordt berekend aan de hand van de twee domeinnamen van de requests. Het is gedefinieerd op deze manier:

$$\mu_{Dom} = \frac{\beta}{\beta_{tot}} \tag{2.2}$$

waarin β het aantal gemeenschappelijke labels is tussen de twee domeinen vertrekkende vanaf de Top Level Domain (TLD) en exclusief de TLD; en β_{tot} het maximum van het aantal labels waaruit elk van de twee domeinen bestaat en opnieuw exclusief de TLD.

Het verband is opnieuw symmetrisch en genormaliseerd. Het neemt de waarde 0 aan als er geen enkel label gemeenschappelijk is en de waarde 1 als alle labels gelijk zijn. Eveneens is het verband gekwantiseerd.

2.4.2 Batch Processor

Het doel van de Batch Processor is om de interactie tussen het algoritme en de analist te verhogen door een maximum aan gegevens te bereken en zo min mogelijk parameters te fixeren op voorhand. De bedoeling van het algoritme is om een HTTP-proxylog te modelleren aan de hand van één graaf waarin de verbanden de relatie ouder-kind voorstellen. Verschillende verbanden zijn reeds gedefinieerd. Om de analist de keuze te laten hoe deze verbanden samen worden gevoegd, wordt voor elk verband een graaf berekend. Eveneens is het evident dat de analist slechts bepaalde gebruikers wil selecteren om te onderzoeken. Aldus wordt voor elke gebruiker een graaf berekend. De uiteindelijke graaf wordt bekomen door samenvoeging van de grafen van de domeinen per gebruiker en per verband.

Aangezien er een graaf per gebruiker en per verband wordt berekend, worden k-NN grafen gebruikt in plaats van volledige grafen om zo geheugen en rekentijd te besparen. De waarde van k moet dus op voorhand worden gekozen. Zoals reeds eerder vermeld, is de keuze van k van cruciaal belang voor de nauwkeurigheid van het model.

De Batch Processor gebruikt de HTTP-proxylog als startpunt. Eerst wordt de HTTP-proxylog geparsed en worden de requests gescheiden per gebruiker. Vervolgens worden de k-NN grafen van de requests per gebruiker en per verband berekend. Het is de bedoeling om een gerichte graaf te bekomen t.t.z. een graaf met bogen gericht van ouders naar kinderen. Aangezien de verbanden symmetrisch zijn, worden voor elke request (knoop) de buren verwijderd welke deze request voorafgaan. Desondanks het model uitgaat van de relatie ouder-kind, is het mogelijk voor de analist om deze actie niet uit te voeren zodanig er geen kinderen worden geselecteerd uit de buren. Als de actie wordt uitgevoerd, zijn niet alle grafen nog noodzakelijk k-NN grafen. Het aantal buren is kleiner dan of gelijk aan k.

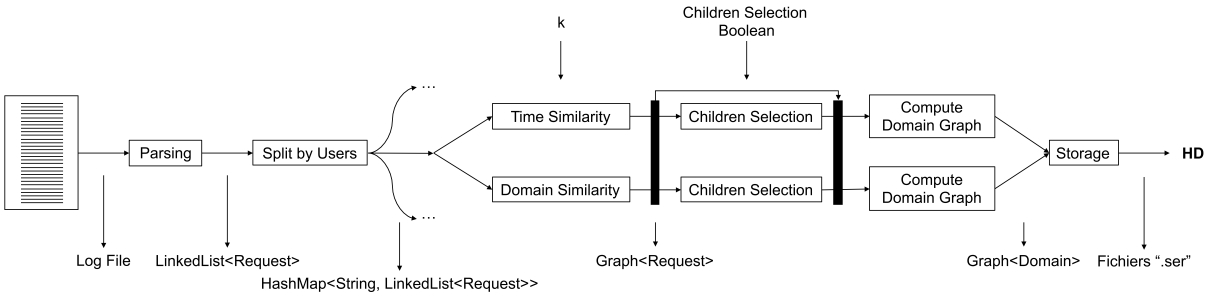
Voorlopig zijn enkel de grafen van de requests per gebruiker en per verband berekend. Om de grafen van de domeinen per gebruiker en per verband te berekenen, moet eerst het verband

tussen twee domeinen gedefinieerd worden. Het verband tussen twee domeinen is de som van de verbanden tussen de requests die toebehoren aan de domeinen in kwestie. Na het bekomen van de grafen van de domeinen, kan het aantal buren zowel kleiner, groter of gelijk aan k zijn.

Uiteindelijk worden alle bekomen resultaten opgeslagen. Voor elke gebruiker wordt er een bestand gemaakt met daarin zowel de graaf van de domeinen voor het tijdsgebonden verband als voor het naamsgebonden verband. Ook wordt er een lijst met alle gebruikers en alle subnets opgeslagen. Ten slotte wordt de initiële waarde van k ook opgeslagen.

De Batch Processor kan als volgt worden samengevat (Fig. 2.6):

1. parsing;
2. scheiden van de requests per gebruiker;
3. berekenen van de k-NN grafen van de requests voor elke gebruiker en elk verband;
4. selecteren van de kinderen uit de buren (optioneel);
5. berekenen van de grafen van de domeinen voor elke gebruiker en elk verband;
6. opslaan van gegevens (grafen per gebruiker, lijst van gebruikers en subnets, de waarde van k).



Figuur 2.6: Schema van de werking van de Batch Processor [1].

2.4.3 Server

De server verzorgt de interactiviteit tussen het algoritme en de analist en is verantwoordelijk voor de samenvoeging van de grafen bekomen door de Batch Processor en voor de verwerking hiervan. Het doel van de server is om deze bewerkingen dusdanig uit te voeren dat de domeinen die zich gedragen als C2 van een APT worden geïsoleerd in de finale graaf. Als vertrekpunt heeft de server de grafen die werden bekomen door de Batch Processor en parameters die door de analist worden voorzien. Op het einde maakt de Server een klassement van de domeinen dat het mogelijk maakt de domeinen te identificeren die zich gedragen als C2 van een APT.

De Server begint met het laden van alle grafen van elke gebruiker en elk verband. Zoals reeds vermeld, zijn deze grafen geen k-NN grafen meer door de selectie van de kinderen uit de buren en door de samenvoeging van requests in domeinen. De analist kan kiezen op welke manier de verbanden worden samengevoegd en welke gebruikers hij wil analyseren. Voor elke gebruiker wordt er eerst één graaf van domeinen gemaakt waarin de verbanden de intensiteit van de relatie ouder-kind voorstellen. Deze graaf wordt bekomen door een gewogen gemiddelde te nemen van

het tijdsgebonden en het naamsgebonden verband waarin de gewichten door de analist worden gekozen. Nadien worden de grafen van de geselecteerde gebruikers samengevoegd.

Om een maximaal aantal domeinen te isoleren die zich gedragen als C2 van een APT zijn nog enkele bewerkingen noodzakelijk. De C2 gedraagt zich als een 'illegaal', sporadisch kind dat zwak gelinkt is aan vele ouders. Door pruning toe te passen worden deze zwakke verbanden verwijderd in de graaf. De pruningdrempel wordt bepaald door de analist en kan zowel absoluut als gestandaardiseerd worden gegeven. Het gebruik van de z-score laat toe de pruningdrempel aan te passen aan de verdeling van de verbanden. Bijvoorbeeld, een z-score gelijk aan nul, verwijdert alle verbanden waarvan de waarde kleiner dan of gelijk is aan het gemiddelde. De z-score wordt als volgt gedefinieerd:

$$z = \frac{x - \mu}{\sigma} \tag{2.3}$$

met x, de absolute waarde; μ, het gemiddelde en σ, de standaardafwijking.

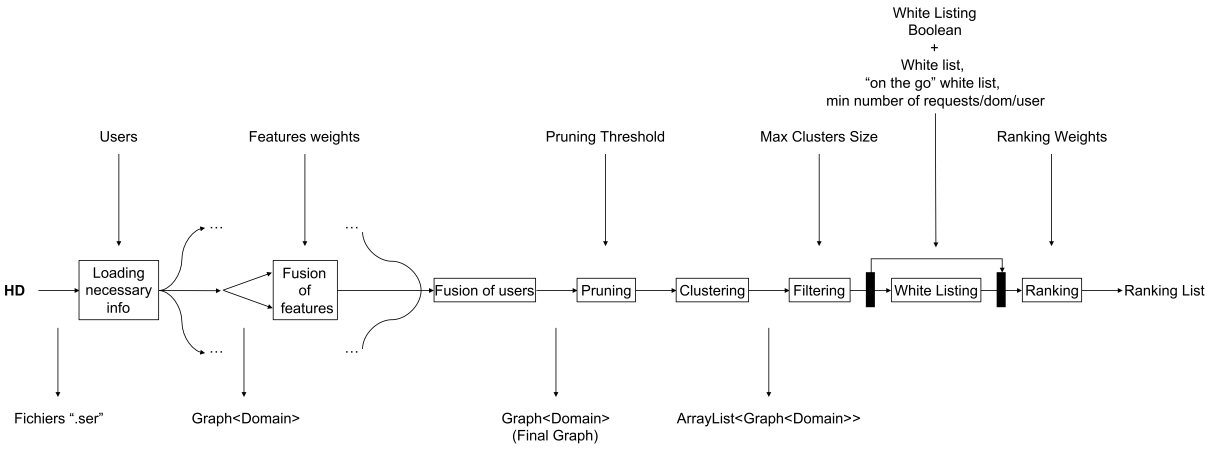
Om de geïsoleerde domeinen duidelijk te laten verschijnen wordt vervolgens clustering toegepast. Indien de C2 van de APT nog enkele verbanden heeft, zal deze een kleine cluster vormen. Door nu alle clusters te verwijderen waarvoor de clustergrootte groter is dan een bepaalde drempel, worden de kleine clusters beter zichtbaar. Deze stap heet filtering. Opnieuw kiest de analist de drempel en dit kan zowel absoluut als gestandaardiseerd.

Zoals reeds vermeld is de gebruiker niet verantwoordelijk voor al het netwerkverkeer maar creëren achtergrondprocessen eveneens netwerkverkeer. Aangezien dit netwerkverkeer gelijkaardige eigenschappen als deze van de C2 van een APT bezit, wordt er whitelisting toegepast. De analist kan ervoor kiezen om whitelisting toe te passen met een lijst van domeinen voorzien door het algoritme maar kan tevens zelf een lijst met domeinen geven. Whitelisting zorgt ervoor dat alle domeinen die voorkomen op de lijst verdwijnen uit de graaf. Eveneens heeft de analist de keuze om een minimaal aantal request per gebruiker en per domein in te stellen.

Ten slotte wordt er aan de hand van de finale graaf een klassement opgesteld van alle domeinen. Het klassement is gebaseerd op drie scores. Namelijk de som van alle verbanden gericht naar ouders, de som van alle verbanden gericht naar kinderen en het aantal requests naar het desbetreffende domein. Een gewogen gemiddelde van deze scores wordt genomen om een klassement te bekomen waarin de gewichten opnieuw door de analist worden bepaald. In de finale graaf zou de C2 maximaal geïsoleerd moeten zijn wat betekent dat de C2 een minimum aan verbanden naar zowel ouders als kinderen heeft. Daarenboven heeft de APT als doel om discreet te blijven dus zijn het aantal requests naar de C2 beperkt. De domeinen worden gerangschikt volgens waarden van klein naar groot wat ervoor zorgt dat verdachtste domeinen bovenaan de lijst terug te vinden zijn.

De werking van de Server wordt samengevat als volgt (Fig. 2.7):

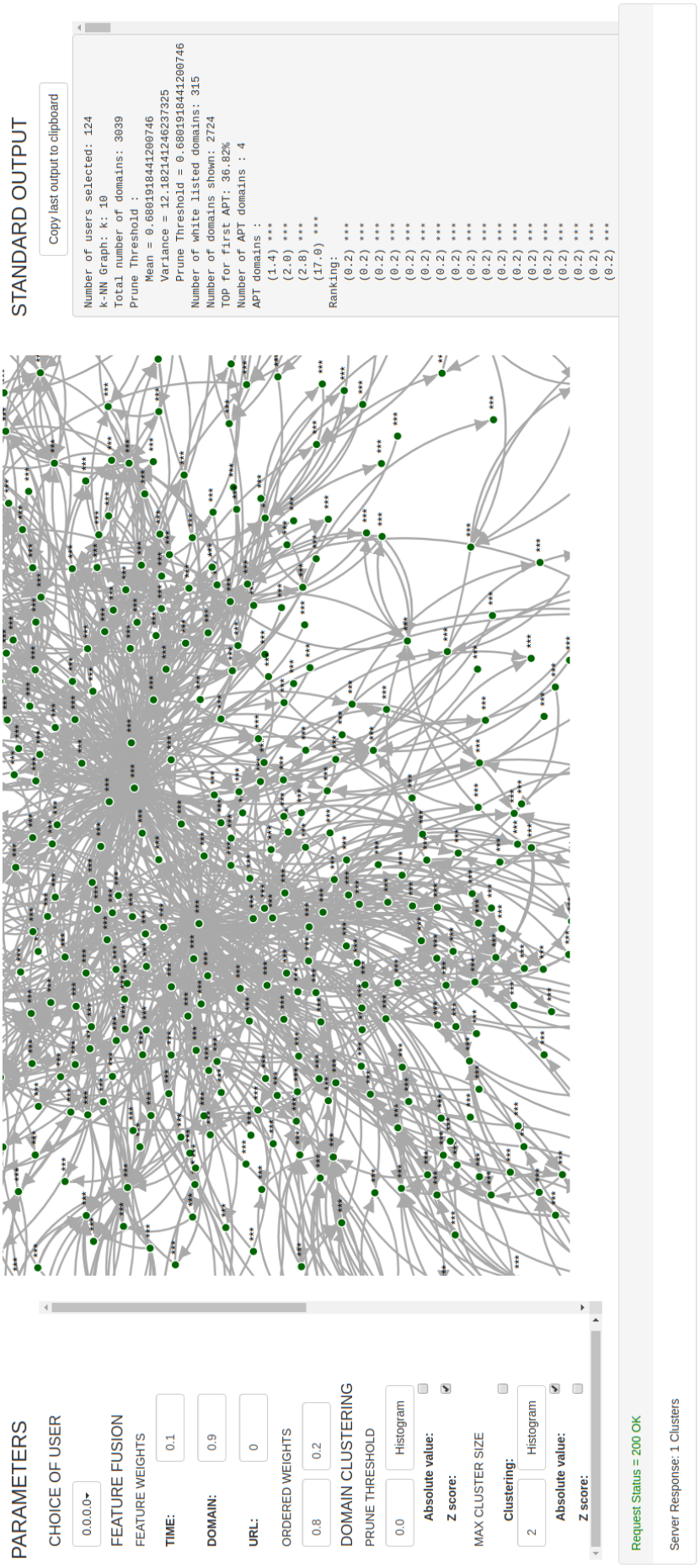
1. laden van de grafen per gebruiker en per verband, de gebruikers, de subnets en de waarde van k;
2. samenvoegen van de verbanden;
3. samenvoegen van de gebruikers;
4. pruning;
5. clustering;
6. filtering;
7. whitelisting (optioneel);
8. klassement.



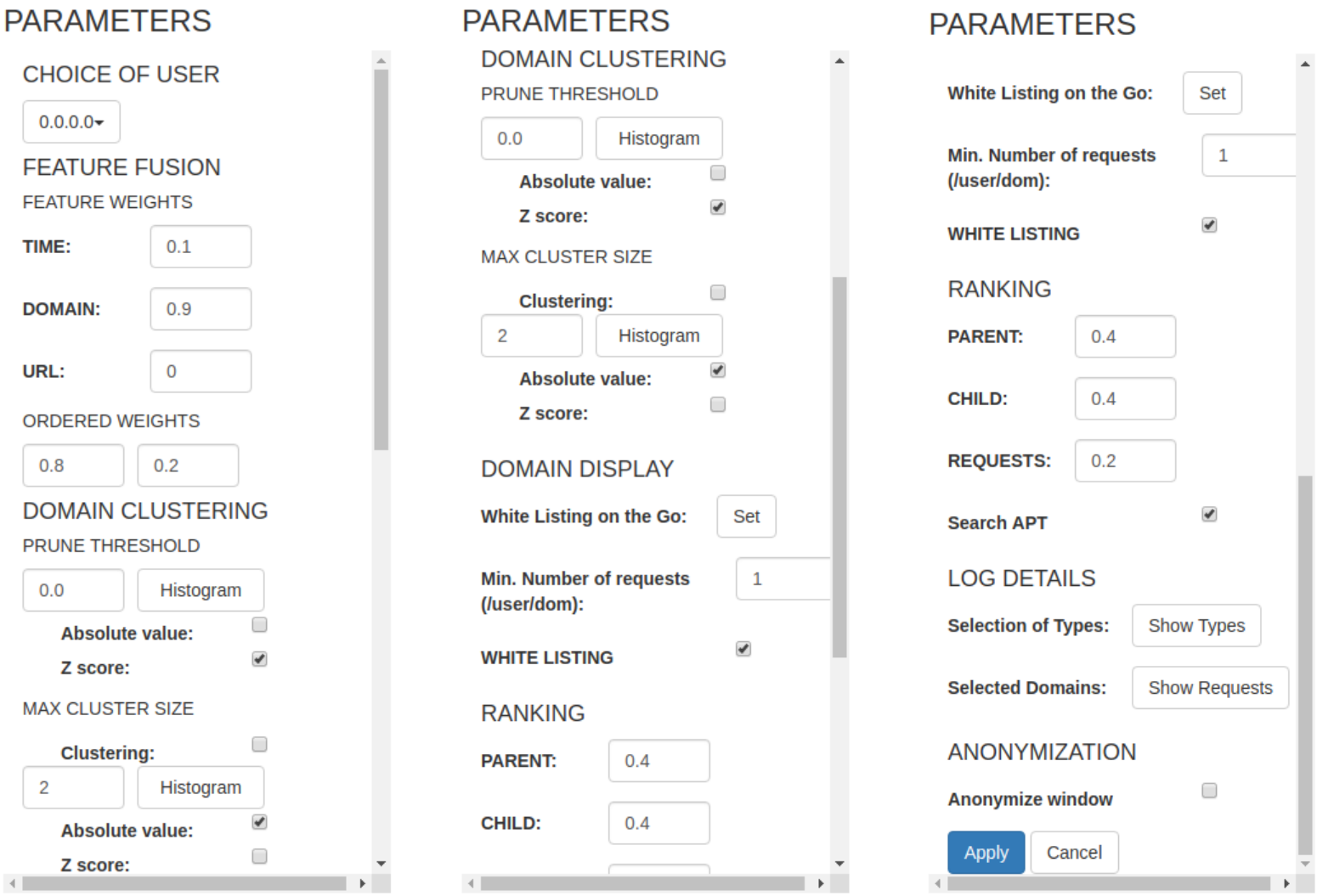
Figuur 2.7: Schema van de werking van de Server [1].

2.4.4 Grafische interface

Het algoritme beschikt over een grafische interface die zorgt voor de interactiviteit met de analist. De grafische interface wordt weergegeven in figuur 2.8. Alle mogelijke parameters die de analist kan instellen zijn weergegeven in figuur 2.9. Eens de parameters zijn ingesteld, kan de analist het algoritme starten door op de knop *Apply* te duwen. Na enige tijd wachten verschijnt het resultaat. In het midden komt de finale graaf met alle domeinen en verbanden. Door te klikken op een verband verschijnt de waarde van het verband. Het is mogelijk om alle requests van een domein te bekijken door op het domein te klikken. Aan de rechterzijde verschijnt de output van het algoritme. In de output staan het klassement en enkele interessante waarden zoals het aantal gebruikers, het aantal domeinen, de gemiddelde waarde van de verbanden.



Figuur 2.8: Grafische interface (Resultaten zijn anoniem gemaakt).



(a) (b) (c)

Figuur 2.9: Instellingen voor de parameters in de grafische interface.

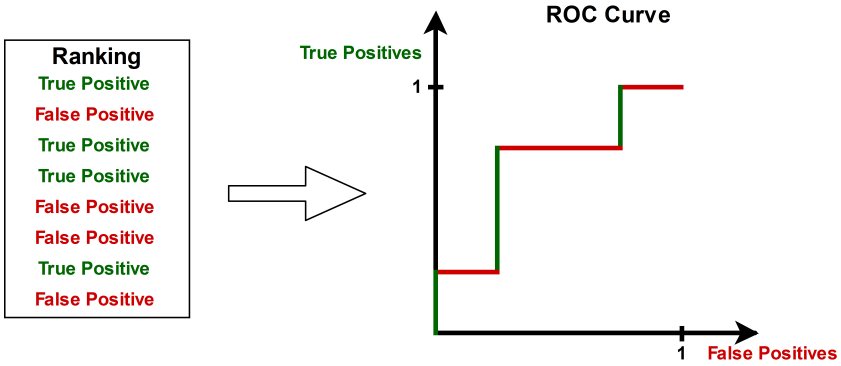
2.5 Resultaten en bespreking

Nu het model en de implementatie van het algoritme gekend is, is het noodzakelijk om een goed startpunt voor de combinatie van parameters te vinden die de analist op weg helpt om APTs te detecteren. Eerst wordt uitgelegd hoe het algoritme geëvalueerd kan worden. Vervolgens wordt de testomgeving gedetailleerd evenals de gebruikte methode om een goede combinatie van parameters te vinden. Daarna worden de parameters gepresenteerd en wordt de invloed van de parameters op het algoritme besproken. Ten slotte worden de parameters gevalideerd in een andere testomgeving.

2.5.1 Evaluatiecriterium

Om het algoritme te kunnen testen is er nood aan een evaluatiecriterium. Het algoritme wordt geëvalueerd gebruikmakende van de *Receiver Operating Characteristic* (ROC) en de *Area Under the Curve* (AUC). De ROC wordt veelvuldig gebruikt om detectiemethoden te evalueren. De ROC van een detectiemethode is een kromme die de kans op detectie voorstelt in functie van de kans op vals alarm. Het is evident dat een zo groot mogelijke kans op detectie is gewenst voor een zo klein mogelijke kans op vals alarm.

De kansen op detectie en op vals alarm worden berekend aan de hand van het klassement van de domeinen. Dit wordt geïllustreerd in figuur 2.10. Voor een bepaalde plaats in het klassement is het mogelijk om beide kansen te schatten. De kans op detectie is gelijk aan de verhouding tussen het aantal gedetecteerde C2-domeinen voor de desbetreffende plaats zelf en de plaatsen vooraf, en het totaal aantal aanwezige C2-domeinen in de log. De kans op vals alarm is gelijk aan de verhouding tussen het aantal domeinen voor de desbetreffende plaats zelf en de plaatsen vooraf exclusief het aantal C2-domeinen, en het totaal aantal domeinen in de log exclusief de C2-domeinen.



Figuur 2.10: Berekening van de ROC curve aan de hand van het klassement.

Er dient opgemerkt te worden dat er steeds wordt gesproken over kansen hoewel het over een schatting van de kans gaat. Om de domeinen te kunnen classificeren is het eveneens noodzakelijk a priori te weten of het domein al dan niet een C2 is. Aangezien er gewerkt wordt met reële data is dit niets steeds het geval. Voor de testen wordt er verondersteld dat de logs niet besmet zijn en dat er enkel APTs aanwezig zijn die doelbewust zijn toegevoegd aan de log.

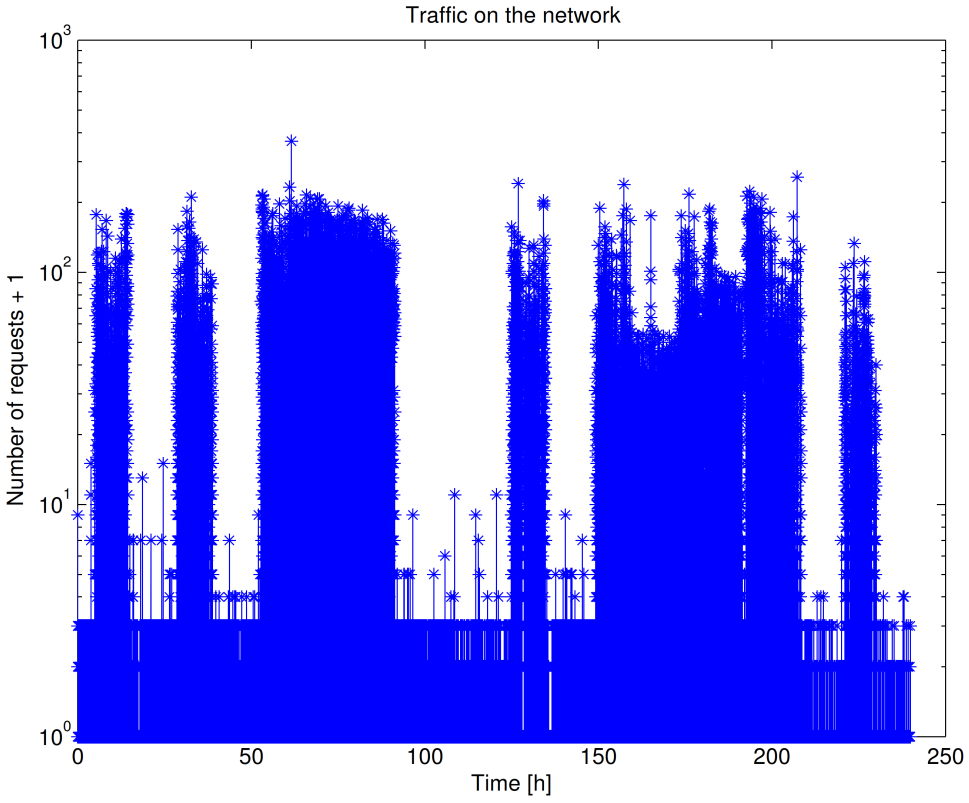
De AUC stelt de oppervlakte onder de ROC-kromme voor en kan variëren tussen 0 en 1. De AUC wordt gebruikt om verschillende ROC-krommes te vergelijken. De AUC meet het on-

derscheidingsvermogen van de detectiemethode. Een AUC gelijk aan 1 komt overeen met een perfecte onderscheiding daar staat tegenover dat een AUC gelijk aan 0,5 overeenstemt met geen enkele vorm van onderscheiding. Voor de evaluatie van het algoritme is het wenselijk een zo groot mogelijk AUC te bekomen of anders gezegd dat de ROC-curve zich zo dicht mogelijk bij de linkerbovenhoek van de grafiek bevindt.

