Learning-Based Network Intrusion Detection : an Imbalanced, Constantly Evolving and Timely Problem

Nicolas Sourbier

DOCTORAT BRETAGNE LOIRE / MATHSTIC

INSA RENNES

THÈSE DE DOCTORAT DE

L'INSTITUT NATIONAL DES SCIENCES APPLIQUEES DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Signal, Image, Vision

Par

Nicolas SOURBIER

Learning-Based Network Intrusion Detection : an Imbalanced, Constantly Evolving and Timely Problem

Thèse présentée et soutenue à Rennes, le 29 Septembre 2022
Unité de recherche : IETR

Thèse N° :D22 - 22 / 22ISAR 22

Rapporteurs avant soutenance :

Isabelle Chrisment	Professeur des universités, Université de Lorraine, LORIA
Malcolm Heywood	Professeur, Dalhousie University

Composition du Jury :

Président :	Gilles Grimaud	Professeur des universités, Université de Lille, CRIStAL
Examinateurs :	Peggy Cellier	Maître de conférences HDR, INSA de Rennes, IRISA
	Grégory Blanc	Maître de conférences, SAMOVAR, IMT/Télécom Sud Paris, Institut Polytechnique de Pa
Dir. de thèse :	Maxime Pelcat	Maitre de Conférences, HDR, INSA Rennes

Invités :

Thomas Guyet	Maître de Conférences HDR, INRIA centre Lyon
Frédéric Majorczyk	Expert SSI, DGA MI, Rennes
Olivier Gesny	Directeur innovations, PROPH3CY Rennes

Table of Contents

Acknowledgements

Ce travail est le résultat de trois années. La plupart des personnes qui ont croisé mon chemin sur cette période méritent ici quelques mots.

Dans un premier temps, les membres de mon encadrement méritent une mention particulière. Maxime, pour la direction de thèse que tu as su mener, toi même sous la pression de nombreux projets, merci. Tu as su me transmettre beaucoup, scientifiquement mais aussi humainement. Thomas, Frédéric et Olivier, merci à vous pour l'encadrement, les conseils, les relectures et votre implication en général sur ce projet. Merci à Karol Desnos, qui bien qu'en dehors de l'encadrement de cette thèse à été une aide importante pour le développement des outils, les conseils, les discussions parfois trop longues -mais toujours très riches- lors des pauses café.

Merci aux rapporteurs ainsi qu'aux examinateurs de ma thèse, d'avoir consacré du temps à évaluer mon travail. Vos retours ont été constructifs à tout niveaux et leur bienveillance m'a aidé à en terminer joyeusement avec ma vie de doctorant.

Merci à l'équipe VAADER, pour la convivialité et les échanges, pour les moments de détentes et les pauses café, pour les petits à côtés et les attentions diverses. En particulier, merci aux personnes qui se seront personnellement impliquées dans mon projet, le temps d'un échange, d'une relecture. Merci à ceux qui ont été là pour changer d'air, sortir, boire une bière en terrasse, vous avez assurément apporté un peu de légèreté dans ce quotidien. Une mention particulière au bureau 214 qui en plus du reste, à su me divertir quotidiennement à coup de musique et de blagues douteuses.

Merci à ma famille qui à été d'un soutien inestimable sur ces trois ans. Sans votre appui constant, ce document ne serait peut-être pas ce qu'il est aujourd'hui. Merci pour le bonheur que vous me procurez.

Merci à mes amis qui ont été là pour moi, qui ont pris soin de moi et qui m'ont soutenu. Sans avoir besoin de vous nommer, je vous porte haut dans mon cœur, mais ça, vous le savez déjà.

Même si certaines personnes se retrouveront dans le paragraphe au dessus, merci à mes partenaires sportifs, qu'ils soient du groupe de capoeira ou de l'escalade pour avoir été un refuge et un havre de tranquillité pour moi et mon esprit.

CHAPTER 1

French Summary

1.1 Introduction

Les systèmes de détection d'intrusion réseau (NIDS) sont des systèmes observant l'environnement réseau IP, essayant d'en isoler les comportements malveillants compromettant l'intégrité, la disponibilité ou la confidentialité des services et des données fournies par ce réseau [Van20 ; HH05 ; Zam01]. Il existe deux types de NIDS. Les uns identifient les comportements malveillants en se référant à une base de connaissance pré-existante (SIDS). [HS14 ; KT03]. Les systèmes de détection d'intrusion par anomalie (AIDS), eux, essayent de qualifier les comportements malveillants en se basant sur des modèles de comportements sains et malveillants, généralement issus d'un apprentissage machine (ML) [Gar+09 ; JPP11].

La détection d'intrusion dans des environnements dynamiques, tels que les réseaux IP, sont liés à d'importants problèmes [KA17 ; KV02]. La collecte de données réseau représentatives et étiquetées est complexe et coûteuse et pose un problème pour la conception et l'évaluation des NIDS. Le problème de détection d'intrusion dans un réseau se formalise également comme un problème de classification déséquilibré. Enfin, il n'y a aucune garantie qu'un modèle appris sur un jeu de données d'intrusions réseau puisse être utile pour la détection d'intrusion sur un système opérationnel.

Ce manuscrit explore les capacités des TPGs [KH17b], des graphes de programmes, pour la création d'un AIDS. Les TPGs sont une forme d'apprentissage machine légers et polyvalents basés sur des concepts de programmation génétique (GP).

Ce manuscrit étudie les biais de représentation et d'étiquetage des données réseau, ainsi que leur effet sur l'apprentissage des TPGs. L'impact des données déséquilibrées sur l'apprentissage des TPGs est également exploré. Enfin, nous montrons que le caractère évolutif des TPGs permet l'adaptabilité du modèle à un environnement dynamique, rendant possible la détection de nouvelles attaques par un apprentissage continu. Il est montré que la légèreté de la méthode peut être utilisée pour une utilisation rapide et basse consommation, faisant des TPGs de très bons candidats pour le futur des NIDS.

1.2 Contexte : apprentissage de comportements dans un réseau IP

Ce manuscrit se concentre sur la sécurité des réseaux et la détection d'intrusion ainsi que sur l'utilisation de techniques ML pour la modélisation du trafic IP. Cette section introduit des notions de sécurité réseau essentielles à la compréhension du domaine et discute de leur intersection avec le domaine du ML.

1.2.1 La sécurité du réseau

Un réseau est un ensemble d'éléments communicants appelés **terminaux**. Les données sont échangées entre deux terminaux par l'intermédiaire de **paquets** réseaux. La compromission de la sécurité des services est considérée comme une **intrusion**. Les intrusions peuvent avoir de fortes conséquences économiques, juridiques ou même sur la réputation des victimes. La détection d'intrusion, qui constitue la troisième étape de la procédure NIST [Bar+18], est une des 5 facettes de la sécurité des réseaux. Un NIDS analyse les informations qui circulent sur le réseau et essaye d'isoler le trafic spécifique menant à une intrusion [Den87]. Les NIDS sont utilisés pour signaler les intrusions à un analyste (humain) afin qu'il puisse agir et intervenir, en prenant des contres mesures pour réduire l'impact de l'intrusion. Il est primordial que les NIDS ne génèrent que très peu de fausses alertes positives. Pour un analyste seul, pas plus de 20 fausses alertes par jour doivent être générées. Par exemple, la précision minimale d'un AIDS pour une journée de trafic

de CICIDS 2017 [SLG18] est de 99.67%. Ce manuscrit explicite les enjeux et objectifs des AIDS de demain.

1.2.2 Apprentissage machine et AIDS

Dans le but de construire des modèles précis pour la détection d'intrusions réseau, une quantité massive de données réseau doit être recueillie. Les AIDS basés sur du ML prennent en entrée des données réseau ou des séquences particulières de données réseau appelées **flux** et lèvent des alertes lorsque le trafic observé correspond à une intrusion [Lev16]. Les données d'entraînement sont nécessaire à la modélisation du problème. Cela implique que des données biaisées peuvent causer la création d'un modèle également biaisé. Par exemple, les biais d'étiquetage causent dans cette étude (voir Chapitre 5) la perte de près de 14% de performance. De plus, il y a naturellement plus de comportements sains que d'intrusions sur un réseau. Cela veut dire que le ML se doit d'être résilient à la présence de données déséquilibrées [HG09]. Les jeux de données CICIDS 2017 [SLG18] et PUF [SSG18] comportent respectivement 17% et 12% de trafic d'intrusions. En résumé, la méthode ML idéale serait résistante aux biais, erreurs d'étiquetage et au déséquilibre de données. Cette méthode serait facilement mise à jour pour s'adapter à de nouveaux comportements émergents et permettrait un traitement de flux efficace afin de lever les alertes au plus tôt.

Ce manuscrit détaille les capacités des TPGs pour la détection d'intrusions et approfondit les axes précédents, à la croisée des domaines du ML et de la détection d'intrusion.

1.3 Challenges et objectifs

Dans le Chapitre 5, les avantages et inconvénients des données réseau disponibles sont évalués. Le défi est ici de trouver des moyens corrects d'exploiter les données du réseau et de trouver une méthode résistante à des erreurs d'étiquetage. De manière similaire à la résilience au biais, le Chapitre 6 vise à améliorer la robustesse de la méthode ML face à des données déséquilibrées. Tous les réseaux sont différents et par conséquent, un modèle entraîné sur un jeu de données n'est intuitivement pas adapté pour être utilisé dans un contexte opérationnel, différent de son contexte d'entraînement. Cette question est détaillée dans le Chapitre 7. Dans ce chapitre, nous présentons également une étude sur l'efficacité du TPG pour le traitement des données réseau. Enfin, afin de réduire le

temps d'analyse et de pouvoir réduire le temps entre l'intrusion et sa détection, un effort doit être fait pour être capable de traiter efficacement les données du réseau. Pour cela, nous avons étudié les capacités d'adaptation et la rapidité des TPGs.