2.5.2 Testomgeving en methode

Omgeving

Voor de testen worden er met reële logbestanden gewerkt van een grote organisatie. Een willekeurig subnet wordt gekozen maar om praktische redenen wordt toch de grootte van het subnet in rekening gebracht. Het gekozen subnet telt 26 gebruikers. Eveneens moet een bepaalde tijdspanne worden gekozen. Een willekeurige periode van 10 opeenvolgende dagen wordt gekozen. De periode start op een woensdag en loopt tot en met de vrijdag van de volgende week. In het totaal telt het bestand 721 921 requests en 4310 domeinen. Het uiteindelijke netwerkverkeer wordt weergegeven in figuur 2.11.



Figuur 2.11: Netwerkverkeer van het gekozen subnet (Resolutie: 1s) [1].

Infecties

Om het algoritme te kunnen testen is het noodzakelijk dat de C2-domeinen op voorhand gekend zijn. Er wordt verondersteld dat het netwerkverkeer niet besmet is. De APTs worden opzettelijk toegevoegd aan het netwerkverkeer. Zowel de periodieke als de gegevensstroom gebaseerde APT worden gebruikt om het algoritme te testen. Alhoewel het algoritme niet specifiek is

ontworpen voor de detectie van periodieke APTs, kan het deze toch detecteren. Er bestaan reeds andere technieken voor de detectie van periodieke APTs zoals de frequentiële analyse van logs. Voor beide soorten APTs worden er twee extreme gevallen gemaakt, een discrete APT en een agressieve APT. In totaal worden er dus 4 gebruikers besmet met een APT. Er kan worden aangetoond dat de besmetting van een gebruiker met een agressieve APT overeenstemt met de besmetting van verschillende gebruikers met eenzelfde discrete APT. Indien het algoritme efficiënt de extreme gevallen kan detecteren voor zowel de periodieke als de gegevensstroom gebaseerde APTs, zou het ook in staat moeten zijn om alle andere gevallen tussen deze extremen te detecteren.

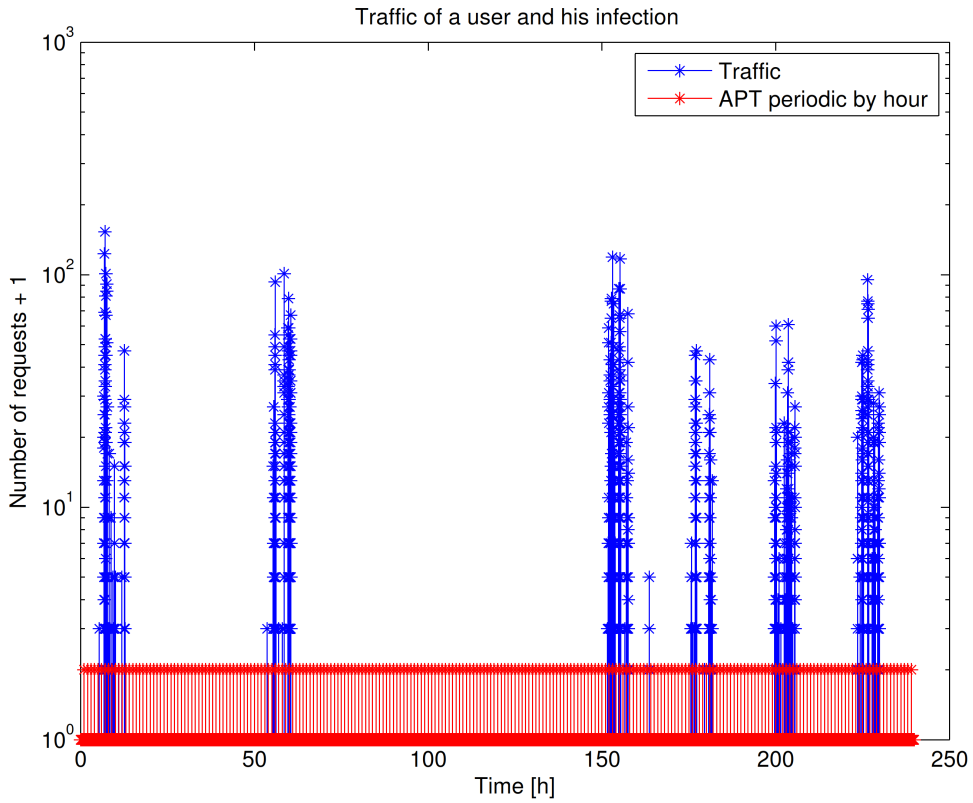
De eerste twee infecties zijn periodieke APTs. De eerste infectie heeft een periode van 1 uur, wat de APT agressief maakt. Deze eerste APT maakt in totaal 239 verbindingen met de C2 en is dus verantwoordelijk voor 239 requests in de log. Het netwerkverkeer van de geïnfecteerde gebruiker samen met de infectie wordt weergegeven in figuur 2.12. De tweede infectie is discreter en heeft een periode van 12 uur zodanig er in totaal 19 verbindingen met de C2 zijn gemaakt. Figuur 2.13 stelt het netwerkverkeer voor van de gebruiker besmet met de tweede infectie.

De derde en vierde infectie zijn beide gegevensstroom gebaseerde APTs. Voor deze is het noodzakelijk een burst te definiëren zodanig de APT weet wanneer het toegestaan is om een verbinding te maken met de C2. Een burst is een snelle opeenvolging van requests gedurende een bepaalde periode waaruit kan geconcludeerd worden dat de gebruiker actief is. Voor de derde infectie bestaat de burst uit requests die maximaal 2 seconden gespreid zijn en dit gedurende minimaal 10 seconden. De APT wordt in het midden van de burst geïnjecteerd dus na 5 seconden. Het aantal verbindingen met de APT is beperkt tot drie per dag en er moet minimaal drie uur tussen twee opeenvolgende verbindingen zijn. Al dit leidt tot 13 verbindingen door de derde APT. Deze verbindingen samen met het netwerkverkeer van de geïnfecteerde gebruiker worden weergegeven in figuur 2.14.

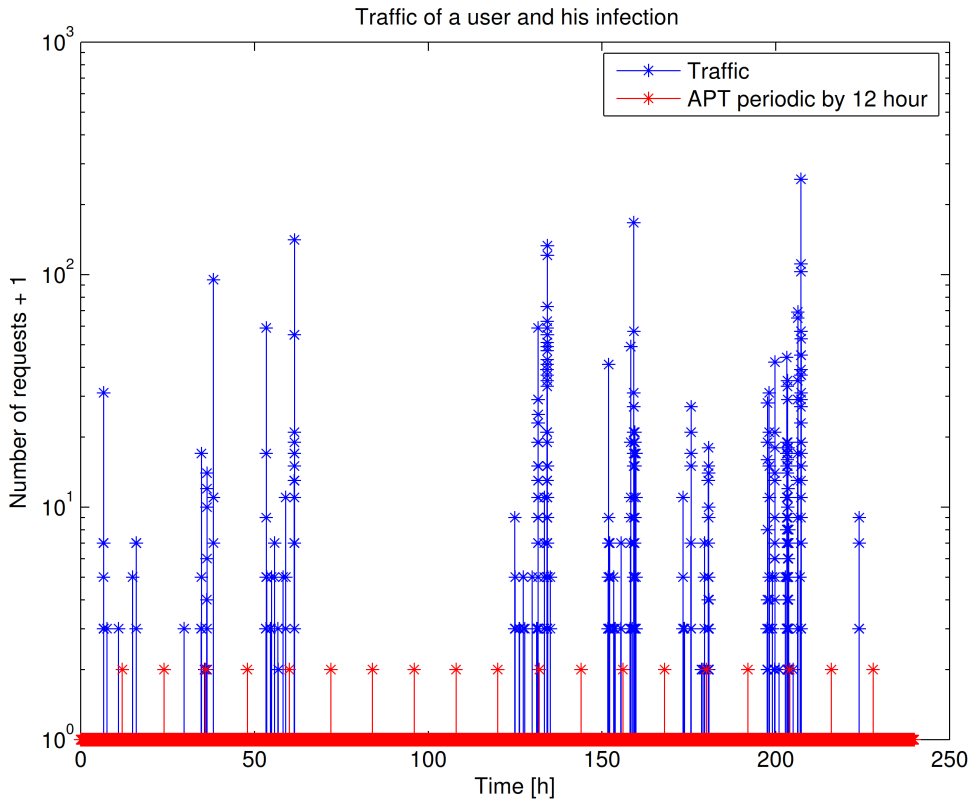
De vierde infectie is agressiever dan de derde maar de definitie van de burst is strenger. Voor de vierde infectie bestaat een burst uit requests die maximaal 1 seconde in tijd verschillen en opnieuw gedurende minimaal 10 seconden. Het maximum aantal infecties per dag loopt op tot 10 en de minimale afstand tussen twee infecties is gelijk aan 1 uur. De vierde APT maakt uiteindelijk 37 verbindingen met de C2. Dit wordt weergegeven in figuur 2.15.

Methode

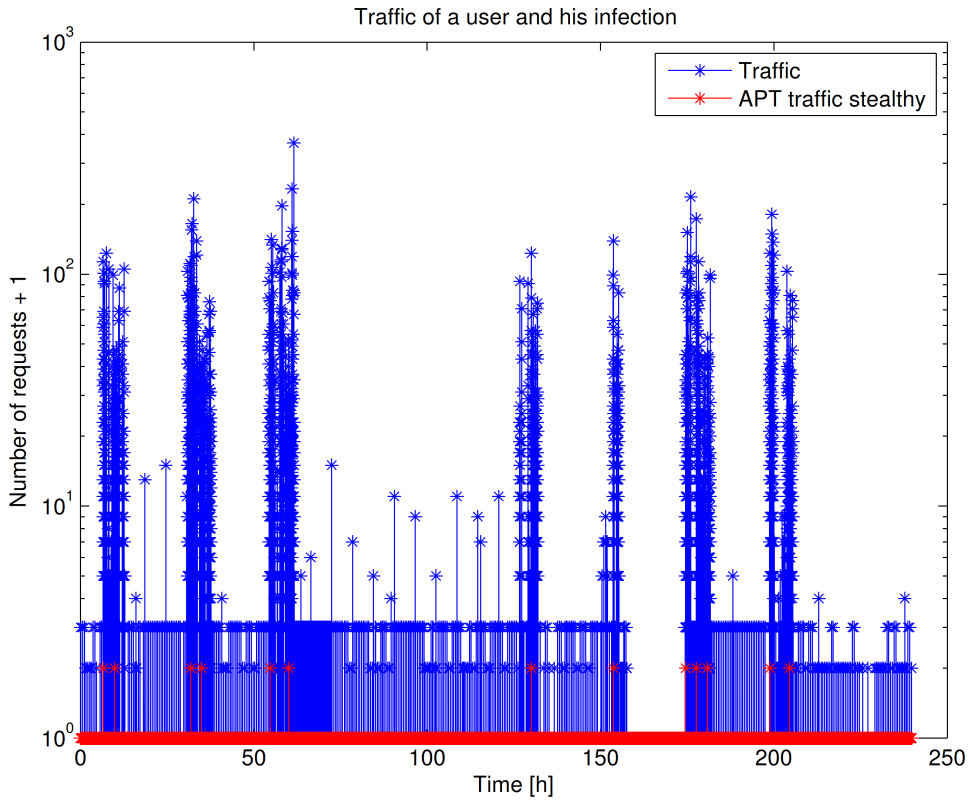
Het is noodzakelijk om een combinatie van parameters te vinden die de AUC maximaliseren. Er wordt gestart met een weldoordachte initiële combinatie van de parameters, waarna elke parameter afzonderlijk wordt aangepast in de volgorde waarin ze in het algoritme voorkomen. Éénmaal een parameter bestudeerd is, wordt de optimale waarde voor deze parameter gebruikt in het vervolg van de testen.



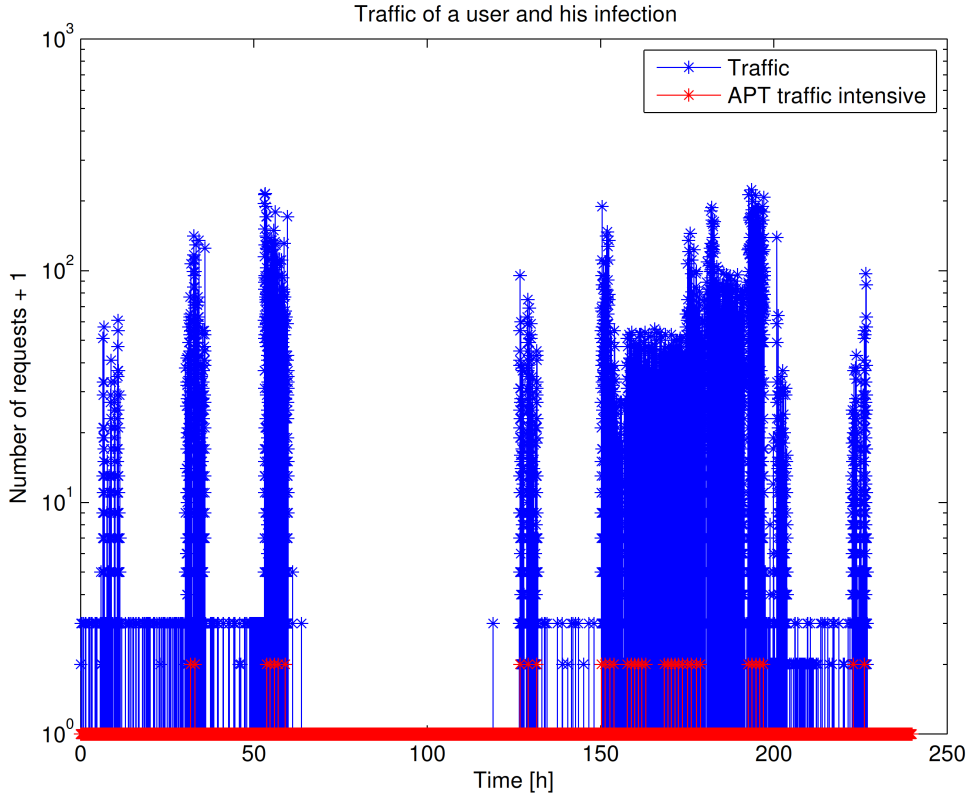
Figuur 2.12: Netwerkverkeer van de geïnfecteerde gebruiker met een periodieke APT met een periode van 1u (Resolutie: 1s) [1].



Figuur 2.13: Netwerkverkeer van de geïnfecteerde gebruiker met een periodieke APT met een periode van 12u (Resolutie : 1s) [1].



Figuur 2.14: Netwerkverkeer van de geïnfecteerde gebruiker met een discrete gegevensstroom gebaseerde APT (Resolutie : 1s) [1].



Figuur 2.15: Netwerkverkeer van de geïnfecteerde gebruiker met een agressieve gegevensstroom gebaseerde APT (Resolutie : 1s) [1].

De initiële combinatie is als volgt:

Waarde van k Dit is de eerste parameter die onderzocht wordt en dus is er geen nood aan een initiële waarde.

Gewichten voor de samenvoeging van de verbanden De initïele waarden worden beiden gelijkgesteld aan 0,5 om geen verband te bevoordelen.

Pruningdrempel Als initiële waarde wordt het gemiddelde genomen dus een z-score gelijk aan 0. Dit leidt tot de verwijdering van alle zwakke verbanden en het behoud van de sterke verbanden.

Maximale clustergrootte Initieel is het gewenst om niet te filteren op de clustergrootte dus wordt er een grote waarde gekozen, namelijk 1 000 000. Dit laat toe de positie van de C2-domeinen te observeren wat ook hun positie in de finale graaf is.

Minimaal aantal request per domein en per gebruiker Om dezelfde redenen als voor de maximale clustergrootte wordt er gekozen om initieel niet te filteren dus wordt een waarde van 1 gekozen.

Gewichten van de scores voor het klassement Bijna identieke gewichten worden gebruikt. Aangezien de C2 van de APT een geïsoleerd domein in graaf zou zijn, wordt een kleine voorkeur gegeven aan de scores gelinkt aan de relatie ouder-kind. Voor zowel de score van de verbanden gericht naar de ouders als deze gericht naar de kinderen wordt een waarde van 0,35 gekozen. Bijgevolg wordt voor de score van het aantal requests 0,3 gekozen.

2.5.3 Resultaten en bespreking

De parameters die voortvloeien uit de testen worden weergegeven in tabel 2.1. Het resultaat dat wordt bekomen door het algoritme met deze parameters toe te passen op de testomgeving wordt geïllustreerd door figuur 2.16. Met de parameters wordt een AUC van 0,87999 behaald. In de volgende alinea's wordt de invloed van elke parameter op het algoritme besproken.

Tabel 2.1: Combinatie van parameters voor een goede detectie.

Waarde van k	100
Gewicht: tijdsgebonden verband	0,1
Gewicht: naamsgebonden verband	0,9
Pruningdrempel	0,0
Pruning z-score	True
Maximale clustergrootte	1 000 000
Clusters z-score	False
Minimaal aantal requests	5
Gewicht klassement: score ouders	0,4
Gewicht klassement: score kinderen	0,4
Gewicht klassement: score aantal requests	0,2

Waarde van k

De waarde van k bepaalt het aantal buren in de initiële grafen. De parameter moet op voorhand gekozen geworden aangezien deze wordt gebruikt in de Batch Processor en niet in de Server. De parameter bepaalt de nauwkeurigheid van het model dat opgesteld is in paragraaf 2.3.1. Aangezien gebruik wordt gemaakt van het NN-Descent algoritme is een minimale waarde van 10 noodzakelijk om goede resultaten te kunnen garanderen. Experimenteel is er vastgesteld dat de rekentijd voor de berekening van de grafen kwadratisch varieert ten opzichte van k. Het verhogen van de waarde van k leidt tot een betere modellering van de webpagina's. Bijgevolg zijn er minder geïsoleerde domeinen en vermindert het aantal valse alarmen. Grote waarden van k moeten steeds worden afgewogen tegen een toenemende rekeningtijd. Daardoor wordt een waarde van 100 gekozen.

Gewichten voor de samenvoeging van de verbanden

De AUC stijgt voor een dalend gewicht van het tijdsgebonden verband. De AUC is maximaal voor een waarde van 0,1 voor het gewicht van het tijdsgebonden verband. Het tijdsgebonden verband is noodzakelijk voor de detectie van APTs maar het gewicht voor de samenvoeging van de verbanden mag relatief klein zijn ten opzichte van het naamsgebonden verband.

Pruningdrempel

Voor een dalende pruningdrempel stijgt de AUC. Toch wordt er niet gekozen om pruningdrempel kleiner dan het gemiddelde te nemen. Een pruningdrempel kleiner dan het gemiddelde bevoordeelt de detectie van periodieke APTs ten opzichte van gegevensstroom gebaseerde APTs. Aangezien er andere methodes bestaan om periodieke APTs te detecteren, wordt een pruningdrempel van 0 (z-score) gekozen.

Maximale clustergrootte

Als er wordt gefilterd op clustergrootte, worden enkel de periodieke APTs gedetecteerd. Als er een grote waarde voor de maximale clustergrootte wordt gebruikt en dus niet wordt gefilterd, worden alle APTs gedetecteerd. Uit de analyse van de finale graaf blijkt dat er een groot aantal heel kleine clusters zijn die meestal geïsoleerde domeinen voorstellen alsook een groot aantal heel grote clusters. Dit zorgt ervoor dat het filteren op clustergrootte niet evident is. Een mogelijke oorzaak van dit verschijnsel is de manier waarop het onderscheid tussen ouders en kinderen wordt gemaakt. De gegevensstroom gebaseerde APTs maken deel uit van grote clusters doordat tijdens clustering de domeinen van een bepaalde cluster te vaak voorkomen als ouder of kind van een APT alhoewel de verbanden met de APT zwak zijn. Het algoritme is gefocust op de detectie van de gegevensstroom gebaseerde APT en bijgevolg wordt er niet gefilterd op clustergrootte dus wordt een waarde van 1 000 000 gekozen voor de maximale clustergrootte.

Aangezien er niet wordt gefilterd op clustergrootte en een groot deel van de rekentijd van de Server wordt gespendeerd aan de berekening van de clusters, is het interessant om het algoritme te wijzigen. Mogelijke aanpassingen zijn ofwel het verwijderen van clustering in het algoritme zodat er rekentijd wordt bespaard, ofwel het aanpassen van clustering waardoor de C2-domeinen van de APTs geen deel uitmaken van grote clusters.

Minimaal aantal request per domein en per gebruiker

Deze parameter heeft een grote invloed op de AUC aangezien deze de ruis verwijdert uit de finale graaf. Als het minimaal aantal requests per domein en per gebruiker stijgt, verbetert de AUC aanzienlijk en verschuift de ROC naar links. Het is belangrijk om de waarde van de parameter niet te groot te maken omdat de kans bestaat dat C2-domeinen ook worden verwijderd. Indien een APT meerdere gebruikers heeft besmet maar voor één gebruiker zijn het aantal verbindingen met de C2 kleiner dan het minimaal aantal, wordt het C2-domein verwijderd uit de graaf. Met de parameter moet er dus voorzichtig worden omgegaan. Zonder verdere uitleg, wordt een waarde van 5 gekozen voor het minimaal aantal request per domein en per gebruiker.

Gewichten van de scores voor het klassement

De score van het aantal requests is minder belangrijk dus mag een klein gewicht worden gekozen. Opnieuw is er duidelijk verschil tussen de twee types APT. Voor de periodieke APTs leidt een negatief gewicht van de score van het aantal requests tot een kleinere kans op vals alarm, daar staat tegenover dat voor een positief gewicht er een kleine kans op vals alarm is voor de gegevensstroom gebaseerde APTs. De periodieke APTs hebben de neiging om meer verbindingen te maken dan de gegevenstroom gebaseerde APTs. Een negatief gewicht bevoordeelt de geïsoleerde domeinen met veel requests, omgekeerd leidt een positief gewicht tot de bevoordeling van geïsoleerde domeinen met weinig requests wat het verschil tussen de detectie van beide types APTs verklaard. Uiteindelijk wordt er gekozen om een klein positief gewicht te gebruiken voor de score van het aantal requests, namelijk 0,2.

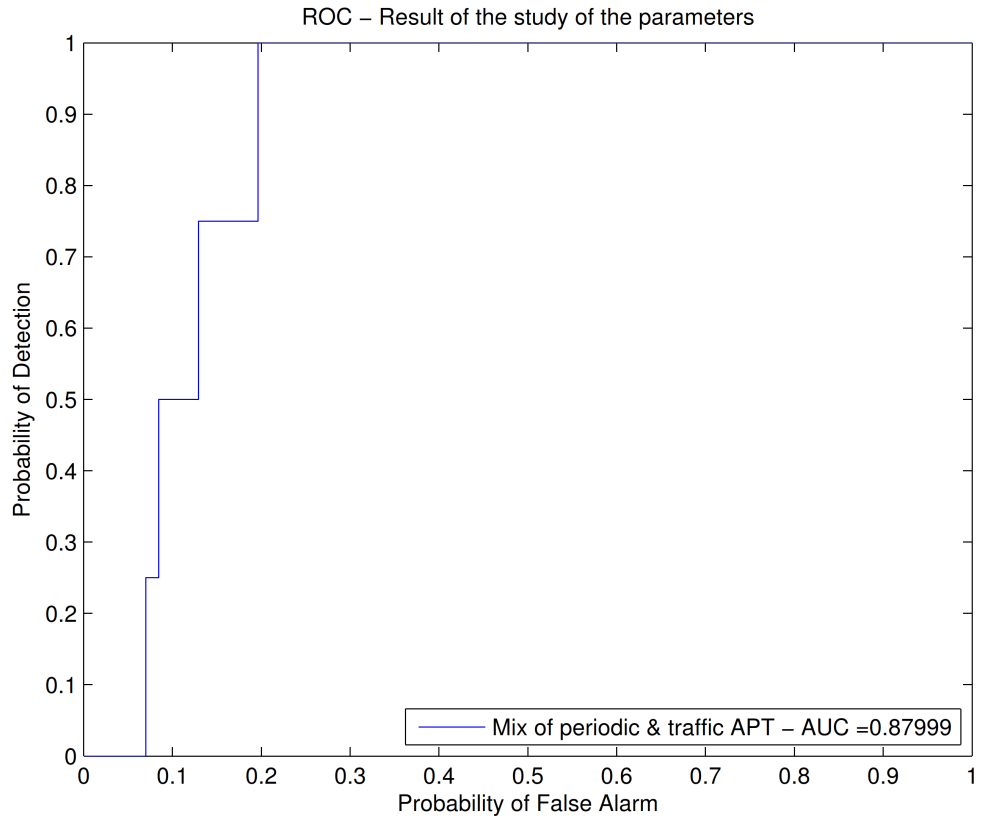
Voor de gegevensstroom gebaseerde APTs heeft het moment waarop de APT verbinding maakt in de burst een grote invloed op de resultaten. Voor deze testen werd gekozen om de verbinding in het midden van de burst te maken (Zie par. 2.5.2). Voor de scores van som van de verbanden voor de ouders en voor deze van de kinderen worden gelijke waarden gekozen, namelijk 0,4.

Voor de opbouw van het klassement wordt een gewogen gemiddelde gebruikt. Evenzeer kunnen andere aggregatietechnieken worden gebruikt die mogelijks leiden tot een betere detectie van APTs. Enkele voorbeelden van andere aggregatietechnieken zijn *Ordered Weighted Average* (OWA) en *Weighted Ordered Weighted Average* (WOWA).