1.4 Contributions : vers la création d'un AIDS basé sur de l'apprentissage machine.

Il est nécessaire de mieux comprendre les contraintes qui ont un impact sur la conception des AIDS. Cette section explore les inconvénients et les limites des AIDS actuels afin d'introduire nos contributions.

1.4.1 Impact des biais d'apprentissage sur les TPGs

Les données réseau peu représentatives du problème de la détection d'intrusion mènent à la création de modèles inexacts [GD95 ; PKP21]. Lorsque l'on extrait des données réelles du réseau, on se heurte à la RGPD et à un problème d'étiquetage très coûteux. Lorsque le trafic réseau est simulé, le comportement humain réaliste est absent de la jeu de données [Rin+19]. De plus, la manipulation des données est source d'une nouvelle introduction de biais. Il semble impératif de disposer de méthodes plus résistantes aux biais de données, pour réduire l'impact des biais dans la conception d'un AIDS. Le Chapitre 5 s'intéresse à l'impact du biais de représentation qui se produit lorsque les caractéristiques disponibles introduisent des corrélations non pertinentes avec l'étiquette associée. Le biais d'étiquetage de CICIDS 2017 est également étudié, car plus de 20% de l'ensemble du jeu de données est sujet à une erreur d'étiquetage [ERJ21]. Ce biais d'étiquette impacte la construction du modèle et conduit à plus de 14% d'erreurs dans la détection d'intrusion. Les TPGs sont utilisés comme des classificateurs et l'impact du biais d'étiquette sur les TPGs est étudié. Les résultats expérimentaux montrent que le TPGs peut apprendre lorsque jusque 20% des étiquettes sont incorrectes, ce qui en fait un bon candidat pour la détection d'intrusion où les données d'apprentissage sont souvent biaisées.

1.4.2 Apprentissage déséquilibré : une approche par la programmation génétique

Lorsque l'on utilise des techniques ML pour classer les entrées d'un jeu de données, ces entrées sont la plupart du temps équilibrées car il est plus facile de modéliser une solution à un problème équilibré où l'on peut construire des modèles indépendants de chacune des classes. Lorsqu'il existe un déséquilibre de données, des méthodes d'échantillonnage sont souvent utilisées pour rééquilibrer artificiellement les données [YL06 ; ZL14]. un jeu de données NIDS possède un trafic d'intrusions représentant en moyenne 10% du trafic. Des méthodes d'échantillonnage peuvent être appliquées mais cela éloigne les NIDS de leur utilisation opérationnelle. Pour garantir les performances de l'inférence, un entraînement en ligne sur des données réalistes est recommandé. L'entraînement y est soumis au déséquilibre des données. Le Chapitre 6 étudie l'impact des données déséquilibrées sur l'apprentissage et propose des modifications du TPGs en conséquence [JCD13]. Les résultats expérimentaux montrent que la robustesse des TPGs aux données déséquilibrées peut être améliorée et que les TPGs peuvent continuent d'apprendre dans des conditions où jusqu'à un échantillon positif sur 10000 échantillons est présent.

1.4.3 Adaptabilité des AIDS haute performance sur des réseaux opérationnels

Il semble contre-intuitif de se fier à un mécanisme de défense statique qui devrait être extrêmement précis dans des conditions d'inférence en constante évolution [Mor11 ; Rai12 ; Sym17]. Le biais de déploiement se produit lorsque l'entraînement ne représente pas les mêmes conditions que l'utilisation opérationnelle. Ce problème motive l'utilisation de méthodes d'apprentissage incrémental, présentes de par la programmation génétique dans les TPGs. Le Chapitre 7 explore les capacités des TPGs en terme de détection et d'adaptabilité [KH17b]. En tant que preuve de concept, le TPG est utilisé comme une sonde à apprentissage incrémental, intégrée à un système sur puce Exynos 5422, capable d'effectuer un apprentissage continu sur le flux de réseau entrant. En tant que dispositif toujours actif, un effort est fait pour le rendre fonctionnel à faible énergie. Les résultats expérimentaux montrent que la sonde est capable de s'adapter aux changements de l'environnement réseau, tout en étant capable d'analyser le flux de connexion au fur et à mesure, le tout pour moins de 3.5W. Les expériences sous conditions opérationnelles montre une précision

de 99.96%, supérieure aux 99.67% théoriques requis pour le jeu de données CICIDS2017. Cette précision implique des taux de détection d'attaque moins élevés.

CHAPTER 2

Introduction

Network Intrusion Detection Systems (NIDS) observe a network environment and try to identify malicious behaviors that compromise integrity, confidentiality or availability of either the network data or the systems [Van20; HH05; Zam01]. NIDS can be classified as Signature-based (SIDS) or Anomaly-based (AIDS). SIDS identify known malicious behaviors by comparing the traffic with a knowledge base [HS14; KT03]. Conversely, AIDS try to qualify the unknown, supposedly malicious behaviors [Gar+09; JPP11]. AIDS are mostly based on Machine Learning techniques.

Performing detection of rare events such as intrusions in an ever-changing environment using AIDS based on Machine Learning (ML) is a challenge bound to several big issues [KA17; KV02]. Network data is used to both setup and evaluate NIDS. Gathering representative network data with accurate label information is costly. Once qualitative network data is available, the detection problem can be addressed as a highly imbalanced classification problem. Finally, there is no guarantee that a learned AIDS on a network intrusion detection dataset is useful for real NIDS inference and is able to keep pace with the input data flow.

This thesis explores the capabilities of the Tangled Program Graphs (TPGs) framework [KH17b] to act as an AIDS probe. TPG is a form of machine learning based on genetic programming that offers lightweight and versatile learning capabilities. NIDS training data tend to be biased while the problem is imbalanced by nature. We study

the consequences of training data biases on the TPG framework and explore the impact of mislabeling and representation biases in the NIDS domain. We then specialize the TPG framework to fit imbalanced classification problems. We show that the genetic programming selection phase associated with the use of an adequate fitness function offer mitigation of data imbalance. We also show that the evolving topology of the TPG framework makes it capable of adapting to a dynamic network environment, taking into account zero-day intrusions. Finally, we demonstrate on experiments that the lightweight properties of the TPG framework can be exploited for a fast and low energy inference, making it a promising candidate for future NIDS technologies.

2.1 Context: learning behaviors in a dynamic IP network environment

This thesis is built on two technologies. It focuses on IP network security and intrusion detection, and uses ML techniques to contribute to this domain. This section introduces the context of IP network environment and security, and discusses the intersection of this domain with ML statistical learning methods.

2.1.1 Intrusion detection and security in a dynamic IP network

An IP network environment is a complex and ever changing framework. A network is a set of communicating devices called **endpoints**. These endpoints exchange data through the use of structured data containers called IP **Packets**. The security of the network is bound to the security of three of its individual components that are integrity, availability and confidentiality of both the data and systems. It means that, at any time, an authorized party should have access to a data (availability) that shall not be accessible by an unauthorized party (confidentiality) and that shall not have been modified by an unauthorized party (integrity) [Zam01; Van20]. Compromising any of these three characteristics is considered as an **intrusion**. Intrusions are harmful as they can cause economical, reputational and regulatory damages to the victims. Attackers that intrude a network use specific network packets that exploit one or several vulnerabilities of the network. The security of the network is partly bound to the detection of these intrusions that comes as the third component of the NIST framework [Bar+18]. This framework orders the five main high-level cyber-security actions as Identify, Protect, Detect, Respond

and Recover. An Intrusion Detection System (IDS) is a system that observes the networks packets and tries to recognize the packets that lead to an intrusion [Den87]. There exist IDS that examine the network traffic, comparing it to a knowledge base of malicious data packets and other IDS called AIDS that explore the unknown data packets and try to correlate them with the known sane and malicious behaviors to determine whether or not an IP packet is likely to lead to an intrusion. An AIDS raises alerts as a signal to help a human **analyst** to examine the data packets and then take counter-measures to reduce the impact of the intrusion. It is then primordial that alerts are early raised and that as few as possible false alerts are raised from the AIDS. For a single analyst, no more than 20 false positive alerts per day should be raised. For example, during the four days of traffic generation of the CICIDS 2017 dataset [SLG18], an IDS with a precision of 99.67% is acceptable.

This thesis explicates the goals of an AIDS and explores the research challenge of the conception of the next generation AIDS.