2.5.4 Validatie

De combinatie van parameters die leiden tot een goede detectie van APTs (Tabel 2.1) zijn bekomen door de studie van één bepaalde testomgeving. Om deze parameters te valideren worden ze onderworpen aan een andere testomgeving. Voor deze test wordt een nieuw subnet gekozen om zo andere gebruikers te onderzoeken. Het subnet telt nu 66 gebruikers en de log bevat 1 032 021 requests. Wel wordt opnieuw dezelfde periode bestudeerd.

Voor de validatie wordt er gewerkt met nieuwe APTs. Ditmaal worden er 8 APTs toegevoegd waarvan 4 periodieke en 4 gegevensstroom gebaseerde APTs. De periodieke APTs hebben een periode van 24 uur, 12 uur, 6 uur en 1 uur en hebben respectievelijk 9, 19, 39 en 239 verbindingen gemaakt met hun C2. De gegevensstroom gebaseerde APT hebben allen dezelfde definitie van de burst, namelijk deze moet minimaal 6 seconden duren en tussen 2 requests mag maximum 2 seconden zitten om tot dezelfde burst te behoren. Het maximaal aantal verbindingen per



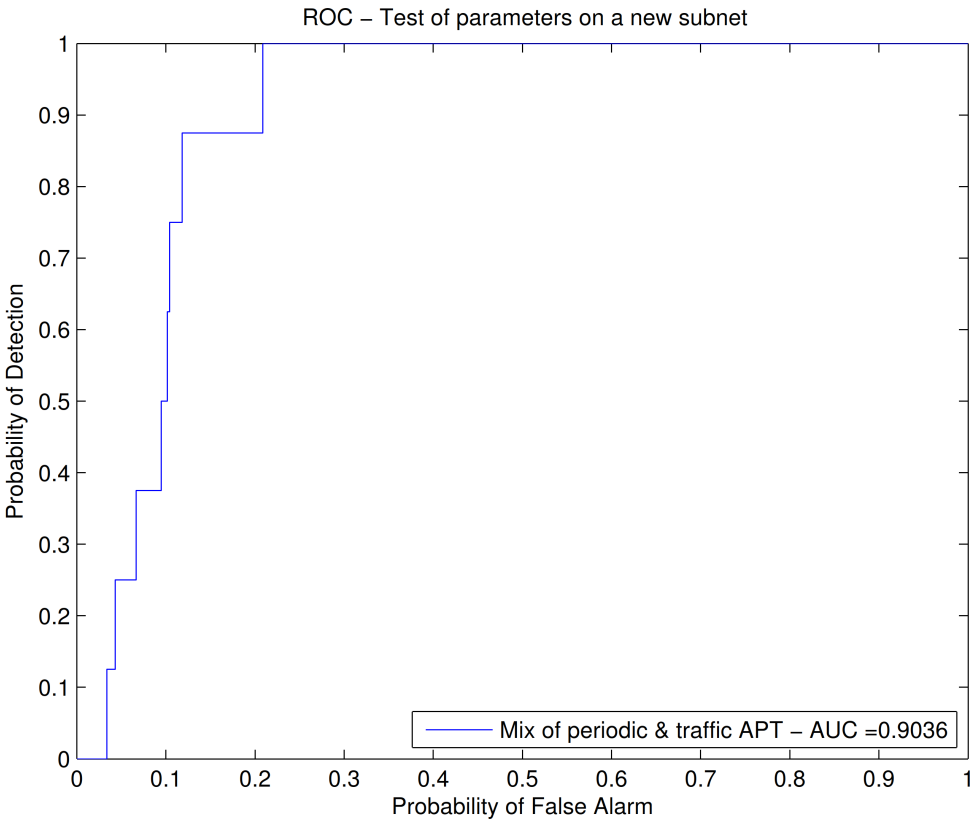
Figuur 2.16: ROC voor de uiteindelijke parameters.

dag voor de gegevensstroom gebaseerde APT is voor allen gelijk aan 10. De gegevensstroom gebaseerde APTs verschillen enkel in de minimale tijd die tussen 2 verbindingen zit, namelijk 4 uur, 2 uur, 1 uur en 30 minuten. Uiteindelijk maken de APTs respectievelijk 11, 24, 25 en 33 verbindingen met de C2.

Het resultaat wordt weergegeven in figuur 2.17. De AUC bedraagt 0,9036. Daaruit valt te concluderen dat het algoritme daadwerkelijk in staat is om APTs te detecteren en dat de initiële parameters voor een goede detectie robuust zijn. De rekentijd voor de Batch Processor bedraagt 11 uur en de Server had 1 uur 30 minuten nodig om de grafen samen te voegen en te verwerken. Van de 1 uur 30 minuten werd er 1 uur gespendeerd aan clustering. Zoals reeds eerder vermeld, zou het handig zijn om clustering aan te passen of te verwijderen uit het algoritme.

2.6 Conclusie

Het doel van algoritme was om een HTTP-proxylog te modelleren als een graaf van domeinen waarin de C2-domeinen van APTs als een geïsoleerde domeinen verschijnen. Het idee achter het model is om de log voor te stellen als een geheel van webpagina's. Om een volledige webpagina te bekijken moeten verschillende domeinen geraadpleegd worden. Er bestaat dus een verband tussen deze domeinen en het initiële domein van de webpagina. Dit verband ligt aan de basis voor de constructie van de grafen. Verder bestaat het algoritme uit drie delen. De Core definieert objecten en methoden die door de andere delen worden gebruikt. Het tijdsgebonden en naamsgebonden verband zijn gedefinieerd in de Core. De Batch Processor is verantwoordelijk voor de opbouw van de grafen per gebruiker en per verband. Vervolgens voegt de Server deze



Figuur 2.17: ROC voor de validatie van de parameters.

grafen samen en voert er enkele bewerkingen op uit om ten slotte een klassement te maken waarin de verdachtste domeinen naar voor komen. Het algoritme is voorzien van een grafische interface die interactiviteit met de analist verzorgt. Eveneens wordt de analist voorzien van een combinatie van parameters die dienen als startpunt om APTs te detecteren. De bekomen resultaten tonen aan dat het mogelijk is om APTs te detecteren aan de hand van grafen gebaseerd op een HTTP-proxylog.

Het idee achter het algoritme is gekend en de werking van het algoritme is toegelicht. Een eerste combinatie van parameters die leiden tot een goede detectie zijn gepresenteerd. De focus in het vervolg van het werk ligt op de optimalisatie van het detectie-algoritme dat is ontworpen door Gilon. De bedoeling is om de optimale parameters voor het detectie-algoritme te vinden t.t.z. een combinatie van parameters die leiden tot een maximale Area Under the Curve.

3. Theoretisch kader

3.1 Inleiding

Nu het detectie-algoritme en de parameters ervan gekend zijn, wordt er gezocht naar methodes om het detectie-algoritme te optimaliseren. In eerste instantie, lijkt het logisch om het volledige detectie-algoritme te optimaliseren t.t.z. de parameters te zoeken die zorgen voor een maximale AUC in een bepaalde omgeving. Aangezien het doorlopen van het detectie-algoritme computationeel intensief is, wordt eveneens onderzocht of het efficiënt is om slechts een deel van het algoritme te optimaliseren. Hierbij wordt gedacht aan de optimalisatie van het klassement omdat de finale graaf dan reeds is berekend. Zo kan er worden gewerkt met de resultaten van de finale graaf en moet het volledige detectie-algoritme niet steeds doorlopen worden. Voor de opbouw van het klassement kan een andere aggregatietechniek worden gebruikt. Als nieuwe aggregatietechniek wordt er geopteerd voor de Weighted Ordered Weighted Average (WOWA) operator, die de voordelen van het gewogen gemiddelde combineert met deze van de Ordered Weighted Average (OWA) operator.

Dit hoofdstuk introduceert de WOWA operator in de volgende paragraaf. Vervolgens wordt er naast de optimalisatie ook uitgelegd welke twee methodes er toegepast worden om het detectie-algoritme te optimaliseren. Het is niet de bedoeling om een formele wiskundige inleiding tot de optimalisatie te geven, maar eerder de lezer vertrouwd te maken met enkele begrippen, noties en methodes van optimalisatie.

3.2 Aggregatietechnieken

Aggregatietechnieken laten toe, net zoals de naam het al suggereert, om data te aggregeren, samen te voegen zodat nieuwe inzichten worden verworven uit deze data. Bekende operatoren zijn het minimum, het maximum, de som en het rekenkundige gemiddelde. Eveneens bestaan er operatoren die gewichten toekennen aan de data, zoals het gewogen gemiddelde, Ordered Weighted Average. Het gewogen gemiddelde laat toe om een gewicht te koppelen aan elke bron van data. Daarentegen laat de OWA operator toe om een gewicht toe te kennen in functie van de waarden van de data. Het maximum en minimum zijn speciale gevallen van de OWA operator. De WOWA operator is tot stand gekomen omdat men de voordelen van het gewogen gemiddelde wou combineren met de voordelen van OWA. WOWA kan dus aanzien worden als een veralgemening van beide operatoren [13]. Met bepaalde parameters leidt WOWA inderdaad tot het gewogen gemiddelde en tot OWA.

3.2.1 Rekenkundige gemiddelde, gewogen gemiddelde en OWA operator

Het rekenkundig gemiddelde en het gewogen gemiddelde zijn een van de bekendste aggregatie-technieken. Het rekenkundig gemiddelde maakt geen gebruik van gewichten. Indien bepaalde data van groter belang zijn, is het mogelijk om een gewogen gemiddelde te gebruiken. Het gewogen gemiddelde koppelt aan elke bron van data een gewicht in functie van hoe belangrijk de data is ten opzichte van het eindresultaat. Wat volgt is de definitie van een vector van gewichten, het rekenkundig gemiddelde en het gewogen gemiddelde.

Definitie 1. [13] Laat $A = (a_1, ..., a_N)$ N data zijn in $\mathbb{R}$. Dan worden een vector van gewichten, het rekenkundige gemiddelde ($AM : \mathbb{R}^N \to \mathbb{R}$) van A, het gewogen gemiddelde (WM) van A met betrekking tot een vector van gewichten als volgt gedefinieerd:

- Een vector $v = (v_1...v_N)$ is een vector van gewichten met dimensie N als en slechts als $v_i \in [0, 1]$ en $\sum_i v_i = 1$.
- AM is het rekenkundig gemiddelde, als $AM(a_1, ..., a_N) = (1/N) \sum_{i=1}^{N} a_i$.
- WM is het gewogen gemiddelde met betrekking tot een vector van gewichten $\mathbf{p}$, als $WM_{\mathbf{p}}(a_1, ..., a_N) = \sum_{i=1}^{N} p_i a_i$.

De OWA operator koppelt ook een gewicht aan de data maar de data wordt eerst geordend. Het eerste gewicht wordt bijgevolg gekoppeld aan de grootste waarde, het laatste (Nde) gewicht met de laagste waarde.

Definitie 2. [13] Laat $\mathbf{w}$ een vector van gewichten met dimensie N, de afbeelding OWA: $\mathbb{R}^N \to \mathbb{R}$ is dan de Ordered Weighted Average (OWA) operator van dimensie N als

$$OWA_{\mathbf{w}}(a_1, ..., a_N) = \sum_{i=1}^{N} w_i a_{\sigma(i)}, \tag{3.1}$$

met $\{\sigma(1), ..., \sigma(N)\}$ een permutatie van $\{1, ..., N\}$ zodoende $a_{\sigma(i-1)} \geq a_{\sigma(i)}$ voor alle $i = \{2, ..., N\}$ (m.a.w. $a_{\sigma(i)}$ is het $i$de grootste element in de verzameling $a_1, ..., a_N$).

Doordat OWA gewichten koppelt aan geordende data, zijn het minimum en het maximum twee speciale gevallen van de OWA operator. Het maximum wordt bekomen door de eerste waarde van de vector van gewicht gelijk te stellen aan 1 en de rest gelijk aan 0. Het minimum doet het omgekeerde, namelijk de laatste waarde van de vector gelijk aan 1 en de rest is 0.

3.2.2 WOWA operator

Weighted Ordered Weighted Average is een operator die een lineaire combinatie van waarden maakt aan de hand van bepaalde gewichten. Deze gewichten worden genoteerd als de vectoren $\mathbf{p}$ en $\mathbf{w}$. De vector $\mathbf{p}$ kan geïnterpreteerd als de gewichten met betrekking tot het gewogen gemiddelde en omgekeerd de vector $\mathbf{w}$ met OWA. De WOWA operator combineert dus beide methoden in één methode.

Definitie 3. [13] Laat $\mathbf{p}$ en $\mathbf{w}$ twee vectoren van dimensie N zijn met bepaalde gewichten, de afbeelding WOWA: $\mathbb{R}^N \to \mathbb{R}$ is dan de Weighted Ordered Weighted Average (WOWA) operator

van dimensie N als

$$WOWA_{\mathbf{p},\mathbf{w}}(a_1, ..., a_N) = \sum_{i=1}^{N} w_i a_{\sigma(i)}, \tag{3.2}$$

met σ gedefinieerd zoals in OWA (m.a.w. $a_{\sigma(i)}$ is het $i$de grootste element in de verzameling $a_1, ..., a_N$), en het gewicht w_i is gedefinieerd als volgt

$$w_i = w^*(\sum_{j \leq i} p_{\sigma(j)}) - w^*(\sum_{j<i} p_{\sigma(j)}), \tag{3.3}$$

met w^* een monotoon stijgende functie die volgende punten interpoleert

$$\{(i/N, \sum_{j \leq i} w_j)\}_{i=1,...,N} \cup \{(0,0)\}. \tag{3.4}$$

3.3 Optimalisatie

Optimalisatie is het proces waarbij gezocht wordt naar de beste oplossing onder alle toegestane oplossingen. Een oplossing is een combinatie van waarden van beslissingsveranderlijken. De beslissingsveranderlijken hebben elk een bepaald bereik. Het gehele domein waarin alle beslissingsveranderlijken vallen heet het zoekdomein. Om een oplossing te kunnen evalueren is er nood aan een evaluatiecriterium. De doelfunctie wordt gebruikt als evaluatiecriterium. De doelfunctie heeft als input de waarden van de beslissingsveranderlijken en als output één waarde, de functiewaarde. Het doel van optimalisatie is om de combinatie van waarden van de beslissingsveranderlijken te vinden die de doelfunctie optimaliseert. Indien de doelfunctie geoptimaliseerd is in de nabijheid van een oplossing, wordt de oplossing een lokaal optimum genoemd. Voor een lokaal optimum bestaat er dus geen betere waarde van de doelfunctie voor alle andere oplossingen in de nabijheid van het lokaal optimum. In het geval dat de oplossing, de beste oplossing in het gehele zoekdomein is, wordt de oplossing het globaal optimum genoemd.

Er bestaan verschillende zoekmethodes. Zoekmethodes die een lokaal optimum zoeken, andere die een globaal optimum zoeken. In het kader van dit werk, wordt er gezocht naar een globaal optimum. Veel methodes maken gebruik van wiskundige bewerkingen op doelfunctie. Aangezien de doelfunctie van het detectie-algoritme niet analytisch te beschrijven is, worden deze methoden niet verder besproken. Een naïeve methode om de beste oplossing te vinden is door alle mogelijke oplossingen te evalueren. Aangezien voor veel problemen het zoekdomein te groot is of de berekening van een functiewaarde te veel tijd in beslag neemt, zijn andere methodes ontwikkeld, de heuristieken. Heuristieken gaan op zoek naar goede oplossingen in een aanvaardbare tijd, zonder de optimaliteit van de oplossing te garanderen. Een metaheuristiek wordt gedefinieerd als volgt [14]:

> "A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions."

Metaheuristieken zijn dus strategieën die het proces leiden met als doel het zoekdomein efficiënt te doorzoeken om zo (bijna-)optimale oplossingen te vinden. Veel metaheuristieken zijn ge-

baseerd op fenomenen van de natuur. Zo is *Particle Swarm Optimization* (PSO) gebaseerd op delen van informatie bij de zoektocht van dieren naar voedsel. Genetisch Algoritme (GA) maakt gebruikt van principes uit de genetica en natuurlijke selectie.

Daar het de bedoeling is om een optimum te vinden in een aanvaardbare tijd, worden meta-heuristieken gebruikt in het werk. Echter kan er niet worden gegarandeerd dat de oplossing het globaal optimum is.

3.3.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is ontstaan na onderzoek over de simulatie van zwermen vogels en scholen vissen. Hun zoektocht naar voedsel is gebaseerd op het principe van het delen van informatie tussen de verschillende individu's [15]. Het PSO-algoritme maakt gebruikt van dit principe. In het algoritme wordt een populatie aan individuen (*particles*) bijgehouden. Deze populatie wordt de zwerm (*swarm*) genoemd. Elk individu stelt een locatie in het zoekdomein voor. De individuen starten op een willekeurige positie en gaan opzoek naar het minimum (of maximum) van een gegeven functie door zich te bewegen in het zoekdomein. De analogie met de zwermen vogels is dat de functie de kwaliteit of de hoeveelheid voedsel op verschillende plaatsen weergeeft en dat de zwerm op zoek is naar de plaats met het beste of het meeste voedsel. De beweging van elk individu is enkel afhankelijk van zijn snelheid en van de plaatsen waar reeds goede oplossingen zijn gevonden ofwel door zichzelf ofwel door andere individuen in de zwerm. Inderdaad, in zwermen vogels zijn de beslissingen door elke vogel gebaseerd op zowel cognitieve als sociale aspecten. Het cognitieve aspect wijst op de eigen kennis dus de plaats waar de vogel zelf al veel eten heeft gevonden. Daartegenover staat het sociale aspect wat uitgaat van de kennis van de groep dus de beste plaats volgens de zwerm.

Een individu wordt gekenmerkt door zijn positie, snelheid en fitheid. Elke keer dat de positie van een individu verandert, wordt de fitheid van het individu aangepast. De fitheid stemt overeen met de functiewaarde van de positie. Elk individu houdt bij wat zijn beste positie is. Dit is de positie met de beste fitheid namelijk de positie met de minimale (of maximale) functiewaarde. Deze positie wordt ook wel de persoonlijke beste positie genoemd. De zwerm houdt eveneens zijn beste positie bij. Dit is de beste positie gekend door alle individuen in de zwerm en wordt de globale beste positie genoemd.

Laat f de te minimaliseren functie zijn die gedefinieerd is over een D-dimensionele ruimte. De locatie van individu i wordt genoteerd als de vector $\mathbf{x}_i = (x_{i1}, ..., x_{iD})$ en de snelheid als $\mathbf{v}_i = (v_{i1}, ..., v_{iD})$. l_d en u_d zijn respectievelijk de onder- en bovenlimiet van de positie van de individuen in de d-de dimensie waarbij $d \in [1, D]$. De persoonlijke beste positie van individu i wordt weergegeven als $\mathbf{p}_i = (p_{i1}, ..., p_{iD})$. De globale beste positie wordt als $\mathbf{g} = (g_1, ..., g_D)$ genoteerd.

Het algoritme begint met het initialiseren van de zwerm. De eerste parameter van het algoritme is de zwermgrootte. Voor elk individu wordt een willekeurige initiële positie en snelheid bepaald. De initiële positie wordt bepaald door voor elke dimensie een willekeurig getal te nemen uit toegestane interval voor de desbetreffende dimensie. Dus voor de d-de dimensie worden er getallen gekozen die een uniforme verdeling hebben over het interval $[l_d, u_d]$. Voor de snelheid wordt voor elke dimensie een willekeurig getal gekozen dat een uniforme verdeling heeft over

Algorithm 1 PSO

```
for elk individu do                                   ▷ Initialiseren van de swarm
    Bepaal initiële positie
    Bepaal initiële snelheid
    Bereken fitheid en persoonlijke beste positie
end for
Bepaal globale beste positie
repeat
    for elk individu do
        Bereken nieuwe snelheid
        Update positie
        Update fitheid en persoonlijke beste positie
    end for
    Update globale beste positie
until Stopcriterium
```

het interval $[-v_{max,d}, +v_{max,d}]$. Typische waarden voor $v_{max,d}$ liggen in het interval $[0,1\text{I}; 1,0\text{I}]$ waarin I gelijk is aan $u_d - l_d$.

Met de initiële positie kan de eerste fitheid berekend worden. Elke keer dat de positie van een individu wijzigt, wordt de persoonlijke beste positie geüpdatet. Indien $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ dan wordt $\mathbf{p}_i = \mathbf{x}_i$. Aangezien het de eerste fitheid is, wordt de eerste positie steeds ook de eerste persoonlijke beste positie voor elk individu. Nadat alle individuen gekend zijn, is het mogelijk om de globale beste positie te bepalen.

Vervolgens worden enkele stappen steeds herhaald tot er voldaan is aan een bepaald stopcriterium. Dit stopcriterium is een andere parameter van het algoritme en vaak wordt er een maximum aantal iteraties opgelegd.

Eerst wordt voor elk individu een nieuwe snelheid berekend. De snelheidscomponent in de d-de dimensie voor k-de iteratie wordt als volgt geüpdatet:

$$v_{id}^{k} = w.v_{id}^{k-1} + c_1.r_1.(p_{id}^{k-1} - x_{id}^{k-1}) + c_2.r_2.(g_{d}^{k-1} - x_{id}^{k-1}) \tag{3.5}$$

waarin

- w, het inertiegewicht is; het bepaalt de invloed van de oude snelheid; hoe hoger deze waarde, hoe meer de individuen in nieuwe gebieden zoeken; typische waarden voor w zijn iets kleiner dan 1;
- c_1 en c_2, respectievelijk het cognitief en het sociaal gewicht zijn; deze bepalen de invloed van de persoonlijke beste en globale beste positie; typische waarden zijn c_1 en c_2 gelijk aan 2;
- r_1 en r_2, willekeurige waardes tussen 0 en 1 zijn.

Als de snelheid groter is dan de vooropgestelde maximumsnelheid, wordt de snelheid gelijk gesteld aan de maximum snelheid. Daaropvolgend wordt de positie van elk individu aangepast

op volgende manier:

$$x_{id}^{k} = x_{id}^{k-1} + v_{id}^{k}. \tag{3.6}$$

Indien de nieuwe positie de limieten van een bepaalde dimensie overschrijdt, wordt het individu weerkaatst. Nu de nieuwe positie gekend is, kan de fitheid bepaald worden. Voor elk individu wordt de persoonlijke beste positie geüpdatet en voor de zwerm de globale beste positie. Een beknopte samenvatting van het algoritme wordt weergeven in *Algorithm 1.*

De parameters waarmee het algoritme wordt geregeld zijn de zwermgrootte, de maximum snelheid, het inertiegewicht, het cognitief gewicht, het sociaal gewicht en het stopcriterium. In de literatuur wordt een zwermgrootte aangeraden van 2 tot 5 keer de dimensie van het probleem [16]. Voor de maximum snelheid worden waardes van 10 tot 100 % van het bereik van de desbetreffende dimensie aangeraden. Te lage waardes voor de maximumsnelheid, beletten de individuen om het volledige domein te doorzoeken. Het inertiegewicht bepaalt eveneens in welke mate nieuwe domeinen doorzocht worden. Hoge waardes bevorderen de zoektocht in nieuwe domeinen, omgekeerd leiden lage waardes tot lokale zoektochten. Gebruikelijke waarden voor het inertiegewicht zijn iets kleiner dan 1. Het cognitieve en sociale gewicht bepalen in welke mate de persoonlijke en globale best oplossing de snelheid beïnvloeden en worden meestal beide gelijk gesteld aan 2. Uiteindelijk is er het stopcriteria waarvoor meestal een maximum aantal iteraties wordt gekozen. Hoe groter de dimensie, hoe wenselijker het is om een groot aantal maximum iteraties te kiezen. Het is echter belangrijk om rekening te houden met de tijd waarin het algoritme afgerond moet zijn. Indien de tijd gekend is voor de berekening van 1 functiewaarde, kan met deze tijd makkelijk een schatting worden gemaakt van de totale rekentijd. Het aantal functiewaarden dat het algoritme moet bepalen, bedraagt namelijk het product van de zwemgrootte en het maximum aantal iteraties. Vaak speelt de rekentijd een belangrijke rol bij de keuze van het maximum aantal iteraties.

Het is mogelijk om parameters te wijzigen tijdens de verschillende iteraties. Experimenteel is aangetoond dat een lineair dalend inertiegewicht met beginwaarde 0,9 en stopwaarde 0,4 performant is [17]. Voor elke iteratie wordt het inertiegewicht als volgt berekend:

$$w = w_1 - (w_1 - w_2)\frac{k}{N} \tag{3.7}$$

waarin w_1, gelijk is aan de startwaarde voor het gewicht; w_2, de eindwaarde; k, de iteratie; N, het maximum aantal iteraties.

3.3.2 Genetisch algoritme

Genetisch algoritme (GA) is gebaseerd op de principes van genetica en natuurlijke selectie [18]. GA maakt gebruikt van een populatie van individuen waarin elk individu een mogelijke oplossing van het probleem voorstelt. Het GA zorgt ervoor dat de populatie evolueert naar de beste oplossing, de oplossing waarvoor de functiewaarde minimaal (of maximaal) is. De functiewaarde van een oplossing wordt ook de fitheid van het individu genoemd. Naar analogie met de genetica wordt een oplossing een chromosoom genoemd, een parameter van de oplossing wordt een gen en de uiteindelijke waarde van deze parameter noemt een allel. Momenteel bestaan er twee mogelijke implementatietechnieken voor GA, de discrete en de continue. Voor het discreet GA

stellen de allelen een symbolenreeks uit een alfabet voor. Meestal wordt er gebruikt gemaakt van het binair alfabet. Het is dus noodzakelijk om de beslissingsveranderlijken te coderen volgens een bepaald schema. Als de beslissingsveranderlijken continu zijn, wordt er dus een kwantisatiefout gemaakt. Daaruit is het idee ontstaan om een continu GA te ontwikkelen waarin de allelen continue waarden voorstellen. In de volgende alinea's wordt het continu genetisch algoritme verder besproken. Bij het discreet GA is de methode hetzelfde maar zijn de bewerkingen anders aangezien er niet met continue waarden wordt gewerkt.