2.1.2 Learning AIDS using IP network data

Machine Learning (ML) is mostly used when it comes to the AIDS design. Other approaches using signal processing and decomposition are rarely used [ZL03; Ren+08]. In order to train AIDS, a considerable amount of network data containing realistic traffic and intrusion traffic is required. More than 85 % of recent ML-based AIDS are **classifiers** [Har07] that take as an input, a network packet or a sequence of network packets with a fixed source and destination called **network flow**, and raise an alert if the incoming data is classified as an intrusion [Lev16]. This training data is a key component for the design of an AIDS as the AIDS model takes decisions based on its acquired knowledge base. This implies that biased training data will result into a biased AIDS model, making the intrusion detection task less efficient (up to 14% in our case study on the bias impact). Intrusion detection data is also naturally imbalanced in a normal operating mode. There are more sane traffic exchanged between endpoints than intrusion traffic on a network. For example, the imbalance ratio on the CICIDS 2017 dataset [SLG18] is close to one intrusion for a hundred of connections (Two orders of magnitude or **Imbalance Order of Magnitude (IOM)**). In the PUF dataset [SSG18], extracted from real traffic, the IOM is one. This means that the ML method used for the design of an AIDS (i.e. a classifier) must be resilient to the presence of imbalanced data [HG09]. Even though the datasets do not necessarily reflect the ground truth, these values are lowest bound in which the

AIDS should be able to learn. There exist many different ML techniques, each of them having advantages and drawbacks when facing the intrusion detection problems [HTF09; CA16; KLM96]. In particular, the ideal ML method for the design of an AIDS is a ML method that:

- is resilient to imbalanced learning
- is resilient to label errors and biases, as analysts can not perfectly label datasets and tend to miss some attacks
- facilitate the update of the model, as the network environment and traffic is ever-changing
- provide a fast inference, as early intrusions detection opens for more countermeasures

In this thesis, the capabilities of the TPG framework for intrusion detection are explored. In particular, these four main axes are deepened as thesis contributions.

2.2 Challenges and thesis Objectives

This work aims to help in the comprehension of the requirements of ML-based AIDS. This work requires data that are fair and representative of the network intrusion detection problem. In Chapter 5, we evaluate the strengths and weaknesses of available network intrusion detection datasets. Data biases have to be studied and understood in order to have a correct and fair use of the network data and to be able to produce an unbiased operational intrusion detection system. The challenge is here to find correct methods for exploiting network data. Furthermore, as labeling network data is a complex task, it is important to find a method resilient to errors in the dataset labels. Chapter 6 focuses on the imbalance mitigation in AIDS. Intrusion data are naturally rare with respect to the normal traffic on a network. An IOM of two is present in the CICIDS 2017 dataset. This raises the question of imbalanced data comprehension and handling by the ML method. Furthermore, and similarly to the resilience to the biases, the resilience of the considered ML method to imbalance is an important challenge. Chapter 7 focuses on operational TPG-based AIDS performances. Operational NIDS require a training that conditions the model. This issue has to be studied as all networks are different and thus, a model trained offline is intuitively not suited to be used as a detector in a different context than

it has been trained for. Finally, in order to reduce the packet analysis time and to be able to reduce the time between the intrusion and its detection, an effort should be made to efficiently process the network data.

2.3 Thesis contributions: toward designing a ML-based AIDS

In order to fulfill AIDS needs using ML techniques, it is required to understand the ML constraints that have an impact on the AIDS performance. This section explores the drawbacks and limitations of present AIDS in order to introduce our contributions.

2.3.1 Contribution 1: assessing the biases of IP networks intrusion detection datasets and evaluating their effect on a TPG-based AIDS

To understand a problem, it is important to build an accurate representation of this problem. This is exactly the issue faced when training a ML method using low quality data. The representation of the problem is based on data that is poorly representative of the problem, leading to inaccurate models [GD95; PKP21]. Qualitative network data is hard to obtain. When extracting real traffic data, one faces the General Data Protection Regulation (GDRP) and must manually analyze the traffic in order to characterize which traffic is or is not an intrusion (unless unsupervised learning methods are used) [Bri+18]. When the network traffic is simulated (which is the case of most labeled network intrusion datasets), the realistic human behavior is shadowed in the generated dataset. *"[...]the perfect network-based data set is up-to-date. correctly labeled, publicly available, contains real network traffic with all kinds of attacks and normal user behavior as well as payload and spans a long time. Such a data set, however, does not exist and will (probably) never be created"* [Rin+19]. Furthermore, the manipulation of the data is subject to the introduction of **biases** that have an impact on the learned model. These data biases can be of several kinds such as non-representative features, label errors, mistakes in the acquisition of the data or the use of a non-representative population for the generation of the dataset [GHF22; Meh+21]. Two mitigation approaches can be thought. On the first hand, we can deal with the data itself, trying to extract representative features or to

generate qualitative normal and attack traffic. On the other hand, having methods that are more resilient to inaccurate labels helps in the reduction of the impact of the biases in the design of a learned AIDS. Chapter 5 focuses on the CICIDS 2017 datasets and explores the use of non-relevant features in the dataset. This issue is known as a **representation bias** and happens when available features introduce irrelevant correlations between the feature and the associated label. The label bias of the CICIDS 2017 is also explored, as more than 20% of the CICIDS dataset is subject to a label error [ERJ21]. This label bias has an impact on the built model and leads to errors in intrusion detection. In particular, the use of our TPG-based method results in a drop of 14% of precision when training using incorrect labels. Finally, the TPG framework is used as an ML-classifier and the impact of the label bias on the TPG is studied. Experimental results show that the TPG is resilient to up to 25 % of label bias making it a good candidate to face the intrusion detection problem.