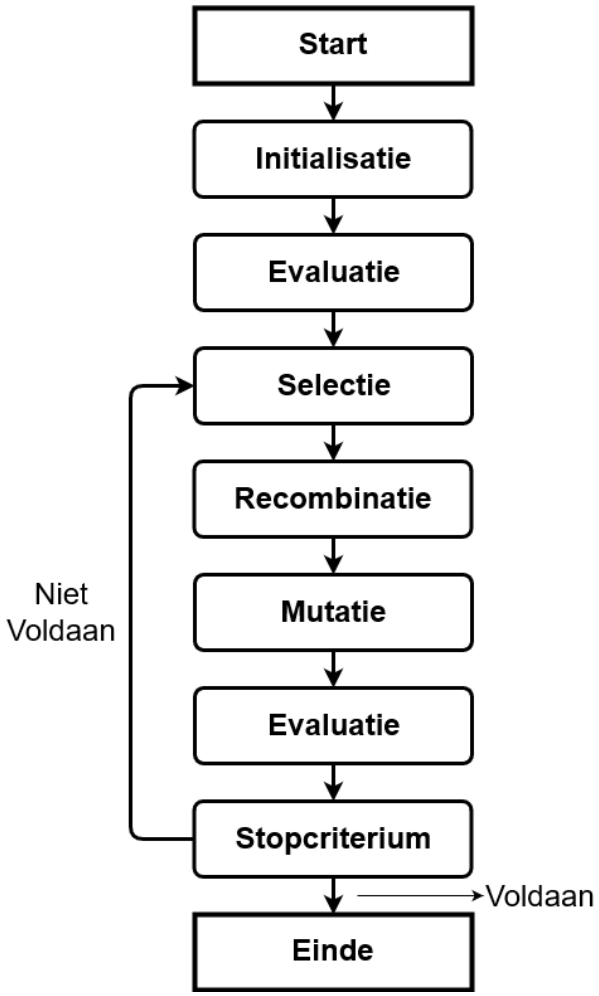
Het genetisch algoritme start met het maken van een initiële populatie. Voor elk individu wordt de fitheid bepaald. Uit de initiële populatie wordt op basis van natuurlijke selectie een nieuwe populatie gemaakt. Deze populatie telt minder individuen dan de initiële populatie. Uit deze nieuwe populatie worden vaders en moeders gekozen om door recombinatie kinderen te maken totdat de grootte van de populatie gelijk is aan de grootte van de initiële populatie. Vervolgens treedt er mutatie op van de chromosomen. Dit zorgt ervoor dat er nieuwe domeinen worden doorzocht. Uiteindelijk wordt de fitheid van de individuen van de nieuwe populatie berekend en worden er steeds nieuwe populaties gemaakt volgens voorgaande stappen totdat er aan een stopcriterium wordt voldaan. Een schematische voorstelling van het algoritme wordt weergegeven in figuur 3.1. De parameters die het algoritme beïnvloeden zijn de populatiegrootte, de selectiemethode, de recombinatieverhouding, de mutatiegraad en het stopcriterium. Als stopcriterium wordt vaak een maximum aantal iteraties opgelegd.

Initialisatie

Het bepalen van de initiële populatie berust meestal op het willekeurig kiezen van punten in het zoekdomein. Indien er reeds informatie over de functie beschikbaar is, kan de keuze van de punten worden beïnvloed door deze informatie. Ook is het mogelijk om de initiële populatie uniform te verdelen over de ruimte.

Selectie

De selectie is gebaseerd op natuurlijke selectie dus individuen met een grotere fitheid hebben een grote kans om geselecteerd te worden. Er bestaan verschillende methodes zoals *Roulette Wheel Selection* (RWS) en *Tournament Selection* (TOS). Bij RWS wordt aan elk individu een kans op selectie gekoppeld die proportioneel is met hun fitheid. Hoe groter de fitheid (kleinere functiewaarde in geval van minimalisatie), hoe groter de kans. Vervolgens wordt er, rekening houdend met deze kansen, een individu geselecteerd. Naar analogie met de roulette is dat met elk individu een vakje overeenstemt. De grootte van een vakje is evenredig met de fitheid van het individu. Vervolgens wordt aan de roulette gedraaid. Het individu, dat overeenstemt met het vakje waar het balletje terechtkomt, wordt geselecteerd. Een andere methode is de TOS. Hierbij worden willekeurig twee individuen geselecteerd en het individu met de beste fitheid wordt definitief geselecteerd. Deze methodes worden herhaald totdat er voldoende individuen zijn gekozen. Dit aantal wordt bepaald door het complement van de recombinatieverhouding. Stel dat de populatie 100 individuen telt en de recombinatieverhouding is gelijk aan 0,60. Dan worden er 40 (100.(1-0,60)) individuen geselecteerd. Eveneens is het mogelijk om een vorm van elitarisme toe te voegen aan de selectie. Dit wil zeggen dat de elite sowieso geselecteerd wordt. De elite zijn één of meerdere individuen die de beste fitheid hebben.



Figuur 3.1: Schematische voorstelling van het GA.

Recombinatie

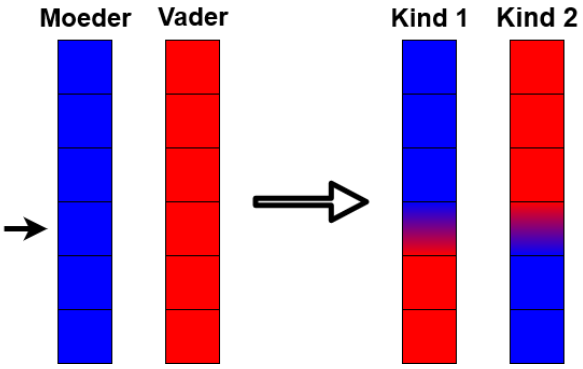
Recombinatie heeft als doel om nieuwe individuen te maken uit de individuen die voortkomen uit de selectie. Daardoor worden de individuen die geselecteerd werden ook wel ouders genoemd en bijgevolg zijn de nieuwe individuen de kinderen. Om te starten worden er willekeurig twee ouders gekozen, een moeder en een vader. Uit deze ouders zullen twee kinderen voortkomen. Vervolgens wordt willekeurig een gen gekozen. Dit gen is de plaats waar de chromosomen van de ouders gesplitst worden. Voor de moeder gaat het eerste deel van het chromosoom naar het eerste kind en het tweede deel naar het tweede kind. Voor de vader geldt het omgekeerde. Dit wordt weergegeven in figuur 3.2. De nieuwe chromosomen van de kinderen zijn bijna volledig. De plaats waar het chromosoom is gesplitst, moet enkel nog bepaald worden. Hiervoor wordt een willekeurig getal tussen 0 en 1 genomen. Vervolgens wordt het ontbrekende gen van het eerste kind als volgt berekend:

$$p_{kind_1} = \alpha.p_{moeder} + (1 - \alpha).p_{vader} \tag{3.8}$$

met α, het willekeurig getal; p_{kind_1}, de waarde van het gen voor het eerste kind; p_{moeder}, de waarde van de moeder; p_{vader}, de waarde van de vader.

Deze berekening is eigenlijk een gewogen gemiddelde waarbij de gewichten worden bepaald door

het willekeurig getal α. Voor het tweede kind geldt bijna dezelfde formule, nu worden echter de gewichten omgewisseld. Andere recombinatiemethodes bestaan maar deze vallen buiten het bestek van dit werk.



Figuur 3.2: Visuele voorstelling van de recombinatie.

Mutatie

Mutatie is noodzakelijk om nieuwe domeinen te doorzoeken. Mutatie wordt geregeld door de mutatiegraad. Deze waarde stelt de kans voor dat een individu muteert. Elk individu in de populatie wordt doorlopen en heeft dus een kans om te muteren. Een individu muteert op volgende manier. Eerst wordt een willekeurig gen van het chromosoom gekozen waarna dit gen een nieuwe willekeurige waarde krijgt in het toegestane domein van die waarde. Indien elitarisme is toegevoegd aan het algoritme, ondergaat de elite geen mutatie d.w.z. de kans op mutatie van een individu uit de elite is gelijk aan 0. Andere mutatiemethodes bestaan maar deze vallen buiten het kader van dit werk.

3.4 Conclusie

In dit hoofdstuk werden theoretische begrippen uitgelegd die in het vervolg van het werk worden toegepast. De eerste paragraaf werd besteed aan aggregatietechnieken met als doel de WOWA operator in te leiden. De WOWA operator combineert de voordelen van zowel het gewogen gemiddelde als van de OWA operator. Daardoor heeft de WOWA operator nood aan twee vectoren van gewichten. Vervolgens werd het begrip optimalisatie uitgelegd. Optimalisatie heeft als doel om de beste oplossing te vinden uit alle mogelijke oplossingen. Er bestaat een waaier aan optimalisatiemethoden. Belangrijk bij de keuze van de optimalisatiemethode is de totale rekentijd. Aangezien het detectie-algoritme rekenintensief is, wordt er geopteerd voor metaheuristieken. Deze hebben als doel om een goede oplossing in een aanvaardbare tijd te vinden, echter kan de optimaliteit van de oplossing niet gegarandeerd worden. Ten slotte werden twee metaheuristieken beschreven, namelijk Particle Swarm Optimization en Genetisch algoritme. Beide algoritmen gaan uit van een verzameling van oplossingen die stap na stap evolueren naar betere oplossingen.

4. Implementatie en integratie

4.1 Inleiding

Het detectie-algoritme samen met de optimalisatiemethodes zijn reeds uitvoerig beschreven. In dit hoofdstuk wordt uitgelegd op welke manier deze methoden geïmplementeerd en geïntegreerd zijn in het detectie-algoritme. Het eerste deel van dit hoofdstuk gaat over de aanpassingen die gemaakt zijn aan het detectie-algoritme. Zoals reeds vermeld heeft de manier van clustering in het detectie-algoritme geen meerwaarde. Ook wordt het detectie-algoritme aangepast zodat de optimalisatiemethodes kunnen toegepast worden op het algoritme. Het tweede deel focust op de implementatie en integratie van de optimalisatie van de gewichten van de WOWA operator. Ten slotte, wordt de implementatie en integratie van de optimalisatiemethode voor het volledige detectie-algoritme verduidelijkt.

4.2 Aanpassingen aan het detectie-algoritme

4.2.1 Clustering

De manier van clustering heeft geen meerwaarde voor het detectie-algoritme. Mogelijke verbeteringen zijn het schrappen van clustering zodoende er aanzienlijke rekentijd wordt gewonnen, of het aanpassen van de manier waarop er wordt geclusterd zodoende de APTs geen deel meer uitmaken van de grote clusters. De focus van het werk ligt op het vinden van de optimale parameters.

Al meerdere malen is vermeld dat het detectie-algoritme veel rekentijd vraagt en de rekentijd een grote invloed heeft in het optimalisatieproces.Daarom wordt er geopteerd om clustering niet meer toe te passen in het detectie-algoritme. Het detectie-algoritme wordt voorzien van een extra parameter, een boolean[1], waarmee beslist kan worden of het algoritme clustering al dan niet moet uitvoeren. Indien de boolean 'waar' aanduidt, wordt clustering toegepast en omgekeerd leidt 'niet waar' tot geen clustering. Belangrijk om op te merken is dat clustering echter niet wordt verwijderd uit het algoritme. De mogelijkheid voor een opvolgend werk bestaat erin om clustering aan te passen zodat het efficiënter wordt en toch kan toegepast worden in het algoritme. Dus door de keuze van een boolean wordt er werk gespaard voor een opvolgend werk.

Het detectie-algoritme werd onderworpen aan een test om uit te maken of clustering een invloed heeft op het eindresultaat, namelijk het klassement. Voor een willekeurig logbestand werd het detectie-algoritme uitgevoerd met de parameters bekomen door Gilon, zowel met als zonder

[1]Een boolean is een datatype dat twee waardes kan hebben, namelijk *True* ('waar') of *False* ('niet waar').

clustering. Hetzelfde klassement werd bekomen, maar er was wel een beduidend verschil in de rekentijd. De Server had 39 minuten 21 seconden nodig om het klassement te berekenen met clustering daartegenover had het maar 3 minuten 30 seconden nodig zonder clustering.

4.2.2 WOWA Operator

Één van de optimalisatiemethodes is de optimalisatie van de gewichten van de WOWA operator. Om de geoptimaliseerde gewichten te kunnen gebruiken in het algoritme is het noodzakelijk om de WOWA operator toe te voegen aan het detectie-algoritme. Opnieuw wordt er een boolean aan de parameters van het algoritme toegevoegd die aangeeft of de WOWA operator ('waar') of het gewogen gemiddelde ('niet waar') moet worden toegepast.

Het spreekt voor zich dat indien de WOWA operator toegepast moet worden, de gewichten moeten gekend zijn voor het algoritme. Het algoritme wordt voorzien van drie nieuwe parameters, de gewichten van de WOWA operator met betrekking tot de OWA operator (zie paragraaf 3.2.2). De gewichten voor de WOWA Operator met betrekking tot het gewogen gemiddelde kunnen ingegeven worden bij de gewichten voor het gewogen gemiddelde. De gewichten voor het gewogen gemiddelde kunnen dus twee betekenissen hebben. Enerzijds kunnen het inderdaad de gewichten zijn voor het gewogen gemiddelde (als de boolean voor WOWA 'niet waar' is). Anderzijds betekenen de gewichten voor het gewogen gemiddelde, de gewichten voor de WOWA Operator met betrekking tot het gewogen gemiddelde (als de boolean voor WOWA 'waar' is).

Het algoritme is nu voorzien van de parameters om de WOWA operator te gebruiken. De WOWA operator dient echter nog geïmplementeerd te worden in het detectie-algoritme. De implementatie van de WOWA operator door Thibault Debatty wordt gebruikt en is terug te vinden in [19]. De WOWA operator is geïmplementeerd in Java. Vervolgens wordt deze implementatie geïntegreerd in het detectie-algoritme. In de methode die verantwoordelijk is voor de berekening van het klassement, wordt een *IF-ELSE statement* toegevoegd. Aan de hand van WOWA boolean wordt er dan beslist welke operator toegepast moet worden op de scores.

Zoals verder zal blijken in paragraaf 4.3.1, heeft de optimalisatiemethode voor de WOWA operator nood aan genormaliseerde scores om zo de gewichten van de WOWA operator te kunnen optimaliseren. Een genormaliseerde score is een score waarvan de waarde tussen 0 en 1 ligt. Deze scores stellen uiteindelijk de kans voor dat het desbetreffende domein een APT is.

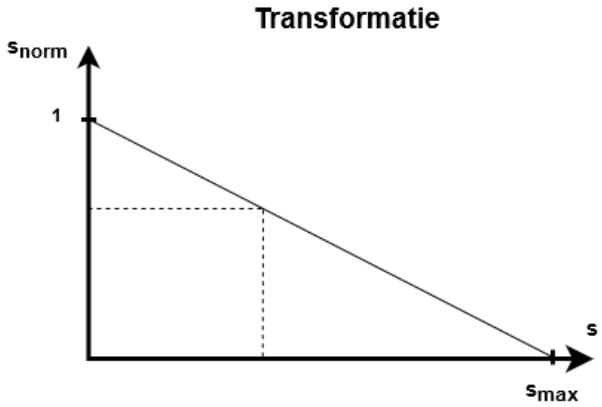
De waarde van de scores van de som van alle verbanden gericht naar ouders, de som van alle verbanden gericht naar kinderen en het aantal requests naar het desbetreffende domein, moeten elk omgezet worden naar een waarde die de waarschijnlijkheid voorstelt dat het overeenstemmende domein een C2 van een APT is. Elke score ondergaat dus een transformatie die de waarden van de score omzet naar een genormaliseerde waarde die de waarschijnlijk voorstelt dat het domein een C2 van een APT is. Zoals reeds vermeld in paragraaf 2.4.3, zou de C2 van de APT in de finale graaf maximaal geïsoleerd moeten zijn wat betekent dat de C2 een minimum aan verbanden naar zowel ouders als kinderen heeft. Daarenboven heeft de APT als doel om discreet te blijven dus zijn het aantal requests naar de C2 beperkt. Bijgevolg moet een kleine waarde van de score leiden tot een grote waarschijnlijk.

De waarde van de scores van de som van de verbanden gericht naar de ouders en de som voor de kinderen zijn positieve reële getallen. De waarde van de score voor het aantal requests zijn

positieve gehele getallen. Bijgevolg wordt er geopteerd om volgende transformatie toe te passen op elke score:

$$s_{norm} = 1 - \frac{s}{s_{max}} \tag{4.1}$$

waarin s_{norm} de genormaliseerde waarde van score voorstelt; s, de oorspronkelijke waarde van de score; s_{max} de maximum waarde van desbetreffende score.



Figuur 4.1: Grafische voorstelling van de transformatie.

De transformatie wordt voorgesteld in figuur 4.1. Indien de waarde van een score 0 is, wordt de waarde genormaliseerde score 1. Voor stijgende waardes van de score daalt de waarde van de genormaliseerde score tot de maximumwaarde van de score is bereikt. De maximumwaarde van de score stemt overeen met een waarde van 0 voor de genormaliseerde score. De transformatie beschikt over de gewenste eigenschappen. Het gedrag van de C2 van een APT leidt tot lage waardes van de scores die worden omgezet naar hoge waarschijnlijkheden.

Het detectie-algoritme beschikt over een nieuwe parameter, een boolean die beslist om genormaliseerde scores te gebruiken ('waar') of de oorspronkelijke scores te gebruiken ('niet waar'). De invloed van het gebruik van genormaliseerde scores kan onderzocht worden in een opvolgend werk. Eveneens is het mogelijk om andere transformaties te bestuderen.

4.3 Optimalisatie van de gewichten van de WOWA operator

4.3.1 Achterliggend idee

Een onderzoeker van de Research Unit Cyber Defense (RUCD) van de Koninklijke Militaire School, Alexandre Croix, was tijdens dit werk bezig met onderzoek naar de training van een multi-criteria beslissingssysteem. Dit systeem werd ontworpen voor de detectie van PHP[1] webshells. Meer informatie over het onderzoek kan worden gevonden in [21]. Aangezien het detectie-algoritme eveneens een beslissing moet maken of het domein al dan niet een C2 van een APT is aan de hand van enkele scores, is het idee gegroeid om dit systeem eveneens te implementeren in het detectie-algoritme.

[1]PHP: Hypertext Preprocessor is een *open source* scriptaal die bijzonder geschikt is voor webontwikkeling [20].

4.3.2 Implementatie

Het systeem is een optimalisatie-algoritme dat initieel is ontworpen in PHP maar later is omgezet naar Java [22]. Het systeem is een implementatie van een Genetisch Algoritme en is eenvoudig in gebruik. Om te starten is het noodzakelijk dat de parameters voor het Genetisch Algoritme worden gegeven. Deze parameters zijn de populatiegrootte, de recombinatieverhouding, de mutatiegraad, de selectiemethode, de manier waarop de populatie geïnitialiseerd wordt en maximum aantal iteraties.

Vervolgens moet het optimalisatie-algoritme voorzien worden van data. Deze data stellen enerzijds de scores van de domeinen voor en anderzijds de classificatie van de domeinen. De scores moeten wel genormaliseerd zijn zoals uitgelegd in paragraaf 4.2.2. De scores stellen dan de verschillende waarschijnlijkheden voor dat het domein zich als een C2 van een APT gedraagt. De waarschijnlijkheden zijn dan gelinkt aan de som van de verbanden gericht naar de ouders, aan de som van de verbanden gericht naar de kinderen en aan het aantal requests naar het domein. Aansluitend moet de classificatie van de domeinen worden gegeven. Dit gebeurt aan de hand van een vector waarin elke waarde een boolean voorstelt die aangeeft of het domein al dan niet een APT voorstelt. Het is dus noodzakelijk om op voorhand te weten welke domeinen er een APT voorstellen en welke niet. Dit kan enkel gegarandeerd worden als er verondersteld wordt dat de log niet is besmet en dat er op vrijwillige basis APTs aan het netwerkverkeer zijn toegevoegd.

Met deze data kan het optimalisatie-algoritme gestart worden. Het algoritme doorloopt alle stappen van een Genetisch Algoritme die beschreven zijn in paragraaf 3.3.2. Een mogelijke oplossing wordt voorgesteld door twee vectoren van gewichten van 3 dimensies. Een vector met betrekking tot het gewogen gemiddelde en een met betrekking tot de OWA operator. Voor de evaluatie van de oplossing wordt de WOWA operator met de respectievelijke gewichten toegepast op de scores van de domeinen. Zo worden de scores van één domein omgezet naar één score die de uiteindelijke waarschijnlijkheid weergeeft waarbij het overeenstemmende domein een APT voorstelt. Uiteindelijk wordt een vector van waarschijnlijkheden bekomen waarin elke waarde de waarschijnlijkheid weergeeft dat het domein een APT is. Een Receiver Operating Characteristic (ROC) wordt opgebouwd uit deze vector en de vector die de classificatie van de domeinen voorstelt. Uiteindelijk leidt de ROC tot een Area Under the Curve (AUC), die fitheid van de oplossing (twee vectoren met gewichten van 3 dimensies) weergeeft.

4.3.3 Integratie

De beslissing werd genomen om het optimalisatie-algoritme te integreren in het detectie-algoritme zodanig dat de parameters van het Genetisch Algoritme vast staan. De keuze van de parameters is gebaseerd op de bevindingen van Alexandre Croix [22]. Een populatiegrootte van 100 wordt gekozen. De recombinatieverhouding bedraagt 0,60 en de mutatiegraad 0,15. De selectiemethode is RWS. Het maximum aantal iteraties bedraagt 110. De initiële populatie wordt volledig willekeurig gekozen.

Het detectie-algoritme wordt opnieuw voorzien van een nieuwe parameter, een boolean die toelaat om de gewichten van de WOWA operator te optimaliseren (waar) of om eigen gekozen gewichten te gebruiken (niet waar). Belangrijk om op te merken is dat de input van het algo-

ritme een log moet zijn waarvoor de APTs vrijwillig zijn toegevoegd met een domeinnaam die eindigt op `.apt`. Dit zorgt ervoor dat het algoritme de classificatie van de domeinen zelfstandig kan uitvoeren. Eveneens worden automatisch de genormaliseerde scores berekend als er wordt gekozen om WOWA optimalisatie toe te passen. Na de optimalisatie worden de bekomen optimale gewichten toegepast in het detectie-algoritme en gaat het verder met de berekening van het uiteindelijke klassement.

4.4 Volledige optimalisatie van het detectie-algoritme

De optimalisatie van het volledige detectie-algoritme wordt uitgevoerd door gebruik te maken van de Particle Swarm Optimization methode. De implementatie wordt net zoals het detectie-algoritme uitgevoerd in Java. De implementatie van het optimalisatie-algoritme bestaat uit twee fasen. In een eerste fase wordt het algoritme geïmplementeerd om analytische doelfuncties met reële beslissingsveranderlijken te kunnen optimaliseren. Deze fase is noodzakelijk om de werking van het algoritme te valideren aan de hand van verscheidene testfuncties. Éénmaal de werking van het algoritme gevalideerd is, begint de tweede fase van de implementatie. In de tweede fase wordt het algoritme aangepast aan de werking van het detectie-algoritme. Niet alle parameters van het detectie-algoritme zijn reëel en hebben eenzelfde bereik. Eveneens is het niet meer mogelijk om de doelfunctie analytisch te beschrijven. De doelfunctie bestaat nu uit het doorlopen van de verschillende stappen van het detectie-algoritme en moet uiteindelijk een AUC voortbrengen.

Ook is het noodzakelijk om het optimalisatie-algoritme te integreren in het detectie-algoritme. Het detectie-algoritme bestaat uit verschillende *packages*[1]. Bijvoorbeeld, de verschillende onderdelen van het detectie-algoritme zijn elk een package. De implementatie van het optimalisatie-algoritme is een nieuwe package van het detectie-algoritme.

4.4.1 Implementatie

Om de implementatie van het optimalisatie-algoritme te begrijpen, is het noodzakelijk om te weten hoe de programmeertaal Java werkt. Java is object georiënteerd d.w.z. dat Java gebruik maakt van objecten en klassen. Een klasse kan gezien worden als een blauwdruk en een object is een realisatie van een bepaalde klasse. De klasse definieert welke data (variabelen) en welke bewerkingen (methoden) er mogelijk zijn voor een realisatie van de klasse (object). Bijvoorbeeld, een klasse persoon waarin de variabelen van de klasse naam, geslacht, geboortedatum en adres kunnen zijn. Mogelijke methodes zijn verhuizen zodat het adres wijzigt of de leeftijd bereken aan de hand van de geboortedatum. Een reëel persoon wordt dan voorgesteld als een object van de klasse persoon. Aan de variabelen worden dan waardes gelinkt die overstemmen met de reële persoon.

Eerste fase

Het detectie-algoritme bestaat uit 8 klassen waarvan een klasse de *Main* klasse voorstelt. Deze klasse bevat de methode *main* die het startpunt is van de uitvoering van het optimalisatie-

[1] Een package in de programmeertaal Java is een bundel van gerelateerde code. Lezers die meer informatie wensen over het begrip package kunnen volgende website raadplegen: `https://docs.oracle.com/javase/tutorial/java/package/index.html`.

algoritme. De andere klassen zijn *Problem*, *Parameters*, *Swarm*, *Particle*, *Position* en *Velocity*. Rekening houdende met de werking van PSO methode, lijkt dit een logische indeling van het optimalisatie-algoritme.

Een realisatie van de klasse Position en Velocity stellen de positie en snelheid van een individu in de zwerm voor. De klasse Position en Velocity bevatten beiden methoden die het mogelijk maken om de positie of snelheid in te stellen en weer te geven. Het individu in de zwerm wordt voorgesteld door een realisatie van de klasse Particle. Deze klasse bevat een realisatie van de klasse Position en Velocity en de variabelen fitness (fitheid), pbest (persoonlijke beste fitheid) en pbestposition (een realisatie van de klasse Position die de persoonlijke beste positie voorstelt). Eveneens bezit deze klasse methoden om al deze variabelen aan te passen of weer te geven.

Om de fitheid van een individu in te stellen wordt er gebruik gemaakt van de klasse Problem. In deze klasse wordt het te optimaliseren probleem gedefinieerd. In de eerste fase van de implementatie is het mogelijk om de dimensie van probleem, het bereik van de beslissingsveranderlijken en de doelfunctie zelf in te stellen in deze klasse.

Nu het individu volledig is beschreven door de klasse Particle, is het nodig om de zwerm te beschrijven. Een realisatie van de klasse Swarm stelt de zwerm voor. Deze klasse houdt een lijst bij van alle objecten van de klasse Particle (alle individuen), een lijst met alle fitheden die gelinkt zijn aan de individuen, en de parameters van het optimalisatie-algoritme. Eveneens heeft de klasse een variabele gbest (globale beste fitheid) en gbestposition (globale beste positie). De methoden van de klasse Swarm bevatten alle bewerkingen die nodig zijn in de PSO methode, zoals het initiëren van de posities en de snelheiden, het updaten van de posities en de snelheid en het updaten van de globale beste positie en fitheid.

De parameters van het optimalisatie-algoritme die worden bijgehouden door een object van de klasse Swarm worden gedefinieerd in een object van de klasse Parameters. Deze klasse bundelt de variabelen met betrekking tot de parameters van het detectie-algoritme, eveneens voorziet het de methoden om de parameters in te stellen en weer te geven. De parameters voor de maximaal toegelaten snelheid wordt gefixeerd in het optimalisatie-algoritme. Deze waarde wordt gelijk gesteld aan 0,5 van het bereik van de beslissingsveranderlijken. Deze waarde zorgt voor een goede balans tussen het zoeken naar het optimum in nieuwe gebieden en in reeds bezochte gebieden.

Het algoritme verloopt op de volgende manier. In de main methode van de klasse main, welke het startpunt van het algoritme is, worden de parameters voor het optimalisatie-algoritme ingegeven. Deze worden gebruikt om een nieuw object van de klasse Parameters te maken. Vervolgens wordt een nieuw object van de klasse Swarm gemaakt waaraan het object Parameters wordt gegeven. De methode voor het initialiseren van de zwerm wordt uitgevoerd. Deze methode maakt de zwerm van individuen aan met een willekeurige initiële positie en snelheid en berekent voor elk individu de fitheid. Vervolgens wordt een FOR-lus gestart die volgende stappen herhaalt tot de maximaal toegestane iteraties bereikt zijn. Deze stappen zijn het updaten van de snelheid, het updaten van de positie, het updaten van de fitheden en ten slotte het updaten van de globale beste positie. Dit alles beschrijft de implementatie van de PSO methode voor de eerste fase.

Deze implementatie wordt getest aan de hand van testfuncties voorzien door [23]. Zes testfuncties worden gekozen en deze zijn terug te vinden in tabel 4.1. De eerste vier testfuncties zijn

Tabel 4.1: Testfuncties.

Naam	Functie	Domein	Optimum	Oplossing
Bolfunctie	$f_1(\mathbf{x}) = \sum_{i=1}^{d} x_i^2$	$[-5.12, 5.12]$	$\mathbf{x}^* = (0, ..., 0)$	$f_1(\mathbf{x}^*) = 0$
Rastrigin functie	$f_2(\mathbf{x}) = 10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)]$	$[-5.12, 5.12]$	$\mathbf{x}^* = (0, ..., 0)$	$f_2(\mathbf{x}^*) = 0$
Griewank functie	$f_3(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]$	$\mathbf{x}^* = (0, ..., 0)$	$f_3(\mathbf{x}^*) = 0$
Schwefel functie	$f_4(\mathbf{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{\lvert x_i \rvert})$	$[-500, 500]$	$\mathbf{x}^* = (420.9687, ..., 420.9687)$	$f_4(\mathbf{x}^*) = 0$
Drop-wave functie	$f_5(\mathbf{x}) = -\frac{1+\cos(12\sqrt{x_1^2+x_2^2})}{0.5(x_1^2+x_2^2)+2}$	$[-5.12, 5.12]$	$\mathbf{x}^* = (0, 0)$	$f_5(\mathbf{x}^*) = -1$
Easom functie	$f_6(\mathbf{x}) = -\cos(x_1)\cos(x_2)\exp(-(x_1-\pi)^2 - (x_2-\pi)^2)$	$[-100, 100]$	$\mathbf{x}^* = (\pi, \pi)$	$f_6(\mathbf{x}^*) = -1$

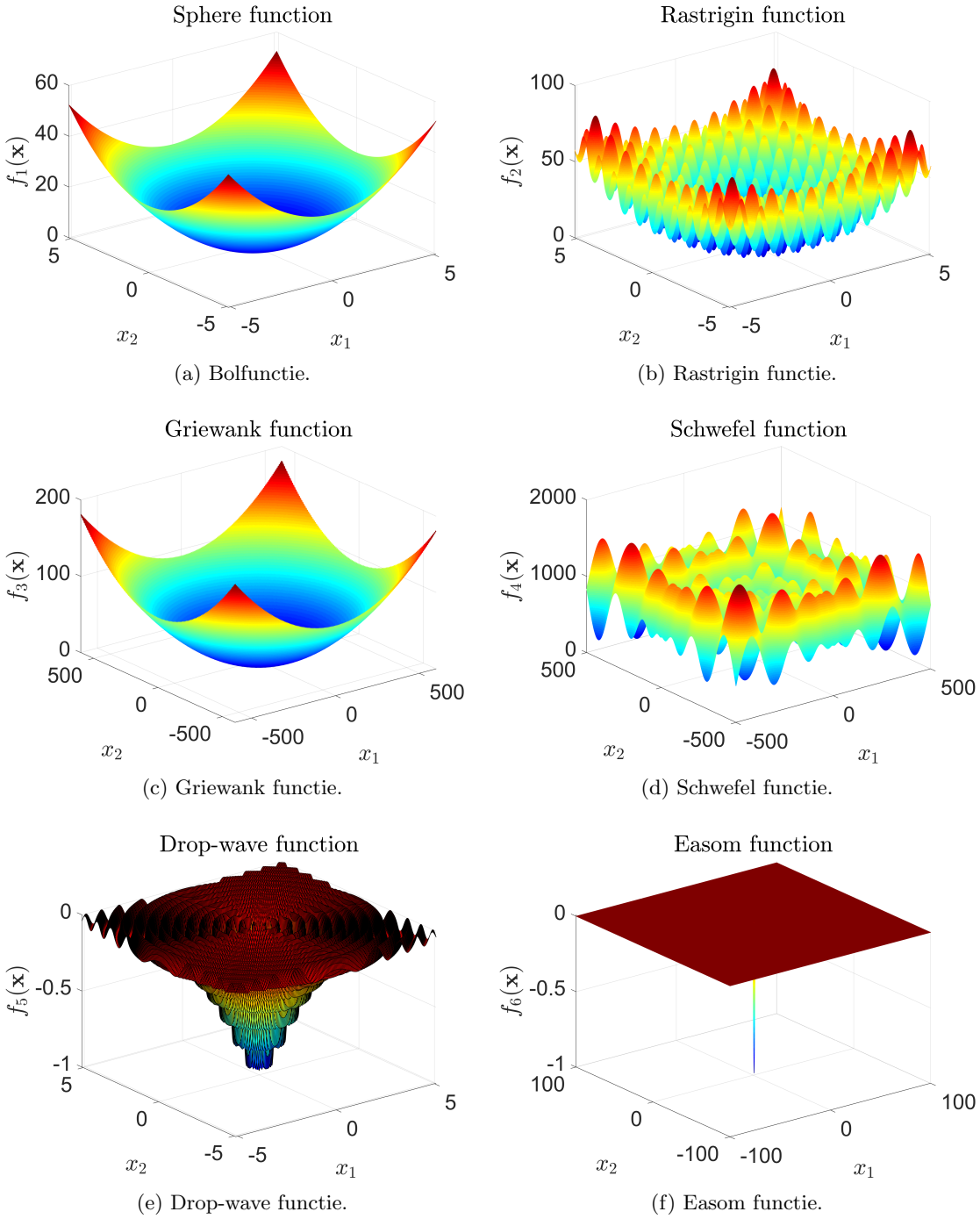
gedefinieerd over de dimensie d dus de dimensie kan vrij gekozen worden. Dit in tegenstelling tot de vijfde en zesde testfunctie waarbij de dimensie gelijk is aan 2. Alle testfuncties voor twee dimensies worden grafisch weergegeven in figuur 4.2.

De eerste functie is de bolfunctie en heeft een duidelijk globaal optimum voor beslissingsveranderlijken gelijk aan nul. De tweede functie is de Rastrigin functie. Deze functie is multimodaal d.w.z. dat de functies meerdere lokale optima heeft waardoor het vinden een globaal optimum moeilijk is. De grafiek voor de Griewank functie laat vermoeden dat deze functie equivalent is aan deze van de bolfunctie. Niets is minder waar, de Griewank functie is eveneens een multimodale functie. Dit wordt geïllustreerd in figuur 4.3 die een gedetailleerde grafiek van de Griewank functie weergeeft. De vierde functie is de Schwefel functie en is eveneens een multimodale functie. De Drop-wave functie is een complexe, multimodale functie. De zesde functie is de Easom functie. Deze functie vertoont een duidelijk minimum dat te vinden is in een klein gedeelte relatief ten opzichte van het zoekdomein.

De eerste vier testfuncties worden getest voor een dimensie gelijk aan 2, 10 en 20. De vijfde en zesde testfunctie logischerwijze enkel voor twee dimensies. De parameters die gebruikt worden om het algoritme te testen zijn gebaseerd op de bevindingen uit [16]. Hierin worden volgende parameters beschouwd als parameters die zorgen voor adequate resultaten. De zwermgrootte wordt gelijk gesteld aan 30. Het inertiegewicht neemt lineair af van 0,9 tot 0,4 tijdens de verschillende iteraties. Het cognitieve en sociale gewicht zijn beiden gelijk aan 2. Voor de maximale iteraties wordt er geopteerd om 10 000 te nemen. Deze waarde wordt enkel genomen voor de testen van 20 dimensies. Voor 10 dimensies wordt een maximum van 1 000 iteraties genomen en voor 2 dimensies worden slecht 100 iteraties genomen.

Het optimalisatie-algoritme wordt 100 keer uitgevoerd voor elke testfunctie en elke dimensie (2, 10 en 20). Vervolgens wordt het gemiddelde, de standaardafwijking en het minimum van de 100 oplossingen genomen. Deze resultaten worden weergegeven in tabel 4.2. Ter herhaling, de oplossing van de eerste vier functies is gelijk aan 0 en van de vijfde en zesde functie is de oplossing -1.

Voor twee dimensies lijkt het algoritme geen problemen te hebben met vinden van de correcte oplossing. Enkel voor de vierde testfunctie, de Schwefel functie lijkt het algoritme moeilijkheden te hebben. Uit het minimum blijkt dat het algoritme wel in staat is om de optimale oplossing te bereiken. Verder bedraagt het gemiddelde 19,01 en de standaardafwijking 43,62, wat er op wijst dat het algoritme toch enige moeite heeft om steeds de optimale oplossingen te vinden. De oorzaak is de multimodaliteit van de functie.



(a) Bolfunctie.

(b) Rastrigin functie.

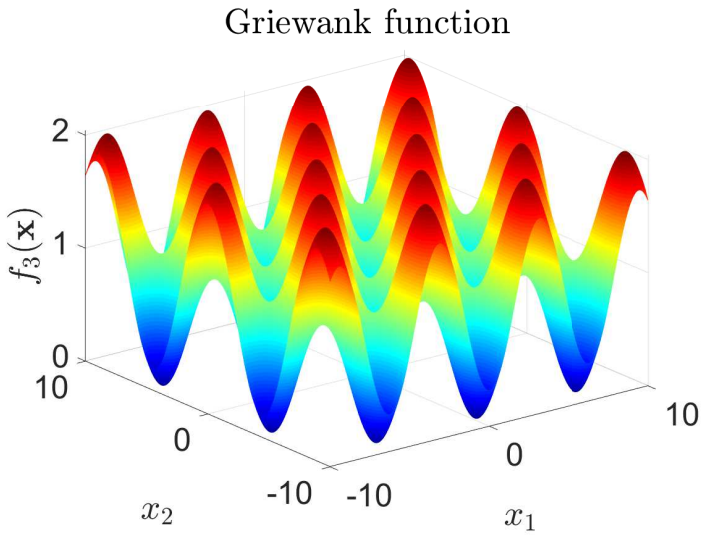
(c) Griewank functie.

(d) Schwefel functie.

(e) Drop-wave functie.

(f) Easom functie.

Figuur 4.2: Testfuncties.



Figuur 4.3: Griewank functie in detail.

Tabel 4.2: Resultaten voor de verschillende testfuncties en dimensies (gemiddelde, standaardafwijking, minimum).

Functie	**2 dimensies**	**10 dimensies**	**20 dimensies**
$f_1(\mathbf{x})$	9,760e-11 2,946e-10 9,894e-14	1,302e-25 3,979e-25 4,961e-29	9,530e-108 5,489e-107 3,198e-115
$f_2(\mathbf{x})$	7,636e-7 2,199e-6 3,118e-11	3,724 1,898 3,379e-12	9,104 3,769 2,985
$f_3(\mathbf{x})$	1,252e-2 1,097e-2 2,902e-7	9,966e-2 5,109e-2 1,478e-2	3,334e-2 2,951e-2 0
$f_4(\mathbf{x})$	1,901e+1 4,362e+1 2,546e-5	3,482e+2 1,559e+2 1,273e-4	2,902e+2 1,555e+2 1,273e-4
$f_5(\mathbf{x})$	−0,9941 1,831e-2 −1	n.v.t.	n.v.t.
$f_6(\mathbf{x})$	−1 9,451e-8 −1	n.v.t.	n.v.t.

Voor 10 en 20 dimensies zijn dezelfde tendensen waar te nemen. Het algoritme heeft geen moeite met het vinden van de optimale oplossing voor de bolfunctie en Griewank functie. De Rastrigin en Schwefel functie zijn moeilijker om te optimaliseren. Indien de resultaten vergeleken worden met de literatuur zijn deze allesbehalve van mindere kwaliteit. Pant, Thangaraj en Abraham ([16]) hebben gelijkaardige testen uitgevoerd. Dezelfde parameters, 20 dimensies en eveneens is de Rastrigin functie gebruikt. Zij bekomen een gemiddelde waarde van 22,339158, hoewel dit werk een gemiddelde waarde van 9,104 behaalt.

Het optimalisatie-algoritme vertoont veelbelovende resultaten. Eveneens worden de gebruikte parameters aanschouwd als een goede combinatie voor het algoritme. De invloed van de parameters op de optimalisatie van het detectie-algoritme kunnen dienen als opvolgend werk. Ook bestaat de mogelijk om de werking van optimalisatie-methode aan te passen zodat deze nog beter omgaat met multimodale functies.

Tweede fase

De tweede fase van de implementatie wijzigt vier grote zaken aan het reeds ontwikkeld optimalisatie-algoritme. De werking van de main methode in de Main klasse wordt aangepast zodanig het optimalisatie-algoritme vanuit de command-line-interface gebruikt kan worden. Dit leidt tot de creatie van een nieuwe klasse, namelijk PSO. Eveneens wordt de Problem klasse aangepast aan het detectie-algoritme. De doelfunctie wordt vervangen door de verschillende stappen van het detectie-algoritme. De beslissingsveranderlijken zijn nu een selectie van de parameters van het detectie-algoritme. Daardoor dient de Position- en Velocity klasse ook aangepast te worden. De laatste aanpassing is de creatie van de *Utils* klasse die verantwoordelijk is voor het initiëren en updaten van de waarde van de positie en de snelheid voor elke beslissingsveranderlijke.

De parameters die geoptimaliseerd worden, zijn de waarde van k, het gewicht voor het tijdsgebonden en naamsgebonden verband, de pruningdrempel, het minimaal aantal requests, alle gewichten voor de WOWA operator. Dit zijn tevens de belangrijkste parameters van het detectie-algoritme (zie paragraaf). Er wordt niet gekozen om gebruik te maken van het gewogen gemiddelde of de OWA operator omdat deze eveneens kunnen gerealiseerd worden met de WOWA operator. Voor de WOWA operator wordt er nu niet met genormaliseerde scores gewerkt in tegenstelling tot de optimalisatie van de gewichten de WOWA operator (zie paragraaf 4.3.1).

Voor de waarde van k wordt een minimumwaarde van 10 gekozen en een maximumwaarde van 100. Deze keuze berust op de resultaten uit paragraaf 2.5.3. Eveneens wordt er gekozen om met stappen van 10 te werken. Dit leidt ertoe dat 10, 20, 30, 40, 50, 60, 70, 80, 90 en 100 de mogelijke waarden voor k zijn. De keuze van de stappen komt voort uit het feit dat de berekening van de grafen computationeel intensief is. Al moeten deze in het optimalisatie-algoritme slechts één maal worden berekend voor elke k, toch wordt er tijd gewonnen.

Voor de gewichten telt dat het bereik begrensd is tussen 0 en 1. Ook moet de som van de gewichten voor zowel de verbanden, als voor beide vectoren van de WOWA operator steeds gelijk zijn aan 1. Daarom wordt na elke stap in het optimalisatie-algoritme die de gewichten wijzigt, de gewichten genormaliseerd. Elk gewicht wordt gedeeld door de som van de respectievelijke gewichten.

Voor de pruningdrempel wordt de z-score verkozen boven de absolute waarde, daar het makke-

lijker is om met de z-score te werken. De pruningdrempel kan variëren tussen -1 en 1. Dit stemt in absolute waarden overeen met een variatie van de pruningdrempel tussen het gemiddelde min de standaardafwijking en het gemiddelde plus de standaardafwijking. Voor het minimaal aantal requests wordt geopteerd voor getallen tussen 1 en 20, gehele getallen weliswaar.

De klasse Position en Velocity worden aangepast zodanig dat ze de correcte gegevens kunnen voorstellen t.t.z. de combinatie van parameters die zonet zijn beschreven. Zo wordt er ook een nieuwe klasse Utils gemaakt die de nodige methodes bezit voor het initiëren en updaten van alle parameters rekening houdende met hun specificiteit.

De klasse Problem wordt gewijzigd zodat alle stappen van het algoritme doorlopen worden, zowel de uitvoering van de Batch Processor als de Server met de nodige parameters. Nadat het klassement wordt verkregen, wordt er een ROC opgesteld waaruit de AUC wordt bepaald.

De PSO is een nieuwe klasse die de rol van de Main uit de eerste fase overneemt. De PSO klasse is dus verantwoordelijk voor het initiëren van de zwerm, het updaten van de globale positie en de uitvoering van de FOR-lus. De FOR-lus omvat nog steeds het updaten van respectievelijk de snelheid, de positie, de fitheden en de globale beste positie. De vernieuwde Main klasse laat de interactie toe met de analist via de command-line-interface.

4.4.2 Integratie

Zoals reeds vermeld wordt het optimalisatie-algoritme geïntegreerd in het detectie-algoritme als een nieuwe package, de PSO package. De PSO package telt in totaal 1547 lijnen aan code. Deze code is terug te vinden in bijlage A. Het optimalisatie-algoritme wordt bestuurd via de command-line-interface. Het optimalisatie-algoritme heeft als input een configuratiebestand. Dit configuratiebestand bevat alle nodige parameters voor het optimalisatie-algoritme. De output van het optimalisatie-algoritme zijn de optimale parameters voor het detectie-algoritme met de overeenstemmende waarde van de AUC.

Een typisch commando voor het optimalisatie-algoritme ziet er als volgt uit:

```
./pso.sh -i <input configuration file>
```

Het configuratiebestand ziet er als volgt uit:

{"input_file":"/home/student/data/logfile.txt","output_dir":"/home/student/data/output/",
"format":"json","n_apt_tot":"4","swarm_size":"30","inertia_weight_1":"0.9",
"inertia_weight_2":"0.4","inertia_weight_ratio":"-1","cognitive_weight":"2","social_weight":"2",
"max_iteration":"100"}

Dit bestand bestaat uit de locatie van het logbestand, de *output directory* voor de resultaten, het formaat van het logbestand, aantal APTs die aanwezig zijn in de log, zwermgrootte, inertiegewicht, cognitief gewicht, sociaal gewicht en het maximum aantal iteraties. Het inertiegewicht kan op drie verschillende manieren worden bepaald.

Constant gewicht Het gewicht is hetzelfde gedurende de verschillende iteraties. Dit wordt bereikt door tweemaal hetzelfde gewicht in te geven bij het inertiegewicht 1 en 2. Voor de inertiegewichtverhouding moet een waarde van 1 worden gegeven.

Rekenkundige rij Het gewicht neemt lineair af tussen twee bepaalde waardes gedurende de verschillende iteraties. De startwaarde hoort bij inertiegewicht 1 en de stopwaarde bij inertiegewicht 2. Bij de inertiegewichtverhouding komt -1.

Meetkundige rij Het gewicht neemt af met een bepaalde quotiënt. De beginwaarde wordt zowel bij inertiegewicht 1 als inertiegewicht 2 gegeven. Bij de verhouding van het inertiegewicht komt de gewenste quotiënt.

4.5 Conclusie

In totaal zijn er 7 parameters toegevoegd aan het detectie-algoritme evenals bevat het detectie-algoritme een nieuwe package. Een boolean is toegevoegd die aanduidt of sing moet uitgevoerd worden. De WOWA operator is geïntegreerd in het detectie-algoritme. Hiervoor zijn er 5 parameters aan het algoritme toegevoegd, een parameter die aanduidt of de WOWA operator of het gewogen gemiddelde moet toegepast worden, een parameter die beslist of er met oorspronkelijke of genormaliseerde scores wordt gewerkt voor de WOWA operator en drie parameters die de gewichten van de WOWA operator zijn met betrekking tot de OWA operator.

Vervolgens is de optimalisatiemethode toegevoegd die de gewichten van de WOWA operator kan trainen. De implementatie van de optimalisatie is uitgevoerd door een onderzoeker van de Research Unit Cyber Defense van de Koninklijke Militaire School. De parameters van het Genetisch Algoritme die zorgen voor de optimalisatie liggen vast in het detectie-algoritme. Belangrijk voor de werking is dat het logbestand vrijwillig wordt besmet met APTs. Een extra parameter is toegevoegd die aangeeft of de gewichten van de WOWA operator geoptimaliseerd moeten worden.

De optimalisatie van het volledige algoritme is toegevoegd als extra package van het detectie-algoritme. De optimalisatiemethode voor het volledige detectiealgoritme is de Particle Swarm Optimization methode. In een eerste fase is deze methode geïmplementeerd om de werking ervan te testen. Vervolgens is de implementatie aangepast aan de werking van het detectie-algoritme. Het is mogelijk om het optimalisatie-algoritme aan te sturen vanuit de command-line-interface. Als input heeft het algoritme nood aan een configuratiebestand dat onder andere de parameters voor PSO, het logbestand en de output directory voor de resultaten bevat.

5. Resultaten van de optimalisatie

5.1 Inleiding

Nu de optimalisatiemethoden zijn geïmplementeerd en geïntegreerd in het detectie-algoritme, is het noodzakelijk om de optimale parameters te berekenen. De optimale parameters worden berekend voor twee verschillende testomgevingen zodanig beide resultaten vergeleken kunnen worden. Voor beide omgevingen worden eerst de parameters van Gilon[2] gebruikt om een indicatie van de AUC te hebben. Het is de bedoeling dat beide optimalisatiemethoden een betere AUC bekomen.

De parameters van Gilon worden als startpunt gebruikt voor de optimalisatie van de gewichten van de WOWA operator. In plaats van het gewogen gemiddelde toe te passen als aggregatietechniek voor het klassement, wordt de WOWA operator toegepast waarvoor de gewichten worden geoptimaliseerd met een Genetisch Algoritme. De andere parameters behouden de waardes die werden bekomen door Gilon.

Aansluitend wordt het algoritme geoptimaliseerd in de algemene zin t.t.z. de parameters die werden besproken in paragraaf 4.4.1. Deze optimalisatie wordt uitgevoerd door de nieuwe PSO package die aan het detectie-algoritme is toegevoegd.

Ten slotte worden de bekomen optimale parameters voor de ene omgeving getest in de andere omgeving om de parameters te valideren. Dit wordt zowel toegepast voor de optimale gewichten van de WOWA operator als voor de optimale parameters voor het gehele detectie-algoritme.

5.2 Testomgeving

Als testomgevingen worden er twee verschillende subnets gekozen uit de log van een grote organisatie. Opnieuw wordt er verondersteld dat de log niet besmet is met APTs. De APTs worden dus vrijwillig toegevoegd aan de logbestanden. De APTs krijgen dezelfde eigenschappen als de APTs die werden gebruikt tijdens de validatie van het detectie-algoritme (zie par. 2.5.4). Er worden dus 8 APTs toegevoegd, 4 periodieke en 4 gegevensstroom gebaseerde.

De eerste testomgeving telt 43 gebruikers die verantwoordelijk zijn voor 662 120 requests (inclusief de verbindingen van de APTs). In het totaal zijn er 7 283 domeinen (inclusief C2-domeinen van de APTs) bezocht. Opnieuw wordt dezelfde tijdspanne onderzocht die gebruikt werd voor de validatie van het detectie-algoritme (zie par. 2.5.4). De periodieke APTs met een periode van

[2]Met de parameters van Gilon worden de parameters bedoeld die Gilon aanraadt als startpunt voor de analist. Deze parameters zijn terug te vinden in paragraaf 2.5.3. Belangrijk om op te merken is dat vanaf nu clustering niet meer wordt toegepast.