2.3.2 Contribution 2: study of the impact of data imbalance on TPG performance

Learning on strongly imbalance data leads to a so-called "imbalanced data representation" [HG09]. When using ML techniques to classify entries of a dataset, these entries are most of the time forced to be balanced as it is easier to model a solution to a balanced problem than design a model of an imbalanced problem. When data imbalance exists, a pre-processing of the dataset is usually performed where sampling methods are used to artificially re-balance the dataset [YL06; ZL14]. As aforementioned, intrusion traffic represents a small proportion of the total amount of traffic on a network. Realistic data representations take into account the natural imbalance of the problem and thus, a fair NIDS dataset is imbalanced having an amount of intrusion traffic representing inferiorly to 10% of the overall data (These 10% are a mean of NIDS datasets). Sampling methods can be applied to facilitate the training of NIDS but unfortunately, doing so gets the NIDS farther from operational exploitation as the differences between the dataset and a live operational system can cause important drops of performances. To guarantee the performances of live inferring AIDSs, online training on realistic data is recommended and thus, the ML method use will be subject to the data imbalance. Chapter 6 explores algorithmic modifications of the ML, that are required to deal with the imbalanced nature of the problem. In order to propose algorithmic modifications to Genetic Programming

(GP) techniques for imbalanced classification, the classification problem and its impact on the ML method is studied [JCD13]. Then, a mitigation method based on the GP selection mechanism and the used fitness functions and evaluation metrics are exposed. Experimental results show that the TPG framework can be designed to be robust to data imbalance and keeps learning with high imbalance ratios in extreme conditions where less than one positive sample out of 10,000 samples is present.

2.3.3 Contribution 3: Evaluating TPG for stream processing, continual learning and high efficiency AIDS

Once trained and used as a detection probe, an AIDS is a static representation of the network environment. Conversely, the network environment is dynamic where new endpoints and services appear and where behaviors evolve. It seems counter intuitive to rely on a static defense mechanism that should be extremely precise in constantly changing inferring conditions [Mor11; Rai12; Sym17]. A first considerable change occurs after the offline training on a NIDS dataset to the operational use of the model on the network environment. For example, it is unlikely that a model trained on a dataset will infer under operational condition using the same IP network topology, same addresses and similar traffic. This issue is addressed as a deployment bias and happens when training and evaluation do not represent operational evaluation conditions. This first issue motivates for the use of incremental learning methods, allowed in the TPG framework by the GP method, in order to incrementally adapt to the live network inference after training on an offline dataset. In Chapter 7, the TPG capabilities in terms of detection capabilities and adaptability are experimentally explored [KH17b]. Furthermore, in order to perform live training on the network environment, the AIDS probe requires to keep pace with the incoming data flow. As an example, a saturated IP network with a 1GBps bandwidth generates a mean of 210 MBps network flow metadata to be analyzed. This figure comes under the pessimistic hypothesis of one flow per packet sent, with 80 floating point metadata extracted and an average of 1.5kB per packet. Once again, the TPG framework displays interesting properties of being lightweight and able to process rapidly large amount of data. As a Proof of Concept, the TPG is used in Chapter 7 as an incremental learning probe, embedded on a small Exynos 5422 System on Chip, able to perform live training on the incoming network flow up to 149 MB/s. As a comparison, the SIDS SNORT functions around an average of 3MB/s and state of the art methods

does not reach an analysis rate over 45MB/s. Finally, as an always-on device, an effort is made to make it able to keep pace with incoming network flow while having a low energy functioning point. Experimental results show that the probe is able to adapt to consequent changes on the network environment, using the incremental learning property of the TPG agent, while being able to process all the network flows information as they come and under 3.5W. Experimenting under operational condition leads to a precision of 99.96 which is above the 99.67% precision theoretically required for the CICIDS 2017 dataset. This high precision implies the detection of fewer attacks and, in particular, more subtle attacks such as infiltration or code injections tend not to be seen.

2.3.4 Appendix A: Prototyping and optimization of a Tangled Program Graph framework (GEGELATI)

As an enabler of this work, Chapter A details modifications to the legacy TPG framework that are used or explored in the different thesis contributions. The Generic Evolvable Graphs for Efficient Learning of Artificial Tangled Intelligence (GEGELATI) version of the TPG is deterministic for the reproducibility of the results and parallel to enhance the efficiency of the TPG framework. Its inferring performances are, thanks to a C code generation of the model, accelerated between 40 and 50 times. The graphs can be exported and imported and benefits from external parametering, which is convenient for the reproducibility of the results as well. Finally, minor modifications such as the use of constants or a semi-supervised version of the TPG have been studied. In particular, the semi-supervised version of the TPG is not yet functioning and thus, is not used in this manuscript.

Thesis outline

As a first part of this thesis, the context is presented. Chapter 3 gives details on the network environment in which the intrusion detection is performed. In particular, this chapter introduces notions of network security and presents the Intrusion Detection System (IDS) and its basic design. Chapter 4 presents the wide scope of the ML domain, discussing the existing methods and their potential advantages facing the intrusion detection problem. Particularly, the TPG framework is detailed as the main interest method of this thesis. The second part of this thesis focuses on research contributions to the domain

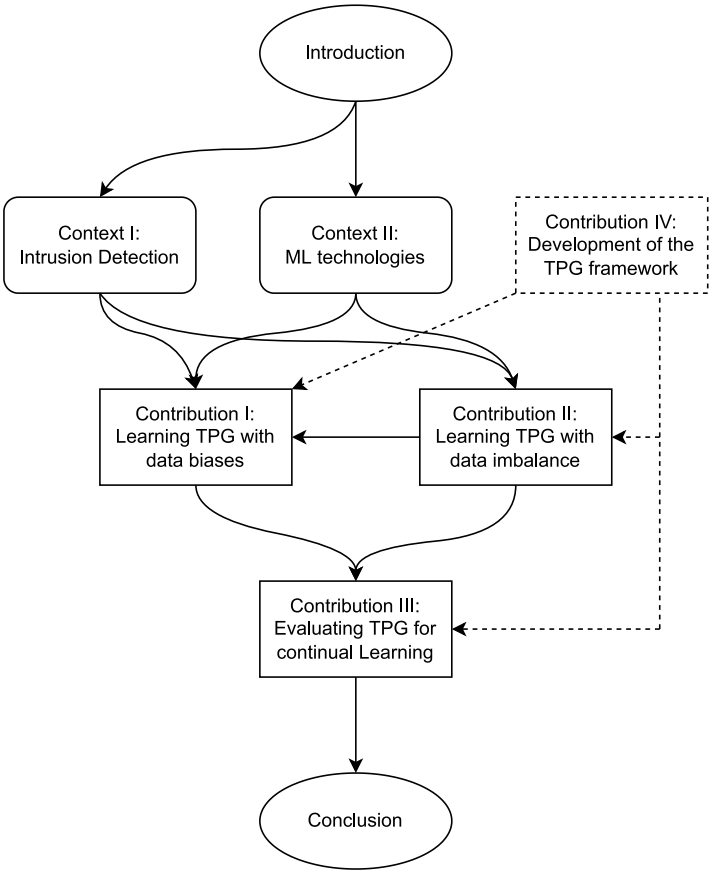


Figure 2.1: Links between this manuscript's chapters

of intrusion detection. Firstly, in Chapter 5 a particular interest is put on the available network data used in order to learn the semantics of the network intrusions. This chapter focuses on the representation bias of the network data and on the impact of errors in labels on the ML method. In Chapter 6, the data imbalance existing in network intrusion datasets is studied and algorithmic mitigation for GP methods are explored. Chapter 7 explores the detection capabilities of an incremental learning IDS, trained offline and used on a simulated live network that differs from the training conditions. Finally, Chapter A details implementations on the TPG framework that permits the different presented contributions. Figure 2.1 illustrates the links between the different chapters.

As a conclusion, Chapter 8 sums up the thesis contributions and details the future research work that constitutes the actual challenges faced by next generation intrusion detection systems.

Part I

Background

CHAPTER 3

Intrusion detection and security in a dynamic IP network

3.1 Introduction

Intrusion detection consists of spotting the actions of attackers attempting to compromise the integrity, confidentiality, or availability of a computer resource [Zam01; Van20]. The first Intrusion Detection System (IDS) was proposed by D. Denning in 1987 [Den87] as a way to early detect and prevent networking attacks and deviant behaviors. Intrusion detection is now used at a large scale and is necessary to ensure the security of a resource as attackers proved their ability to bypass the protections of the resources [Mor11; Sym17; Rai12]. The protection can be ensured, for example, by a firewall blocking incoming attacks and such a firewall can be bypassed by zero-days attacks.

To enhance the security of a network, the messages exchanged between two or more devices must be analyzed for early intrusion detection and for countermeasures to be deployed. In a realistic information system context, the connection rate is high, making it impossible for a human analyst to analyze the traffic in real-time. There is thus a need of precise systems that help the analysts to analyze the traffic and handle intrusions when detected.

This chapter presents generalities on IP computer networks, including its organization and its communication protocols, and then defines the basis of computer network security given the attack context and state-of-the-art network intrusion detection procedures.