24 uur, 12 uur, 6 uur en 1 uur zijn verantwoordelijk voor respectievelijk 9, 19, 39 en 239 requests. De gegevensstroom gebaseerde APTs verschillen enkel in de minimale periode die noodzakelijk is tussen twee verschillende verbindingen van de APT. Voor minimale periodes van 4 uur, 2 uur, 1 uur en 30 minuten maken de APTs respectievelijk 9, 19, 27 en 39 verbindingen.

De tweede testomgeving bestaat uit 35 gebruikers die zorgen voor 428 448 requests (inclusief APTs) over dezelfde tijdspanne als de voorgaande omgeving. Een totaal van 4 837 domeinen (inclusief APTs) zijn terug te vinden. De periodieke APTs met een periode van 24 uur, 12 uur, 6 uur en 1 uur zijn verantwoordelijk voor respectievelijk 9, 19, 39 en 239 requests en de gegevensstroom gebaseerde APTs met minimale periodes van 4 uur, 2 uur, 1 uur en 30 minuten maken respectievelijk 9, 14, 15 en 33 verbindingen.

De optimalisatie van het algoritme in de algemene zin werd gestart voor een testomgeving van 66 gebruikers. Deze optimalisatie is echter onderbroken door een gebrek aan tijd. Een eerste schatting van de rekentijd bedroeg 27 dagen. Indien meer rekenkracht en meer tijd beschikbaar zou zijn, zou het interessant zijn om een grotere groep gebruikers te onderzoeken aangezien deze nog beter de realiteit weerspiegelt dan een kleinere groep.

Eveneens kan de manier worden aangepast waarop de gebruikers worden geselecteerd. Momenteel worden de gebruikers gekozen door een bepaald subnet te selecteren. Meestal berust de indeling van de subnets in organisaties op een logische keuze. Het is mogelijk om per gebouw, verdiep of ruimte te werken. De indeling kan ook gebaseerd zijn op de verschillende categorieën van werknemers in de organisatie. Deze keuze zorgt ervoor dat de gebruikers in een bepaald subnet mogelijk dezelfde karakteristieken vertonen. Deze karakteristieken zijn bijvoorbeeld de momenten die de gebruikers actief zijn (dezelfde werkuren) of gelijkaardige webpagina's die bezocht worden (hetzelfde werkdomein).

De keuze van het subnet kan dus een invloed hebben op de optimale parameters. De mogelijkheid bestaat dat de optimale parameters te veel zijn afgestemd op de omgeving waarin ze worden bepaald. In opvolgend werk zou het mogelijk zijn om de gebruikers willekeurig te selecteren uit het log-bestand. Een selectie van gebruikers die gebruikt wordt als testomgeving om de optimale parameters te berekenen, en een andere selectie die dient als validatieomgeving van de optimale parameters. Op die manier zouden gebruikers met verschillende karakteristieken gekozen worden om zowel de optimale parameters te berekenen als te valideren.

5.3 Resultaten

Als ijkingspunt worden eerst de parameters bekomen door Gilon op beide omgevingen toegepast. De parameters dienen als startpunt voor de analist en garanderen een goede detectie van APTs. De parameters kunnen teruggevonden worden in tabel 2.1. Er dient opgemerkt te worden dat de maximale clustergrootte niet meer van toepassing is aangezien clustering niet meer wordt uitgevoerd.

Voor de eerste omgeving wordt met deze parameters een AUC bekomen van 0,9201 en voor de twee omgeving 0,9249. Deze combinatie van parameters leidt inderdaad tot een adequate detectie van de APTs. Het valt op dat de beide waardes gelijkaardig zijn.

5.3.1 WOWA optimalisatie

Voor de WOWA optimalisatie is de keuze van de parameters gebaseerd op de aanbevelingen door Alexandre Croix [22]. Een populatiegrootte van 100 wordt gekozen. De recombinatieverhouding bedraagt 0,60 en de mutatiegraad 0,15. De selectiemethode is RWS. Het maximum aantal iteraties bedraagt 110. De initiële populatie wordt volledig willekeurig gekozen.

Aangezien enkel de parameters met betrekking tot de WOWA operator worden geoptimaliseerd, is het noodzakelijk om de andere parameters vast te leggen. Deze parameters worden gelijk gesteld aan de parameters bekomen door Gilon. Ook valt op te merken dat, zoals reeds vermeld in paragraaf 4.3.1, er met genormeerde scores wordt gewerkt.

Tabel 5.1: Resultaat voor WOWA optimalisatie.

	Ouders	**Kinderen**	**Aantal requests**	**Hoogste score**	**Middelste score**	**Laagste score**
Eerste omgeving	0,3528	0,1516	0,4956	0,1978	0,7991	0,0031
Tweede omgeving	0,2142	0,4238	0,3620	0,5026	0,4969	0,0005

De resultaten worden weergegeven in tabel 5.1. De respectievelijke parameters voor elke omgeving leiden tot een AUC van 0,9574 voor de eerste omgeving en 0,9422 voor de tweede omgeving. Beide resultaten zijn beter dan degene die werden bekomen met de aanbevolen parameters van Gilon. Voor de eerste omgeving is de toename groter dan voor de tweede omgeving.

5.3.2 Volledige optimalisatie

De parameters nodig voor de volledige optimalisatie zijn gebaseerd op resultaten uit [16]. Deze parameters zijn eveneens getest in 4.4.1 en vertonen betrouwbare resultaten. Vermits de rekenkracht en rekentijd beperkt is, wordt het maximaal aantal iteraties verminderd naar 100. De impact van deze beslissing is ongekend. In opvolgend werk kan de invloed van deze beslissing bestudeerd worden, eveneens de invloed van de andere parameters op de optimalisatie van het detectie-algoritme. De uiteindelijke parameters zijn een zwermgrootte van 30, een inertiegewicht dat lineair afneemt van 0,9 tot 0,4, een cognitief en sociaal gewicht gelijk aan 2 en het maximaal aantal iteraties gelijk aan 100.

Tabel 5.2: Resultaat voor volledige optimalisatie

	Eerste Omgeving	**Tweede Omgeving**
Waarde van k	70	50
Gewicht: tijdsgebonden verband	0,1944	0,3810
Gewicht: naamsgebonden verband	0,8056	0,6190
Pruningdrempel (z-score)	0,0376	0,0870
Minimaal aantal requests	9	9
Gewicht klassement: score ouders	0,3403	0,5390
Gewicht klassement: score kinderen	0,0132	0,3740
Gewicht klassement: score aantal requests	0,6465	0,0870
Gewicht klassement: hoogste score	0,0078	0,0759
Gewicht klassement: middelste score	0,4579	0,6532
Gewicht klassement: laagste score	0,5343	0,2709

De bekomen parameters worden voorgesteld in tabel 5.2. Voor de eerste omgeving leiden de respectievelijke optimale parameters tot een AUC van 0,9840. Ook voor de tweede omgeving

leiden hun optimale parameters tot een AUC van 0,9840. Beide waarden voor de AUC zijn toegenomen in vergelijking met de optimalisatie van de gewichten van de WOWA Operator.

5.4 Bespreking

Zoals reeds vermeld heeft het minimaal aantal requests per gebruiker en per domein een niet te onderschatten invloed op de waarde van de AUC. Voor beide omgevingen is het minimaal aantal verbinden dat een APT maakt, gelijk aan 9. Het is dus logisch dat het optimalisatie-algoritme een waarde van 9 bekomt voor het minimaal aantal requests. Deze waarde zorgt ervoor dat alle domeinen, waarvoor het minimum niet gerespecteerd is, uit de graaf verdwijnen. Alle domeinen, die verdachter zijn dan de APTs én waarvoor het minimum niet gerespecteerd is, verdwijnen uit het klassement. Bijgevolg komen de APTs hoger in het klassement wat eindelijk leidt tot een betere AUC.

Dus om de resultaten beter te kunnen begrijpen, zijn nog enkele andere combinaties van parameters gebruikt in beide omgevingen. Een samenvatting van alle gebruikte combinaties van parameters worden weergegeven in tabel 5.3 en alle AUC waardes die bekomen worden met deze parameters worden weergegeven in tabel 5.4.

Wat volgt is een korte uitleg voor elke combinatie van parameters uit tabel 5.3:

1. De parameters bekomen door Gilon die dienen als startpunt voor de analist.
2. De parameters van Gilon waarvoor het minimaal aantal requests is aangepast naar 9 om een betere vergelijking te kunnen maken met de resultaten van de volledige optimalisatie.
3. De parameters bekomen uit de optimalisatie van de WOWA operator voor de eerste omgeving en vertrekkende vanaf de eerste combinatie van parameters (zie paragraaf 5.3.1).
4. De parameters bekomen uit de optimalisatie van de WOWA operator voor de tweede omgeving en vertrekkende vanaf de eerste combinatie van parameters (zie paragraaf 5.3.1).
5. De parameters bekomen uit de optimalisatie van de WOWA operator voor de eerste omgeving en vertrekkende vanaf de tweede combinatie van de parameters zodoende een betere vergelijking kan gemaakt worden met de resultaten van de volledige optimalisatie.
6. De parameters bekomen uit de optimalisatie van de WOWA operator voor de tweede omgeving en vertrekkende vanaf de tweede combinatie van de parameters zodoende een betere vergelijking kan gemaakt worden met de resultaten van de volledige optimalisatie.
7. De parameters bekomen uit de volledige optimalisatie voor de eerste omgeving (zie paragraaf 5.3.2).
8. De parameters bekomen uit de volledige optimalisatie voor de tweede omgeving (zie paragraaf 5.3.2).

Nu is het mogelijk om de resultaten te vergelijken met een gelijke waarde voor het minimaal aantal requests. Voor de eerste omgeving worden de resultaten uit tabel 5.4 voor de 2de, 5de en 7de combinaties van parameters vergeleken. De bekomen resultaten zijn respectievelijk 0,9543, 0,9657 en 0,9840. De optimalisatie van de gewichten voor de WOWA Operator zorgt voor

een verbetering van de waarde van de AUC ten opzichte van het resultaat bekomen door de parameters van Gilon. De volledige optimalisatie zorgt op zijn beurt voor een verbetering ten opzichte van de optimalisatie van de WOWA Operator. Hetzelfde is op te merken voor de tweede omgeving waarvoor de resultaten van de 2de, 6de en 8ste combinatie van parameters respectievelijk 0,9600, 0,9792 en 0,9840 zijn.

De volledige optimalisatie van het detectie-algoritme voor de eerste omgeving duurt 5 dagen, 22 uur en 18 minuten. Deze tijd is exclusief de berekeningen van de grafen per gebruiker en per verband. De berekening door de Batch Processor van de grafen voor alle nodige waardes van k gebeurt op voorhand. De berekening van de grafen voor k gelijk aan 10 duurt 34 minuten. Daarentegen duurt het 4 uur en 13 minuten om de grafen te berekenen voor een waarde van k gelijk aan 100. Om de grafen voor alle verschillende waardes van k te berekenen heeft het detectie-algoritme ongeveer 1 dag nodig. Bijgevolg neemt de volledige optimalisatie bij benadering 7 dagen tijd in beslag.

De bepaling van de optimale gewichten van de WOWA operator is een stuk sneller. Het detectie-algoritme heeft in 3 minuten en 42 seconden de optimale gewichten voor de WOWA operator berekend en toegepast om het klassement te bekomen. Deze tijd is natuurlijk exclusief de berekening van de grafen per gebruiker en per verband.

De optimalisatie van het volledige algoritme leidt dan wel tot een betere waarde van de AUC maar daar staat tegenover dat de rekentijd aanzienlijk hoger ligt ten opzichte van de optimalisatie van de gewichten van de WOWA operator. Dit was reeds voorspeld aangezien bij de volledige optimalisatie het detectie-algoritme meerdere malen volledig wordt doorlopen en bij de optimalisatie van de WOWA gewichten het slechts één maal wordt doorlopen. De optimalisatie van WOWA gewichten maakt namelijk gebruik van de finale graaf die het detectie-algoritme produceert om de gewichten te optimaliseren.

Voor de optimalisatie van het volledige algoritme valt op dat het niet noodzakelijk is om een waarde van 100 voor k te hebben om goede resultaten te bekomen. Indien de optimale parameters bekomen door de volledige optimalisatie (7de en 8ste combinatie van parameters) met elkaar vergeleken worden, vallen volgende relaties op. Het naamsgebonden verband krijgt een belangrijker gewicht dan het tijdsgebonden verband. Dit werd ook al opgemerkt door Gilon (zie paragraaf 2.5.3). Eveneens wordt een klein positieve waarde als pruningdrempel verkozen. Dit leidt tot de verwijdering van meer verbanden in vergelijking met de parameters van Gilon. De waarde van het minimaal aantal requests werd eerder al uitgelegd. Voor de gewichten van de WOWA operator die gelinkt kunnen worden met OWA, wordt vastgesteld dat de hoogste score weinig invloed heeft. Het zijn vooral de middelste en de laagste score die belangrijk zijn voor het klassement.

De verschillende optimale gewichten voor de WOWA operator hebben allen gemeen dat er een kleine waarde wordt toegekend aan het gewicht met de laagste score, beter gezegd de kleinste waarschijnlijkheid aangezien er genormaliseerde scores worden gebruikt.

Een duidelijk verschil is op te merken tussen het gebruik van de absolute of genormaliseerde scores. Voor de absolute scores heeft de grootste score weinig invloed op het klassement. Daarentegen voor de genormaliseerde scores heeft de kleinste waarde weinig invloed op het klassement. Dit gedrag is perfect te verklaren door de transformatie tussen de absolute en de genormaliseerde

score. Deze transformatie zorgt ervoor dat de grote absolute scores overeenstemmen met kleine waarschijnlijkheden of genormaliseerde scores.

De optimale parameters van de eerste omgeving toegepast op de tweede omgeving, leiden tot een AUC van 0,9687. Omgekeerd leiden de optimale parameters van de tweede omgeving tot een AUC van 0,9456 voor de eerste omgeving. Beide optimale parameters leiden tot een minder goed resultaat in de andere omgeving. De kans bestaat dat de optimale parameters te veel zijn afgestemd op de omgeving waarin ze bepaald zijn. Hierdoor leidt de toepassing van de parameters in een andere omgeving tot minder goede resultaten. De vermoedelijke oorzaak is de manier waarop de testomgevingen gekozen zijn. De manier van de subnets zorgt ervoor dat eenzelfde klasse van gebruikers wordt bestudeerd in plaats van een gevarieerde groep.

Tabel 5.3: Combinaties van parameters.

Combinatie van parameters	**1**	**2**	**3**[1]	**4**[1]	**5**[1]	**6**[1]	**7**	**8**
Waarde van k	100	100	100	100	100	100	70	50
Gewicht: tijdsgebonden verband	0,1	0,1	0,1	0,1	0,1	0,1	0,1944	0,3810
Gewicht: naamsgebonden verband	0,9	0,9	0,9	0,9	0,9	0,9	0,8056	0,6190
Pruningdrempel (z-score)	0,0	0,0	0,0	0,0	0,0	0,0	0,0376	0,0023
Minimaal aantal requests	5	9	5	5	9	9	9	9
Gewicht klassement: score ouders	0,4	0,4	0,3528	0,2142	0,5318	0,0534	0,3403	0,5390
Gewicht klassement: score kinderen	0,4	0,4	0,1516	0,4238	0,0566	0,4748	0,0132	0,3740
Gewicht klassement: score aantal requests	0,2	0,2	0,4956	0,3620	0,4116	0,4718	0,6465	0,0870
Gewicht klassement: hoogste score	n.v.t.	n.v.t.	0,1978	0,5026	0,2335	0,3642	0,0078	0,0759
Gewicht klassement: middelste score	n.v.t.	n.v.t.	0,7991	0,4969	0,7301	0,5589	0,4579	0,6532
Gewicht klassement: laagste score	n.v.t.	n.v.t.	0,0031	0,0005	0,0364	0,0769	0,5343	0,2709

[1] Voor deze combinatie van parameters werden de scores voor het klassement genormaliseerd.

Tabel 5.4: De AUC voor de verschillende combinaties van parameters en voor de twee testomgevingen.

Combinatie	**Testomgeving 1**	**Testomgeving 2**
1	0,9201	0,9249
2	0,9543	0,9600
3	0,9574	0,9290
4	0,9234	0,9422
5	0,9657	0,9265
6	0,9392	0,9792
7	0,9840	0,9687
8	0,9456	0,9840

5.5 Conclusie

Twee verschillende testomgevingen zijn gekozen op basis van hun subnet. De eerste omgeving telt 43 gebruikers, de tweede heeft er 35. Beide omgevingen worden vrijwillig besmet met APTs die dezelfde eigenschappen hebben.

Voor beide omgevingen zijn zowel de optimale gewichten voor de WOWA operator als de volledige optimale parameters bepaald. De optimale gewichten voor de WOWA operator zorgen voor een stijging van de AUC in vergelijking met de parameters van Gilon. De optimale parameters van het detectie-algoritme laten de waarde van AUC nog verder toenemen. Een belangrijk

verschil tussen beide optimalisatiemethoden is de rekentijd. Deze is beduidend hoger voor de volledige optimalisatie aangezien het detectie-algoritme meerdere malen moet doorlopen worden.

Uit de validatie van de optimale parameters blijkt dat deze sterk zijn afgestemd op de omgeving waarin ze gevonden zijn. Als de rekenkracht en rekentijd het toelaat, kan in opvolgend werk een grotere gevarieerdere groep gebruikers worden onderzocht zodanig de optimale parameters niet zijn afgestemd op de omgeving.

6. Conclusie

6.1 Inleiding

Het laatste hoofdstuk is gewijd aan de conclusie van de masterproef. In de eerste paragraaf wordt de eigen bijdrage aan het detectie-algoritme kort samengevat. Vervolgens worden de verschillende mogelijkheden voor een opvolgend werk samengebundeld. Het hoofdstuk wordt afgesloten met het algemeen besluit van de masterproef.

6.2 Eigen bijdragen

De eigen bijdragen aan het detectie-algoritme waren gefocust op de optimalisatie van het detectie-algoritme. Twee verschillende optimalisatiemethoden zijn toegevoegd aan het detectie-algoritme. De eerste optimalisatiemethode had als doel om de gewichten van de WOWA operator te optimaliseren. De tweede methode zorgde voor de optimalisatie van het volledige algoritme.

Verschillende parameters zijn toegevoegd aan het algoritme. De eerste parameter die werd toegevoegd, beslist of clustering wordt toegepast. Deze keuze zorgde ervoor dat het detectie-algoritme beduidend won aan rekentijd met het behoud van de finale resultaten.

Om de gewichten van de WOWA operator te kunnen optimaliseren, moest de WOWA operator eerst worden toegevoegd aan het detectie-algoritme. De implementatie van de WOWA operator van T. Debatty [19] werd geïntegreerd in het detectie-algoritme. De nodige parameters werden toegevoegd zodanig de WOWA operator gebruikt kon worden. Een eerste parameter duidt aan of het klassement opgesteld wordt aan de hand van het gewogen gemiddelde of de WOWA operator als aggregatietechniek. Drie andere nieuwe parameters maken het mogelijk om de gewichten van de WOWA operator met betrekking tot de OWA operator aan het detectie-algoritme te voorzien.

Het optimalisatie-algoritme voor de WOWA operator had nood aan genormaliseerde scores. Een nieuwe parameter werd toegevoegd die het mogelijk maakt om de WOWA operator toe te passen met genormaliseerde scores. Zo was het detectie-algoritme klaar om het optimalisatie-algoritme voor de gewichten van de WOWA operator te integreren. De implementatie van A. Croix [22] werd gebruikt. Dit optimalisatie-algoritme is een Genetisch Algoritme dat de optimale gewichten van de WOWA operator bepaald zodoende de AUC van het detectie-algoritme gemaximaliseerd wordt. Dit optimalisatie-algoritme werd geïntegreerd met vaste waarden voor de parameters van de optimalisatiemethode. Een nieuwe parameter werd aan het detectie-algoritme toegevoegd die beslist of de optimalisatie van de gewichten wordt uitgevoerd. Als er wordt beslist om de optimalisatie uit te voeren, berekent het detectie-algoritme het klassement met de bekomen geoptimaliseerde gewichten.

Voor de optimalisatie van het volledige algoritme werd een optimalisatiemethode geïmplementeerd en toegevoegd als nieuwe package aan het detectie-algoritme. De optimalisatiemethode die werd toegepast is de Particle Swarm Optimization. In een eerste fase werd deze optimalisatiemethode geïmplementeerd om analytische doelfuncties met reële beslissingsveranderlijken te optimaliseren. Deze fase was noodzakelijk om de implementatie te valideren aan de hand van testfuncties. Eenmaal de optimalisatiemethode gevalideerd was, startte de tweede fase van de implementatie. Deze fase had als doel om het optimalisatie-algoritme aan te passen aan de werking van het detectie-algoritme. Het uiteindelijk optimalisatie-algoritme wordt aangestuurd via de command-line-interface en een configuratiebestand. Dit bestand bevat onder andere de parameters van het PSO algoritme en het logbestand waarvoor de parameters geoptimaliseerd worden. De output van het optimalisatie-algoritme zijn de optimale parameters met de overeenstemmende AUC waarde.

6.3 Opvolgend werk

Reeds verschillende mogelijkheden zijn gegeven voor een opvolgend werk. Deze worden allemaal gebundeld in deze paragraaf. Een eerste mogelijkheid is de aanpassing van clustering. Nu werd geopteerd om clustering niet meer uit te voeren in het detectie-algoritme. Dit zorgt voor dezelfde eindresultaten in een kortere rekentijd. Een andere aanpak bestaat erin om clustering aan te passen zodanig het een meerwaarde is voor het detectie-algoritme. Momenteel leidt clustering tot zowel een aantal heel kleine als een aantal heel grote clusters. Hierdoor is het filteren op de clustergrootte niet zo evident aangezien ofwel de kleine clusters behouden blijven ofwel alle clusters. De kleine clusters zijn meestal geïsoleerde domeinen. Het valt voor dat het domein van een C2 van een APT tot een grote cluster behoort alhoewel de verbanden met de andere domeinen zwak zijn. Clustering zou zodanig moeten aangepast worden dat de C2 domeinen niet meer tot de grote clusters behoren maar als geïsoleerde domeinen verschijnen.

Als opvolgend werk zou een grondige studie mogelijk zijn over de invloed van de parameters van de optimalisatiemethode. Voor de optimalisatie van de gewichten van de WOWA operator zijn de parameters gefixeerd bij de integratie van de methode in het detectie-algoritme. De keuze van de parameters berust op aanbevelingen van de onderzoeker die de optimalisatiemethode geïmplementeerd heeft. De parameters voor de volledige optimalisatie zijn vrij te bepalen via het configuratiebestand. Voor de resultaten bekomen in paragraaf 5.3.2 zijn de parameters gebaseerd op de bevindingen die volgen uit de eerste fase van de implementatie van het optimalisatie-algoritme (zie paragraaf 4.4.1). De parameters van deze optimalisatiemethoden hebben zeker een invloed op de kwaliteit van het resultaat. Deze invloed kan onderzocht worden in een opvolgend werk.

Door gebrek aan tijd en rekenkracht waren de keuzes van de omgevingen beperkt voor de optimalisatie van het volledige detectie-algoritme. De optimalisatie werd gestart voor een omgeving die 66 gebruikers telt, maar deze werd stopgezet door een gebrek aan tijd. De schatting van de rekentijd bedroeg 27 dagen. Indien meer tijd en rekenkracht beschikbaar zouden zijn, zou het nuttig zijn om in een opvolgend werk een grotere groep gebruikers te analyseren. Eveneens zou het interessant zijn om de keuze van de gebruikers anders aan te pakken dan op basis van het subnet. Het gevaar bestaat namelijk dat de optimale parameters te veel zijn afgestemd op de gebruikers van het subnet, wat het geval is bij bespreking van de resultaten in paragraaf

5.4. Door een grotere groep gebruikers uit verschillende subnets te analyseren kan dit probleem omzeild worden.

6.4 Algemeen besluit

Deze masterpoef is uitgevoerd in het kader van onderzoek naar een systeem voor de detectie van APTs. De Research Unit Cyber Defense (RUCD) van de Koninlijke Militaire School is bezig met de ontwikkeling van MASFAD (Military multi-Agent System For APT Dectection). Dit systeem bestaat uit verschillende onderdelen die elk op een unieke wijze APTs detecteren. Het systeem bundelt alle resultaten van de onderdelen om zo een betere detectie te garanderen.

Één onderdeel van MASFAD is ontworpen door Thomas Gilon. Als eindwerk heeft hij een systeem ontwikkelt dat APTs detecteert aan de hand van grafen gebaseerd op de logs van een proxy. Dit systeem is voorzien van vele parameters waarop de analist kan inspelen om zo de detectie van APTs te verbeteren. Een eerste beperkte studie naar optimale parameters was reeds uitgevoerd. Deze combinatie van parameters dienden als startpunt voor de analist. Het opzet van deze masterproef was om de optimale parameters van het detectie-algoritme te bepalen. Met de optimale parameters worden de parameters bedoeld die leiden tot een maximale AUC dus de beste detectie.

Het detectie-algoritme is voorzien van 7 nieuwe parameters en een nieuwe package. De nieuwe parameters hebben betrekking tot de integratie van de WOWA operator in het detectie-algoritme. De WOWA operator is een andere aggregatietechniek dan het gewogen gemiddelde die toegevoegd is aan het detectie-algoritme. De aggregatietechniek wordt gebruikt om het finale klassement op te stellen dat bepaalt welke domeinen het meest verdacht zijn. Een optimalisatiemethode van de gewichten van de WOWA operator is eveneens toegevoegd aan het algoritme. De gebruikte methode is een Genetisch Algoritme. Deze optimalisatie leidt tot een betere AUC ten opzichte van de parameters van Gilon.

De nieuwe package die aan het detectie-algoritme is toegevoegd voert de optimalisatie van het volledige detectie-algoritme uit. Deze optimalisatie berust op de Particle Swarm Optimization methode. De resultaten voor de optimalisatie van het volledige detectie-algoritme zijn beter dan deze van de optimalisatie van de gewichten van de WOWA operator. Deze betere resultaten gaan wel gepaard met een toegenomen rekentijd. De rekentijd is beduidend hoger voor de volledige optimalisatie aangezien het detectie-algoritme meerdere malen doorlopen wordt.

Het werk dat is uitgevoerd voldoet aan de doelen die zijn opgesteld. De optimale parameters zijn te bepalen. Uit de resultaten blijkt dat de bekomen optimale parameters te veel zijn afgestemd op de omgeving waarin ze bepaald zijn. Desalniettemin zijn de nodige instrumenten aanwezig om in opvolgend werk de optimale parameters te bepalen als de nodige tijd en rekenkracht aanwezig zijn.

Bibliografie

[1] T. Gilon, "Détection d'APT basée sur le traitement de graphes : analyse paramétrisable et interactive de logs de proxy," Master's thesis, École Royale Militaire, Bruxelles, 2017.

[2] C. Vasudev, *Graph theory with applications*. New Age International, 2006.

[3] S. Vandeput, "De strategische visie voor Defensie," 2016.

[4] Kaspersky Lab, "A slice of 2017 sofacy activity." [Online]. Beschikbaar: `https://securelist.com/a-slice-of-2017-sofacy-activity/83930/`. [Geraadpleegd op 24-05-2019].

[5] W. Mees and T. Debatty, "Multi-agent System for APT Detection," in *2014 IEEE International Symposium on Software Reliability Engineering Workshops*, pp. 401–406, Nov 2014.

[6] NIST, "NIST Mission, Vision, Core Competencies, and Core Values." [Online]. Beschikbaar: `https://www.nist.gov/about-nist/our-organization/mission-vision-values`. [Geraadpleegd op 06-05-2019].

[7] R. S. Ross, *Managing Information Security Risk: Organization, Mission, and Information System View (NIST SP 800-39)*. National Institute of Standards and Technology, 2011.

[8] P. Chen, L. Desmet, and C. Huygens, "A Study on Advanced Persistent Threats," in *15th IFIP International Conference on Communications and Multimedia Security (CMS)* (B. Decker and A. Zúquete, eds.), vol. LNCS-8735 of *Communications and Multimedia Security*, (Aveiro, Portugal), pp. 63–72, Springer, Sept. 2014.

[9] R. Bace and P. Mell, *NIST Special Publication on Intrusion Detection Systems*. National Institute of Standards and Technology, 2001.

[10] L. Herløw and S. J. Hansen, "Detection and Prevention of Advanced Persistent Threats: Evaluating and Testing APT Lifecycle Models Using Real World Examples and Preventing Attacks through the Use of Mitigation Strategies and Current Best Practices," Master's thesis, Denmark: DTU Compute: Department of Applied Mathematics and Computer, 2015.

[11] Tor Project, "History." [Online]. Beschikbaar: `https://www.torproject.org/about/history/`. [Geraadpleegd op 06-05-2019].

[12] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic syntax (RFC 3986)," *Network Working Group*, 2005.

[13] V. Torra, "The WOWA operator: a review," in *Recent developments in the ordered weighted averaging operators: theory and practice*, pp. 17–28, Springer, 2011.

[14] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.

[15] E. K. Burke and G. Kendall, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. New York: Springer Science & Business Media, 2 ed., 2014.

[16] M. Pant, R. Thangaraj, and A. Abraham, "Particle swarm optimization: performance tuning and empirical analysis," in *Foundations of Computational Intelligence Volume 3*, pp. 101–128, Springer, 2009.

[17] F. Van Den Bergh, *An analysis of Particle Swarm Optimizers*. PhD thesis, University of Pretoria, 2001.

[18] R. L. Haupt and S. E. Haupt, *Practical genetic algorithms*. John Wiley & Sons, Inc, 2004.

[19] T. Debatty, "java-aggregation." [Online]. Beschikbaar: `https://github.com/tdebatty/java-aggregation`. [Geraadpleegd op 27-2-2019].

[20] PHP, "PHP Manual - Preface." [Online]. Beschikbaar: `https://www.php.net/manual/en/preface.php`. [Geraadpleegd op 24-5-2019].

[21] A. Croix, "Training a multi-criteria decision system and application to the detection of PHP webshells." [Online]. Beschikbaar: `https://cylab.be/storage/blog/24/files/WOG6vgMhnQGqZLA6o7Uc4n764xYlPtCfHXqMAVQv.pdf`. [Geraadpleegd op 20-5-2019].

[22] A. Croix, "java-wowa-training." [Online]. Beschikbaar: `https://gitlab.cylab.be/cylab/java-wowa-training`. [Geraadpleegd op 21-2-2019].

[23] S. Surjanovic and D. Bingham, "Virtual Library of Simulation Experiments: Test Functions and Datasets - Optimization Test Problems." [Online]. Beschikbaar: `https://www.sfu.ca/~ssurjano/optimization.html?fbclid=IwAR11GYY82e2ZcApl-yaRj_Rs5Tu0he73PjSTxbHw9r0pm-FeE19OTeRU2w4`. [Geraadpleegd op 12-3-2019].

Bijlagen

A. PSO package

A.1 Main.java

```java
package aptgraph.pso;

import java.io.File;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.CommandLineParser;
import org.apache.commons.cli.DefaultParser;
import org.apache.commons.cli.HelpFormatter;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * Main class.
 *
 * @author Jordy Cansse
 */
public final class Main {

    private Main() {

    }

    private static final Logger LOGGER = Logger.getLogger(
            Main.class.getName());

    /**
     * Main method of PSO.
     *
     * @param args Arguments from the command line
     * @throws org.apache.commons.cli.ParseException If text can't be parsed
     */
    public static void main(final String[] args) throws ParseException {
        // Parse command line arguments
        Options options = new Options();
        options.addOption("i", true, "Input configuration file (required)");
        options.addOption("h", false, "Show this help");
        CommandLineParser parser = new DefaultParser();
        CommandLine cmd = parser.parse(options, args);

        if (cmd.hasOption("h") || !cmd.hasOption("i")) {
            HelpFormatter formatter = new HelpFormatter();
```

```
50              formatter.printHelp("java -jar pso-<version>.jar", options);
51          }
52
53          JSONObject obj;
54          List<String> config = null;
55          try {
56              config = Files.readAllLines(
57                      Paths.get(cmd.getOptionValue("i")),
58                      StandardCharsets.UTF_8);
59          } catch (IOException ex) {
60              Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
61          }
62
63          if (config.size() != 1) {
64              throw new ParseException("Corrupt input configuration file");
65          }
66
67          try {
68              obj = new JSONObject(config.get(0));
69          } catch (JSONException ex) {
70              throw new JSONException(ex + "\nJSON did not match ");
71          }
72
73          Problem problem = new Problem(new File(obj.getString("input_file")),
74                  obj.getString("output_dir"), obj.getString("format"),
75                  obj.getInt("n_apt_tot"));
76
77          Parameters parameters = new Parameters(
78                  LOGGER,
79                  obj.getInt("swarm_size"),
80                  new double[]{obj.getDouble("inertia_weight_1"),
81                      obj.getDouble("inertia_weight_2")},
82                  obj.getDouble("inertia_weight_ratio"),
83                  obj.getDouble("cognitive_weight"),
84                  obj.getDouble("social_weight"),
85                  obj.getInt("max_iteration"),
86                  problem);
87          PSO pso = new PSO(parameters);
88          Position best_position = pso.run();
89      }
90  }
```

A.2 Parameters.java

```
1  package aptgraph.pso;
2
3  import java.util.logging.Logger;
4
5  /**
6   * Parameters class. Represents the parameters of the PSO algorithm.
7   *
8   * @author Jordy Cansse
9   */
10 public class Parameters {
11     private Logger logger;
12     private int swarm_size;
13     private double inertia_weight;
14     private double[] inertia_weight_vector;
15     private double inertia_weight_ratio;
16     private double cognitive_weight;
17     private double social_weight;
18     private int max_iteration;
```

```
19      private Problem problem;
20
21      /**
22       * Contains the parameters for the PSO algorithm.
23       *
24       * @param logger
25       * @param swarm_size size of the swarm
26       * @param inertia_weight_vector inertia weight vector
27       * @param inertia_weight_ratio inertia weight ratio
28       * @param cognitive_weight cognitive weight
29       * @param social_weight social weight
30       * @param max_iteration maximum number of iterations (stopping criteria)
31       * @param problem Problem to optimize
32       */
33      public Parameters(
34              final Logger logger,
35              final int swarm_size,
36              final double[] inertia_weight_vector,
37              final double inertia_weight_ratio,
38              final double cognitive_weight,
39              final double social_weight,
40              final int max_iteration,
41              final Problem problem) {
42          this.logger = logger;
43          this.swarm_size = swarm_size;
44          this.cognitive_weight = cognitive_weight;
45          this.social_weight = social_weight;
46          this.max_iteration = max_iteration;
47          this.problem = problem;
48          // constant inertia weight
49          if (inertia_weight_ratio == 1
50                  && inertia_weight_vector[0] == inertia_weight_vector[1]) {
51              this.inertia_weight_vector = new double[this.getMaxIteration()];
52              for (int i = 0; i < this.getMaxIteration(); i++) {
53                  this.inertia_weight_vector[i] = inertia_weight_vector[0];
54              }
55          // linearly decreasing inertia weight (arithmethic progression)
56          } else if (inertia_weight_ratio == -1
57                  && inertia_weight_vector[0] != inertia_weight_vector[1]) {
58              double step = (inertia_weight_vector[0] - inertia_weight_vector[1])
59                      / this.getMaxIteration();
60              this.inertia_weight_vector = new double[this.getMaxIteration()];
61              this.inertia_weight_vector[0] = inertia_weight_vector[0];
62              for (int i = 1; i < this.max_iteration; i++) {
63                  this.inertia_weight_vector[i] =
64                          this.inertia_weight_vector[i - 1] - step;
65              }
66          // inertia weight decreases w.r.t. a ratio (geometric progression)
67          } else if (inertia_weight_ratio != 1
68                  && inertia_weight_vector[0] == inertia_weight_vector[1]) {
69              this.inertia_weight_vector = new double[this.getMaxIteration()];
70              this.inertia_weight_vector[0] = inertia_weight_vector[0];
71              for (int i = 1; i < this.max_iteration; i++) {
72                  this.inertia_weight_vector[i] =
73                      this.inertia_weight_vector[i - 1] * inertia_weight_ratio;
74              }
75          }
76      }
77
78      /**
79       * Setter for the logger.
80       *
81       * @param logger logger
82       */
```

```
83      public final void setLogger(final Logger logger) {
84          this.logger = logger;
85      }
86
87      /**
88       * Getter for the logger.
89       *
90       * @return Logger : logger
91       */
92      public final Logger getLogger() {
93          return this.logger;
94      }
95
96      /**
97       * Setter for the swarm size.
98       *
99       * @param swarm_size size of the swarm
100      */
101     public final void setSwarmSize(final int swarm_size) {
102         this.swarm_size = swarm_size;
103     }
104
105     /**
106      * Getter for the swarm size.
107      *
108      * @return int : size of the swarm
109      */
110     public final int getSwarmSize() {
111         return this.swarm_size;
112     }
113
114     /**
115      * Setter for the inertia weight.
116      *
117      * @param inertia_weight inertia weight
118      */
119     public final void setInertiaWeight(final double inertia_weight) {
120         this.inertia_weight = inertia_weight;
121     }
122
123     /**
124      * Getter for the inertia weight.
125      *
126      * @return double : inertia weight
127      */
128     public final double getInertiaWeight() {
129         return this.inertia_weight;
130     }
131
132     /**
133      * Setter for the inertia weight vector.
134      *
135      * @param inertia_weight_vector inertia weight vector
136      */
137     public final void setInertiaWeightVector(
138             final double[] inertia_weight_vector) {
139         this.inertia_weight_vector = inertia_weight_vector.clone();
140     }
141
142     /**
143      * Getter for the inertia weight vector.
144      *
145      * @return double[] : inertia weight vector
146      */
```

```
147     public final double[] getInertiaWeightVector() {
148         return this.inertia_weight_vector.clone();
149     }
150 
151     /**
152      * Setter for the inertia weight ratio.
153      *
154      * @param inertia_weight_ratio inertia weight ratio
155      */
156     public final void setInertiaWeightRatio(final double inertia_weight_ratio) {
157         this.inertia_weight_ratio = inertia_weight_ratio;
158     }
159 
160     /**
161      * Getter for the inertia weight ratio.
162      *
163      * @return double : inertia weight ratio
164      */
165     public final double getInertiaWeightRatio() {
166         return this.inertia_weight_ratio;
167     }
168 
169     /**
170      * Setter for the cognitive weight.
171      *
172      * @param cognitive_weight cognitive weight
173      */
174     public final void setCognitiveWeight(final double cognitive_weight) {
175         this.cognitive_weight = cognitive_weight;
176     }
177 
178     /**
179      * Getter for the cognitive weight.
180      *
181      * @return double : cognitive weight
182      */
183     public final double getCognitiveWeight() {
184         return this.cognitive_weight;
185     }
186 
187     /**
188      * Setter for the social weight.
189      *
190      * @param social_weight social weight
191      */
192     public final void setSocialWeight(final double social_weight) {
193         this.social_weight = social_weight;
194     }
195 
196     /**
197      * Getter for the social weight.
198      *
199      * @return double : social weight
200      */
201     public final double getSocialWeight() {
202         return this.social_weight;
203     }
204 
205     /**
206      * Setter for the maximum number of iterations.
207      *
208      * @param max_iteration maximum number of iterations
209      */
210     public final void setMaxIteration(final int max_iteration) {
```

```
211         this.max_iteration = max_iteration;
212     }
213 
214     /**
215      * Getter for the maximum number of iterations.
216      *
217      * @return int : maximum number of iterations
218      */
219     public final int getMaxIteration() {
220         return this.max_iteration;
221     }
222 
223     /**
224      *
225      * @param problem problem
226      */
227     public final void setProblem(final Problem problem) {
228         this.problem = problem;
229     }
230 
231     /**
232      * Getter for the problem.
233      *
234      * @return Problem : problem
235      */
236     public final Problem getProblem() {
237         return this.problem;
238     }
239 }
```

A.3 Particle.java

```
1  package aptgraph.pso;
2 
3  /**
4   * Particle class. Represents the individuals in the swarm.
5   *
6   * @author Jordy Cansse
7   */
8  public class Particle {
9      private Position position;
10     private Velocity velocity;
11     private double fitness = Double.POSITIVE_INFINITY;
12     private double pbest = Double.POSITIVE_INFINITY;
13     private Position pbestposition;
14 
15     /**
16      * Getter for fitness.
17      *
18      * @return double : current fitness
19      */
20     public final double getFitness() {
21         return this.fitness;
22     }
23 
24     /**
25      * Setter for fitness.
26      *
27      * @param fitness : fitness
28      */
29     public final void setFitness(final double fitness) {
30         this.fitness = fitness;
```

```
31      }
32
33      /**
34       * Updates the fitness of the particle and the personal best.
35       */
36      public final void updateFitness(final Problem problem) {
37          this.setFitness(problem.evaluate(this.getPosition()));
38          if (this.getFitness() < this.getpbest()) {
39              this.setpbest(this.getFitness());
40              this.setpbestposition(new Position(
41                      this.getPosition().getK(),
42                      this.getPosition().getFeatureWeights(),
43                      this.getPosition().getPruneThreshold(),
44                      this.getPosition().getNumberRequests(),
45                      this.getPosition().getRankingWeights(),
46                      this.getPosition().getRankingWeightsOrdered()));
47          }
48      }
49
50      /**
51       * Getter for personal best fitness.
52       *
53       * @return double : personal best fitness
54       */
55      public final double getpbest() {
56          return this.pbest;
57      }
58
59      /**
60       * Setter for personal best fitness.
61       *
62       * @param pbest personal best fitness
63       */
64      public final void setpbest(final double pbest) {
65          this.pbest = pbest;
66      }
67
68      /**
69       * Getter for the personal best position.
70       *
71       * @return Position : personal best position
72       */
73      public final Position getpbestposition() {
74          return this.pbestposition;
75      }
76
77      /**
78       * Setter for the personal best position.
79       *
80       * @param pbestposition personal best position
81       */
82      public final void setpbestposition(final Position pbestposition) {
83          this.pbestposition = pbestposition;
84      }
85
86      /**
87       * Getter for the current position.
88       *
89       * @return Position : current position
90       */
91      public final Position getPosition() {
92          return this.position;
93      }
94
```

```
95      /**
96       * Setter for the current position.
97       *
98       * @param position position
99       */
100     public final void setPosition(final Position position) {
101         this.position = position;
102     }
103
104     /**
105      * Getter for the current velocity.
106      *
107      * @return velocity : current velocity
108      */
109     public final Velocity getVelocity() {
110         return this.velocity;
111     }
112
113     /**
114      * Setter for the current velocity.
115      *
116      * @param velocity velocity
117      */
118     public final void setVelocity(final Velocity velocity) {
119         this.velocity = velocity;
120     }
121
122     /**
123      * Returns the characteristics of the particle.
124      *
125      * @return string : string of characteristics of the particle
126      */
127     @Override
128     public final String toString() {
129         String output = "Fitness: " + -this.getFitness() + "\n"
130                 + this.getPosition().toString()
131                 + this.getVelocity().toString();
132         return output;
133     }
134 }
```

A.4 Psotion.java

```
1  package aptgraph.pso;
2
3  import java.util.Arrays;
4
5  /**
6   * Position class. Represent the position of a particle.
7   *
8   * @author Jordy Cansse
9   */
10 public class Position {
11     private final int k;
12     private final double[] feature_weights;
13     private final double prune_threshold;
14     private final int number_requests;
15     private final double[] ranking_weights;
16     private final double[] ranking_weights_ordered;
17
18     /**
19      * Position constructor.
```

```
20      *
21      * @param k value of k
22      * @param feature_weights feature weights
23      * @param prune_threshold prune threshold
24      * @param number_requests number of requests
25      * @param ranking_weights ranking weights
26      * @param ranking_weights_ordered ordered ranking weights
27      */
28     public Position(final int k, final double[] feature_weights,
29             final double prune_threshold, final int number_requests,
30             final double[] ranking_weights,
31             final double[] ranking_weights_ordered) {
32         this.k = k;
33         this.feature_weights = feature_weights.clone();
34         this.prune_threshold = prune_threshold;
35         this.number_requests = number_requests;
36         this.ranking_weights = ranking_weights.clone();
37         this.ranking_weights_ordered = ranking_weights_ordered.clone();
38     }
39
40     /**
41      * Returns the value of k.
42      * @return int : value of k
43      */
44     public final int getK() {
45         return this.k;
46     }
47
48     /**
49      * Returns the feature weights.
50      * @return double[] : feature weights
51      */
52     public final double[] getFeatureWeights() {
53         return this.feature_weights.clone();
54     }
55
56     /**
57      * Returns the prune threshold.
58      * @return double : prune threshold
59      */
60     public final double getPruneThreshold() {
61         return this.prune_threshold;
62     }
63
64     /**
65      * Returns the number of requests.
66      * @return int : number of requests
67      */
68     public final int getNumberRequests() {
69         return this.number_requests;
70     }
71
72     /**
73      * Returns the ranking weights.
74      * @return double : ranking weights
75      */
76     public final double[] getRankingWeights() {
77         return this.ranking_weights.clone();
78     }
79
80     /**
81      * Returns the ranking ordered weights.
82      * @return double : ranking weights ordered
83      */
```

```
84      public final double[] getRankingWeightsOrdered() {
85          return this.ranking_weights_ordered.clone();
86      }
87
88      /**
89       * Returns the position as a string.
90       * @return string : string of the position
91       */
92      @Override
93      public final String toString() {
94          String output = "Position: " + String.valueOf(this.getK())
95              + ", " + Arrays.toString(this.getFeatureWeights())
96              + ", " + String.valueOf(this.getPruneThreshold())
97              + ", " + String.valueOf(this.getNumberRequests())
98              + ", " + Arrays.toString(this.getRankingWeights())
99              + ", " + Arrays.toString(this.getRankingWeightsOrdered())
100             + "\n";
101         return output;
102     }
103 }
```

A.5 Problem.java

```
1  package aptgraph.pso;
2
3  import aptgraph.batch.BatchProcessor;
4  import aptgraph.study.ROC;
5  import aptgraph.server.RequestHandler;
6  import aptgraph.server.Output;
7  import java.io.File;
8  import java.io.FileInputStream;
9  import java.io.IOException;
10 import java.nio.file.Path;
11 import java.nio.file.Paths;
12 import java.util.LinkedList;
13 import java.util.TreeMap;
14 import java.util.List;
15
16 /**
17  *
18  * @author Jordy Cansse
19  */
20 public final class Problem {
21
22     private final File file;
23     private String output_dir_string;
24     private String format;
25     private boolean children_bool = true;
26     private boolean overwrite_bool = false;
27     private boolean study_out = true;
28     private String user = "0.0.0.0";
29     private double[] feature_ordered_weights = new double[]{0.8, 0.2};
30     private double max_cluster_size = 100000.0;
31     private boolean prune_z_bool = true;
32     private boolean cluster_bool = false;
33     private boolean cluster_z_bool = false;
34     private boolean whitelist_bool = true;
35     private String white_ongo = "";
36     private boolean wowa_bool = true;
37     private boolean wowa_norm = false;
38     private boolean apt_search = true;
39     private int n_apt_tot;
```

```
40      private boolean opti_ranking = false;
41
42      /**
43       * Problem constructor.
44       *
45       * @param file file
46       * @param output_dir_string outdir dir string
47       * @param format format
48       * @param n_apt_tot n_apt_tot
49       */
50      public Problem(final File file, final String output_dir_string,
51              final String format, final int n_apt_tot) {
52          this.file = file;
53          this.output_dir_string = output_dir_string;
54          this.format = format;
55          this.n_apt_tot = n_apt_tot;
56      }
57
58      /**
59       * Returns the string of the output directory.
60       * @return String : output_dir_string
61       */
62      public String getOutputDirString() {
63          return output_dir_string;
64      }
65
66      /**
67       * Evalution function of the problem.
68       *
69       * @param position position
70       * @return double : auc
71       */
72      public double evaluate(final Position position) {
73          // Acquisition of the parameters for the detectionalgorithm
74          int k = position.getK();
75          double[] feature_weights = position.getFeatureWeights();
76          double prune_threshold = position.getPruneThreshold();
77          double number_requests = position.getNumberRequests();
78          double[] ranking_weights = position.getRankingWeights();
79          double[] ranking_weights_ordered = position.getRankingWeightsOrdered();
80
81          // Execution of the Batch Processor to create the graphs
82          String output_dir_string_temp = output_dir_string;
83          output_dir_string_temp =
84                  output_dir_string_temp.concat(String.valueOf(k) + "_infected/");
85          Path output_dir = Paths.get(output_dir_string_temp);
86          try {
87              BatchProcessor processor = new BatchProcessor();
88              processor.analyze(k,
89                      new FileInputStream(file),
90                      output_dir,
91                      format, children_bool, overwrite_bool);
92          } catch (IOException ex) {
93              System.err.println(ex);
94          }
95
96          // Execution of the Server to produce the ranking
97          RequestHandler handler = new RequestHandler(output_dir, study_out);
98              //Paths.get("src/test/resources/dummyDir"), false);
99          Output output = handler.analyze(user, feature_weights,
100                 feature_ordered_weights, prune_threshold,
101                 max_cluster_size, prune_z_bool, cluster_bool,
102                 cluster_z_bool, whitelist_bool, white_ongo, number_requests,
103                 ranking_weights, wowa_bool, wowa_norm, ranking_weights_ordered,
```

```
104                 apt_search, opti_ranking);
105
106         // Loading the ranking
107         TreeMap<Double, LinkedList<String>> ranking
108                         = output.getRanking();
109
110         // Producing the ROC
111         int n_dom_tot = handler.getMemory().getAllDomains()
112                 .get("all").values().size() - n_apt_tot;
113         List<double[]> roc_curve =
114                 ROC.computeROC(ranking, n_dom_tot, n_apt_tot);
115         // roc_curve doesn't always contains values up to abscissa 1
116         if (roc_curve.get(roc_curve.size() - 1)[0] != 1) {
117             roc_curve.add(new double[]{1,
118                 roc_curve.get(roc_curve.size() - 1)[1]});
119         }
120         // insert values to calculate AUC
121         int counter = 0;
122         int init_size = roc_curve.size();
123         for (int i = 0; i < (init_size - 1); i++) {
124             if (roc_curve.get(i + counter)[1]
125                     == roc_curve.get(i + 1 + counter)[1]) {
126                 continue;
127             } else {
128                 roc_curve.add(i + 1 + counter, new double[]{
129                     roc_curve.get(i + 1 + counter)[0],
130                     roc_curve.get(i + counter)[1]});
131                 counter += 1;
132             }
133         }
134         // calculate AUC
135         double auc = 0;
136         for (int i = 0; i < (roc_curve.size() - 1); i++) {
137             auc += (roc_curve.get(i + 1)[0] - roc_curve.get(i)[0])
138                     * roc_curve.get(i)[1];
139         }
140         return (-auc);
141     }
142 }
```

A.6 PSO.java

```
1  package aptgraph.pso;
2
3  import java.util.logging.Level;
4
5  /**
6   * PSO class. Optimizes the detection algorithm.
7   *
8   * @author Jordy Cansse
9   */
10 public class PSO {
11         private final Parameters parameters;
12         private String msg;
13
14     /**
15      * PSO constructor.
16      *
17      * @param parameters parameters of the PSO algorithm.
18      */
19     public PSO(final Parameters parameters) {
20         this.parameters = parameters;
```

```
21      }
22
23      /**
24       * Run method.
25       *
26       * @return Position : optimal position
27       */
28      public final Position run() {
29          Swarm swarm = new Swarm(this.parameters);
30          swarm.initializeSwarm();
31          swarm.updategbest();
32          swarm.exportData(this.parameters.getProblem().getOutputDirString()
33                  + "iteration" + 0 + ".txt");
34          msg = "Iteration 0" + "\n"
35              + "Best Fitness: " + (-swarm.getgbest()) + "\n"
36              + "Position: " + swarm.getgbestposition().toString();
37          this.parameters.getLogger().log(Level.INFO, msg);
38          for (int i = 0; i < parameters.getMaxIteration(); i++) {
39              swarm.getParameters().setInertiaWeight(
40              swarm.getParameters().getInertiaWeightVector()[i]);
41              swarm.updateVelocity();
42              swarm.updatePosition();
43              swarm.updateFitnessEvals();
44              swarm.updategbest();
45              swarm.exportData(
46                  this.parameters.getProblem().getOutputDirString()
47                          + "iteration" + (i + 1) + ".txt");
48              msg = "Iteration " + (i + 1) + "\n"
49                  + "Best Fitness: " + (-swarm.getgbest()) + "\n"
50                  + "Position: " + swarm.getgbestposition().toString();
51              this.parameters.getLogger().log(Level.INFO, msg);
52          }
53          return swarm.getgbestposition();
54      }
55  }
```

A.7 Swarm.java

```
1   package aptgraph.pso;
2
3   import java.io.FileOutputStream;
4   import java.io.OutputStream;
5   import java.io.IOException;
6   import java.util.ArrayList;
7   import java.util.Collections;
8   import java.util.Random;
9
10  /**
11   * Swarm class. Represents the swarm.
12   *
13   * @author Jordy Cansse
14   */
15  public class Swarm {
16      private Parameters parameters;
17      private ArrayList<Particle> swarm = new ArrayList<Particle>();
18      private double gbest = Double.POSITIVE_INFINITY;
19      private Position gbestposition;
20      private ArrayList<Double> fitnessevals = new ArrayList<Double>();
21      private Random generator = new Random();
22
23      /**
24       * Swarm constructor.
```

```
25      *
26      * @param parameters parameters of the PSO algorithm
27      */
28     public Swarm(final Parameters parameters) {
29         this.parameters = parameters;
30     }
31
32     /**
33      * Setter for the parameters.
34      *
35      * @param parameters parameters
36      */
37     public final void setParameters(final Parameters parameters) {
38         this.parameters = parameters;
39     }
40
41     /**
42      * Getter for the parameters.
43      *
44      * @return Parameters : parameters
45      */
46     public final Parameters getParameters() {
47         return this.parameters;
48     }
49
50     /**
51      * Returns the global best fitness.
52      *
53      * @return double : global best fitness
54      */
55     public final double getgbest() {
56         return this.gbest;
57     }
58
59     /**
60      * Returns the global best position.
61      *
62      * @return Position : global best position
63      */
64     public final Position getgbestposition() {
65         return this.gbestposition;
66     }
67
68     /**
69      * Updates the global best.
70      */
71     public final void updategbest() {
72         if (Collections.min(this.getFitnessEvals()) < this.getgbest()) {
73             Particle p = this.swarm.get(this.getFitnessEvals().indexOf(
74                     Collections.min(this.getFitnessEvals())));
75             this.gbest = p.getpbest();
76             this.gbestposition = new Position(
77                     p.getpbestposition().getK(),
78                     p.getpbestposition().getFeatureWeights(),
79                     p.getpbestposition().getPruneThreshold(),
80                     p.getpbestposition().getNumberRequests(),
81                     p.getpbestposition().getRankingWeights(),
82                     p.getpbestposition().getRankingWeightsOrdered());
83         }
84     }
85
86     /**
87      * Returns the fitness of all the particles.
88      *
```

```
89          * @return ArrayList<Double> : list of all the fitnesses
90          */
91         public final ArrayList<Double> getFitnessEvals() {
92             return this.fitnessevals;
93         }
94 
95         /**
96          * Updates the fitness of all the particles.
97          */
98         public final void updateFitnessEvals() {
99             for (int i = 0; i < swarm.size(); i++) {
100                this.fitnessevals.set(i, swarm.get(i).getFitness());
101            }
102        }
103 
104        /**
105         * Initializes the swarm.
106         */
107        public final void initializeSwarm() {
108            for (int i = 0; i < parameters.getSwarmSize(); i++) {
109                Particle p = new Particle();
110                p.setPosition(new Position(Utils.initiatePosK(),
111                Utils.initiatePosFeatureWeights(),
112                Utils.initiatePosPruneThreshold(),
113                Utils.initiatePosNumberRequests(),
114                Utils.initiatePosRankingWeights(),
115                Utils.initiatePosRankingWeights()));
116                p.setVelocity(new Velocity(Utils.initiateVelK(),
117                Utils.initiateVelFeatureWeights(),
118                Utils.initiateVelPruneThreshold(),
119                Utils.initiateVelNumberRequests(),
120                Utils.initiateVelRankingWeights(),
121                Utils.initiateVelRankingWeights()));
122                p.updateFitness(this.parameters.getProblem());
123                this.swarm.add(p);
124                this.fitnessevals.add(p.getFitness());
125            }
126        }
127 
128        /**
129         * Updates the velocity of all the particles.
130         */
131        public final void updateVelocity() {
132            for (Particle particle : this.swarm) {
133                Velocity vel = particle.getVelocity();
134                particle.setVelocity(new Velocity(
135                    Utils.updateVelK(
136                        particle.getPosition().getK(),
137                        vel.getK(),
138                        particle.getpbestposition().getK(),
139                        this.gbestposition.getK(),
140                        this.parameters),
141                    Utils.updateVelFeatureWeights(
142                        particle.getPosition().getFeatureWeights(),
143                        vel.getFeatureWeights(),
144                        particle.getpbestposition().getFeatureWeights(),
145                        this.gbestposition.getFeatureWeights(),
146                        this.parameters),
147                    Utils.updateVelPruneThreshold(
148                        particle.getPosition().getPruneThreshold(),
149                        vel.getPruneThreshold(),
150                        particle.getpbestposition().getPruneThreshold(),
151                        this.gbestposition.getPruneThreshold(),
152                        this.parameters),
```

```
153                 Utils.updateVelNumberRequests(
154                     particle.getPosition().getNumberRequests(),
155                     vel.getNumberRequests(),
156                     particle.getpbestposition().getNumberRequests(),
157                     this.gbestposition.getNumberRequests(),
158                     this.parameters),
159                 Utils.updateVelRankingWeights(
160                     particle.getPosition().getRankingWeights(),
161                     vel.getRankingWeights(),
162                     particle.getpbestposition().getRankingWeights(),
163                     this.gbestposition.getRankingWeights(),
164                     this.parameters),
165                 Utils.updateVelRankingWeights(
166                     particle.getPosition().getRankingWeightsOrdered(),
167                     vel.getRankingWeightsOrdered(),
168                     particle.getpbestposition().getRankingWeightsOrdered(),
169                     this.gbestposition.getRankingWeightsOrdered(),
170                     this.parameters)));
171         }
172     }
173 
174     /**
175      * Updates the position of all the particles.
176      */
177     public final void updatePosition() {
178         for (Particle particle : this.swarm) {
179             Position pos = particle.getPosition();
180             Velocity vel = particle.getVelocity();
181             particle.setPosition(new Position(
182                 Utils.updatePosK(pos.getK(), vel.getK()),
183                 Utils.updatePosFeatureWeights(
184                 pos.getFeatureWeights(), vel.getFeatureWeights()),
185                 Utils.updatePosPruneThreshold(
186                 pos.getPruneThreshold(), vel.getPruneThreshold()),
187                 Utils.updatePosNumberRequests(
188                 pos.getNumberRequests(), vel.getNumberRequests()),
189                 Utils.updatePosRankingWeights(
190                 pos.getRankingWeights(), vel.getRankingWeights()),
191                 Utils.updatePosRankingWeights(
192                     pos.getRankingWeightsOrdered(),
193                     vel.getRankingWeightsOrdered())));
194             particle.updateFitness(this.parameters.getProblem());
195         }
196     }
197 
198     /**
199      * Exports the characteristics of all the particle to a file.
200      *
201      * @param output string of the output file
202      */
203     public final void exportData(final String output) {
204         try {
205             OutputStream output_file = new FileOutputStream(output);
206             for (Particle particle : this.swarm) {
207                 output_file.write(particle.toString()
208                         .getBytes("UTF-8"));
209             }
210             output_file.close();
211         } catch (IOException ex) {
212             System.out.println(ex.getMessage());
213         }
214     }
215 }
```

A.8 Utils.java

```
package aptgraph.pso;

import java.util.Random;

/**
 * Utils class. Utilities to initiate and update positions and velocities.
 *
 * @author Jordy Cansse
 */
public final class Utils {

    private Utils() {

    }

    private static double vel_max = 0.5;

    private static int k_min = 10;
    private static int k_max = 100;
    private static int k_delta = 10;

    private static double feature_weights_min = 0;
    private static double feature_weights_max = 1;

    private static double prune_threshold_min = -1;
    private static double prune_threshold_max = 1;

    private static int number_requests_min = 1;
    private static int number_requests_max = 20;

    private static double ranking_weights_min = 0;
    private static double ranking_weights_max = 1;

/**
 * Initiates the position of k.
 *
 * @return int : initial position of k
 */
    public static int initiatePosK() {
        Random generator = new Random();
        double pos = k_delta * Math.round(((generator.nextDouble()
                * (k_max - k_min)) + k_min)
                / k_delta);
        return (int) pos;
    }

/**
 * Initiates the velocity of k.
 *
 * @return int : initial velocity of k
 */
    public static int initiateVelK() {
        Random generator = new Random();
        double vel = k_delta * Math.round(vel_max
                * ((2 * generator.nextDouble() * (k_max
                        - k_min))
                        - (k_max - k_min)) / k_delta);
        return (int) vel;
    }

/**
 * Updates the position of k.
```

```
63  *
64  * @param pos position
65  * @param vel velocity
66  * @return int : updated position of k
67  */
68     public static int updatePosK(final int pos, final int vel) {
69         int pos_temp = pos + vel;
70         if (pos_temp > k_max) {
71             pos_temp = k_max - (pos_temp - k_max);
72         } else if (pos_temp < k_min) {
73             pos_temp = k_min + (k_min - pos_temp);
74         }
75         return pos_temp;
76     }
77 
78 /**
79  * Updates the velocity of k.
80  *
81  * @param pos position
82  * @param vel velocity
83  * @param pbest personal best position
84  * @param gbest global best position
85  * @param parameters parameters of the PSO algorithm
86  * @return int : updated velocity of k
87  */
88     public static int updateVelK(final int pos, final int vel, final int pbest,
89             final int gbest, final Parameters parameters) {
90         Random generator = new Random();
91         double vel_temp = k_delta * Math.round((
92             parameters.getInertiaWeight() * vel
93             + parameters.getCognitiveWeight()
94                 * generator.nextDouble() * (pbest - pos)
95             + parameters.getSocialWeight() * generator.nextDouble()
96                 * (gbest - pos)) / k_delta);
97         if (Math.abs(vel_temp)
98             > vel_max * (k_max - k_min)) {
99             vel_temp = k_delta * Math.round((Math.signum(vel_temp)
100                    * vel_max * (k_max - k_min))
101                    / k_delta);
102        }
103    return (int) vel_temp;
104    }
105 
106 /**
107  * Initiates the position of prune threshold.
108  *
109  * @return double : initial position of prune threshold
110  */
111    public static double initiatePosPruneThreshold() {
112        Random generator = new Random();
113        double pos = (generator.nextDouble()
114                * (prune_threshold_max - prune_threshold_min)
115                + prune_threshold_min);
116        return pos;
117    }
118 
119 /**
120  * Initiates the velocity of prune threshold.
121  *
122  * @return double : initial velocity of prune threshold
123  */
124    public static double initiateVelPruneThreshold() {
125        Random generator = new Random();
126        double vel = vel_max * (2 * generator.nextDouble()
```

```
127                 * (prune_threshold_max - prune_threshold_min)
128                 - (prune_threshold_max - prune_threshold_min));
129         return vel;
130     }
131
132 /**
133  * Updates the position of prune threshold.
134  *
135  * @param pos position
136  * @param vel velocity
137  * @return double : updated position of prune threshold
138  */
139     public static double updatePosPruneThreshold(final double pos,
140             final double vel) {
141         double pos_temp = pos + vel;
142         if (pos_temp > prune_threshold_max) {
143             pos_temp = prune_threshold_max
144                     - (pos_temp - prune_threshold_max);
145         } else if (pos_temp < prune_threshold_min) {
146             pos_temp = prune_threshold_min
147                     + (prune_threshold_min - pos_temp);
148         }
149         return pos_temp;
150     }
151
152 /**
153  * Updates the velocity of prune threshold.
154  *
155  * @param pos position
156  * @param vel velocity
157  * @param pbest personal best position
158  * @param gbest global best position
159  * @param parameters parameters of the PSO algorithm
160  * @return double : updated velocity of prune threshold
161  */
162     public static double updateVelPruneThreshold(final double pos,
163             final double vel, final double pbest, final double gbest,
164             final Parameters parameters) {
165         Random generator = new Random();
166         double vel_temp = (parameters.getInertiaWeight() * vel
167             + parameters.getCognitiveWeight() * generator.nextDouble()
168                 * (pbest - pos)
169             + parameters.getSocialWeight() * generator.nextDouble()
170                 * (gbest - pos));
171         if (Math.abs(vel_temp)
172             > vel_max * (prune_threshold_max
173                     - prune_threshold_min)) {
174             vel_temp = Math.signum(vel_temp) * vel_max
175                 * (prune_threshold_max - prune_threshold_min);
176         }
177     return vel_temp;
178     }
179
180 /**
181  * Initiates the position of number of requests.
182  *
183  * @return int : initial position of number of requests
184  */
185     public static int initiatePosNumberRequests() {
186         Random generator = new Random();
187         double pos = Math.round((generator.nextDouble()
188                 * (number_requests_max - number_requests_min))
189                 + number_requests_min);
190         return (int) pos;
```

```
191     }
192
193 /**
194  * Initiates the velocity of number of requests.
195  *
196  * @return int : initial velocity of number of requests
197  */
198     public static int initiateVelNumberRequests() {
199         Random generator = new Random();
200         double vel = Math.round(vel_max
201                 * ((2 * generator.nextDouble() * (number_requests_max
202                     - number_requests_min))
203                     - (number_requests_max
204                             - number_requests_min)));
205         return (int) vel;
206     }
207
208 /**
209  * Updates the position of number of requests.
210  *
211  * @param pos position
212  * @param vel velocity
213  * @return int : updated position of number of requests
214  */
215     public static int updatePosNumberRequests(final int pos, final int vel) {
216         int pos_temp = pos + vel;
217         if (pos_temp > number_requests_max) {
218             pos_temp = number_requests_max - (pos_temp
219                     - number_requests_max);
220         } else if (pos_temp < number_requests_min) {
221             pos_temp = number_requests_min
222                     + (number_requests_min - pos_temp);
223         }
224         return pos_temp;
225     }
226
227 /**
228  * Updates the velocity of number of requests.
229  *
230  * @param pos position
231  * @param vel velocity
232  * @param pbest personal best position
233  * @param gbest global best position
234  * @param parameters parameters of the PSO algorithm
235  * @return int : updated velocity of number of requests
236  */
237     public static int updateVelNumberRequests(final int pos, final int vel,
238             final int pbest, final int gbest, final Parameters parameters) {
239         Random generator = new Random();
240         double vel_temp = Math.round(
241             parameters.getInertiaWeight() * vel
242             + parameters.getCognitiveWeight() * generator.nextDouble()
243                 * (pbest - pos)
244             + parameters.getSocialWeight() * generator.nextDouble()
245             * (gbest - pos));
246         if (Math.abs(vel_temp)
247             > vel_max * (number_requests_max
248                 - number_requests_min)) {
249             vel_temp = Math.signum(vel_temp) * vel_max
250                 * (number_requests_max - number_requests_min);
251         }
252     return (int) vel_temp;
253     }
254
```

```
255 /**
256  * Initiates the position of feature weights.
257  *
258  * @return double[] : initial position of feature weights
259  */
260     public static double[] initiatePosFeatureWeights() {
261         Random generator = new Random();
262         double[] pos = new double[2];
263         for (int i = 0; i < pos.length; i++) {
264             pos[i] = (generator.nextDouble() * (feature_weights_max
265                 - feature_weights_min)) + feature_weights_min;
266         }
267         double sum = 0;
268         for (double weight : pos) {
269             sum += weight;
270         }
271         double[] pos_norm = new double[2];
272         for (int i = 0; i < pos_norm.length; i++) {
273             pos_norm[i] = pos[i] / sum;
274         }
275         return pos_norm;
276     }
277
278 /**
279  * Initiates the velocity of feature weights.
280  *
281  * @return double[] : initial velocity of feature weights
282  */
283     public static double[] initiateVelFeatureWeights() {
284         Random generator = new Random();
285         double[] vel = new double[2];
286         for (int i = 0; i < vel.length; i++) {
287             vel[i] = vel_max * ((2 * generator.nextDouble()
288                 * (feature_weights_max - feature_weights_min))
289                 - (feature_weights_max - feature_weights_min));
290         }
291         return vel;
292     }
293
294 /**
295  * Updates the position of feature weights.
296  *
297  * @param pos position
298  * @param vel velocity
299  * @return double[] : updated position of feature weights
300  */
301     public static double[] updatePosFeatureWeights(final double[] pos,
302             final double[] vel) {
303         double[] pos_temp = new double[2];
304         for (int i = 0; i < pos_temp.length; i++) {
305             pos_temp[i] = pos[i] + vel[i];
306             if (pos_temp[i] > feature_weights_max) {
307                 pos_temp[i] = feature_weights_max
308                         - (pos_temp[i] - feature_weights_max);
309             } else if (pos_temp[i] < feature_weights_min) {
310                 pos_temp[i] = feature_weights_min
311                         + (feature_weights_min - pos_temp[i]);
312             }
313         }
314         double sum = 0;
315         for (double weight : pos_temp) {
316             sum += weight;
317         }
318         double[] pos_norm = new double[2];
```

```
319         for (int i = 0; i < pos_norm.length; i++) {
320             pos_norm[i] = pos_temp[i] / sum;
321         }
322         return pos_norm;
323     }
324 
325 /**
326  * Updates the velocity of feature weights.
327  *
328  * @param pos position
329  * @param vel velocity
330  * @param pbest personal best position
331  * @param gbest global best position
332  * @param parameters parameters of the PSO algorithm
333  * @return double[] : updated velocity of feature weights
334  */
335     public static double[] updateVelFeatureWeights(final double[] pos,
336             final double[] vel, final double[] pbest, final double[] gbest,
337             final Parameters parameters) {
338         Random generator = new Random();
339         double[] vel_temp = new double[2];
340         for (int i = 0; i < vel_temp.length; i++) {
341             vel_temp[i] = (parameters.getInertiaWeight() * vel[i]
342                 + parameters.getCognitiveWeight() * generator.nextDouble()
343                     * (pbest[i] - pos[i])
344                 + parameters.getSocialWeight() * generator.nextDouble()
345                     * (gbest[i] - pos[i]));
346             if (Math.abs(vel_temp[i])
347                 > vel_max * (feature_weights_max
348                     - feature_weights_min)) {
349             vel_temp[i] = Math.signum(vel_temp[i]) * vel_max
350                 * (feature_weights_max - feature_weights_min);
351         }
352         }
353     return vel_temp;
354     }
355 
356 /**
357  * Initiates the position of ranking weights.
358  *
359  * @return double[] : initial position of ranking weights
360  */
361     public static double[] initiatePosRankingWeights() {
362         Random generator = new Random();
363         double[] pos = new double[3];
364         for (int i = 0; i < pos.length; i++) {
365             pos[i] = (generator.nextDouble() * (feature_weights_max
366                     - feature_weights_min))
367                     + feature_weights_min;
368         }
369         double sum = 0;
370         for (double weight : pos) {
371             sum += weight;
372         }
373         double[] pos_norm = new double[3];
374         for (int i = 0; i < pos_norm.length; i++) {
375             pos_norm[i] = pos[i] / sum;
376         }
377         return pos_norm;
378     }
379 
380 /**
381  * Initiates the velocity of ranking weights.
382  *
```

```
383  * @return double[] : initial velocity of ranking weights
384  */
385     public static double[] initiateVelRankingWeights() {
386         Random generator = new Random();
387         double[] vel = new double[3];
388         for (int i = 0; i < vel.length; i++) {
389             vel[i] = vel_max * ((2 * generator.nextDouble()
390                 * (feature_weights_max - feature_weights_min))
391                 - (feature_weights_max - feature_weights_min));
392         }
393         return vel;
394     }
395 
396 /**
397  * Updates the position of ranking weights.
398  *
399  * @param pos position
400  * @param vel velocity
401  * @return double[] : updated position of ranking weights
402  */
403     public static double[] updatePosRankingWeights(final double[] pos,
404             final double[] vel) {
405         double[] pos_temp = new double[3];
406         for (int i = 0; i < pos_temp.length; i++) {
407             pos_temp[i] = pos[i] + vel[i];
408             if (pos_temp[i] > feature_weights_max) {
409                 pos_temp[i] = feature_weights_max
410                         - (pos_temp[i] - feature_weights_max);
411             } else if (pos_temp[i] < feature_weights_min) {
412                 pos_temp[i] = feature_weights_min
413                         + (feature_weights_min - pos_temp[i]);
414             }
415         }
416         double sum = 0;
417         for (double weight : pos_temp) {
418             sum += weight;
419         }
420         double[] pos_norm = new double[3];
421         for (int i = 0; i < pos_norm.length; i++) {
422             pos_norm[i] = pos_temp[i] / sum;
423         }
424         return pos_norm;
425     }
426 
427 /**
428  * Updates the velocity of ranking weights.
429  *
430  * @param pos position
431  * @param vel velocity
432  * @param pbest personal best position
433  * @param gbest global best position
434  * @param parameters parameters of the PSO algorithm
435  * @return double[] : updated velocity of ranking weights
436  */
437     public static double[] updateVelRankingWeights(final double[] pos,
438             final double[] vel, final double[] pbest, final double[] gbest,
439             final Parameters parameters) {
440         Random generator = new Random();
441         double[] vel_temp = new double[3];
442         for (int i = 0; i < vel_temp.length; i++) {
443             vel_temp[i] = (parameters.getInertiaWeight() * vel[i]
444                 + parameters.getCognitiveWeight() * generator.nextDouble()
445                     * (pbest[i] - pos[i])
446                 + parameters.getSocialWeight() * generator.nextDouble()
```

```
447                     * (gbest[i] - pos[i]));
448             if (Math.abs(vel_temp[i])
449                 > vel_max * (feature_weights_max
450                     - feature_weights_min)) {
451             vel_temp[i] = Math.signum(vel_temp[i]) * vel_max
452                 * (feature_weights_max - feature_weights_min);
453         }
454         }
455     return vel_temp;
456     }
457 }
```

A.9 Velocity.java

```
1  package aptgraph.pso;
2
3  import java.util.Arrays;
4
5  /**
6   * Velocity class. Represent the velocity of a particle.
7   *
8   * @author Jordy Cansse
9   */
10 public class Velocity {
11     private final int k;
12     private final double[] feature_weights;
13     private final double prune_threshold;
14     private final int number_requests;
15     private final double[] ranking_weights;
16     private final double[] ranking_weights_ordered;
17
18     /**
19      * Velocity constructor.
20      *
21      * @param k value of k
22      * @param feature_weights feature weights
23      * @param prune_threshold prune threshold
24      * @param number_requests number of requests
25      * @param ranking_weights ranking weights
26      * @param ranking_weights_ordered ordered ranking weights
27      */
28     public Velocity(final int k, final double[] feature_weights,
29             final double prune_threshold, final int number_requests,
30             final double[] ranking_weights,
31             final double[] ranking_weights_ordered) {
32         this.k = k;
33         this.feature_weights = feature_weights.clone();
34         this.prune_threshold = prune_threshold;
35         this.number_requests = number_requests;
36         this.ranking_weights = ranking_weights.clone();
37         this.ranking_weights_ordered = ranking_weights_ordered.clone();
38     }
39
40     /**
41      * Returns the value of k.
42      * @return int : value of k
43      */
44     public final int getK() {
45         return this.k;
46     }
47
48     /**
```

```
49      * Returns the feature weights.
50      * @return double[] : feature weights
51      */
52     public final double[] getFeatureWeights() {
53         return this.feature_weights.clone();
54     }
55 
56     /**
57      * Returns the prunt threshold.
58      * @return double : prune threshold
59      */
60     public final double getPruneThreshold() {
61         return this.prune_threshold;
62     }
63 
64     /**
65      * Returns the number of requests.
66      * @return int : number of requests
67      */
68     public final int getNumberRequests() {
69         return this.number_requests;
70     }
71 
72     /**
73      * Returns the ranking weights.
74      * @return double : ranking weights
75      */
76     public final double[] getRankingWeights() {
77         return this.ranking_weights.clone();
78     }
79 
80     /**
81      * Returns the ranking ordered weights.
82      * @return double : ranking weights ordered
83      */
84     public final double[] getRankingWeightsOrdered() {
85         return this.ranking_weights_ordered.clone();
86     }
87 
88     /**
89      * Returns the velocity as a string.
90      * @return string : string of the velocity
91      */
92     @Override
93     public final String toString() {
94         String output = "Velocity: " + String.valueOf(this.getK())
95             + ", " + Arrays.toString(this.getFeatureWeights())
96             + ", " + String.valueOf(this.getPruneThreshold())
97             + ", " + String.valueOf(this.getNumberRequests())
98             + ", " + Arrays.toString(this.getRankingWeights())
99             + ", " + Arrays.toString(this.getRankingWeightsOrdered())
100            + "\n";
101        return output;
102    }
103 }
```