3.2 The computer network: a dynamic and complex environment

This section aims to comprehend the dynamic network environment, its organization, its communication protocols and the main challenges in terms of network security.

Figure 3.1 presents the schematics used for the description of networks.

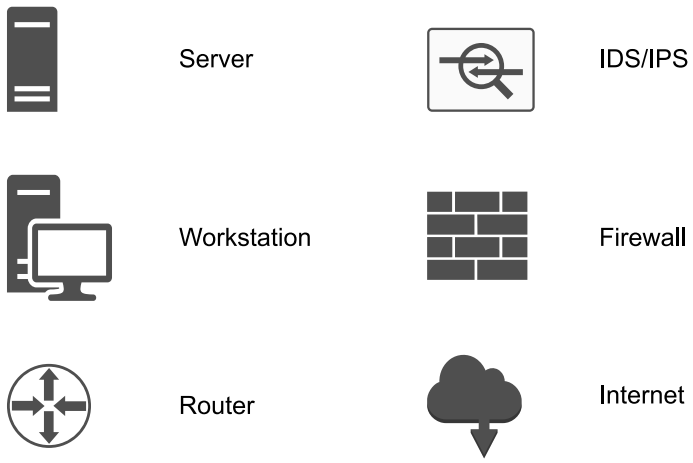


Figure 3.1: Schematics used for different endpoints, devices, software and services in a network. Conversely to IDSs, that raise alert when malicious packets are detected, an Intrusion Protection System (IPS) is more similar to a firewall that actively blocks the known malicious packets.

3.2.1 Computer networks: Endpoints and organization

Definition 3.2.1

*An **Information System (IS)** is a combination of people, hardware, software, communication networks, data resources and policies, as well as procedures that store, retrieve, transform, and disseminate information in an organization [OM05].*

Definition 3.2.2

*A **computer network** is a component of an IS that provides internal and external communication means based on message passing. It consists of a set of inter-connected computers that exchange information. The inter-connected hardware defines the network topology [GJM14] as a graph where nodes are the connection point of the transmission hardware and the edges are the physical links between the nodes including wired technolo-*

gies (Ethernet (RJ45), optical fibber, coaxial cables,...) and wireless technologies (WiFi, satellite communications) using mostly radio waves.

Definition 3.2.3

*An **endpoint** is a computer device that is connected to a network. Servers are endpoints and so are computers, laptops, smartphones and IoT devices.*

The inter-connection of the different endpoints in the network is called the network topology [SA13]. There exist a lot of different network topologies such as ring, star 3.2, bus, mesh 3.3, tree 3.4 and all the possible combination of those topologies. The most common combinations of topology is a tree network where some node drive a star network.

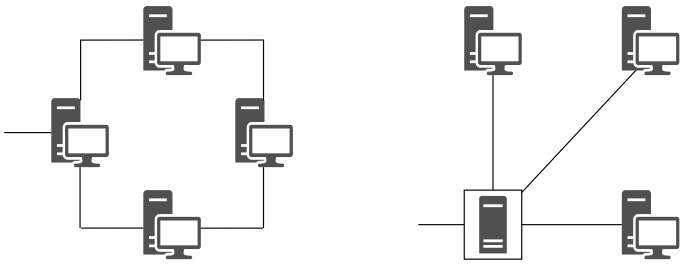


Figure 3.2: On the ring topology (left), each node is directly connected with exactly two nodes. Data travels from node to node, with each node along the way handling every packet. The star network topology (right) is a commonly used topology where each node is connected to a central node (represented by a server) that acts like a conduit to transmit messages.

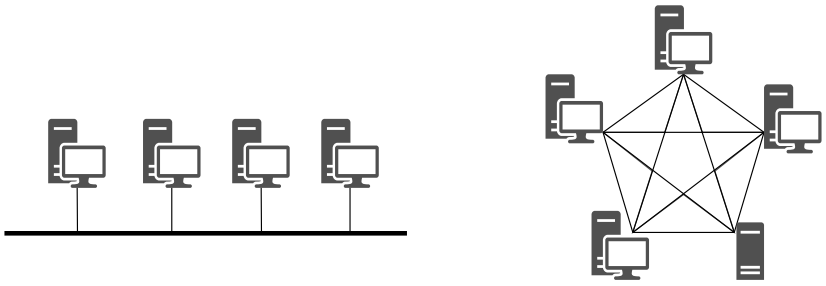


Figure 3.3: On the bus topology (left), each node is connected to a bi-directional (half-duplex) link called a bus. A mesh topology (right) is formed when each node is connected with all the other nodes in the network.

We can differentiate several categories of networks [PD07]. Among them, the Local Area Network (LAN) is a limited area network. Examples of LANs are domestic networks, small companies or school networks. Wide Area Network (WAN) is a network that extends

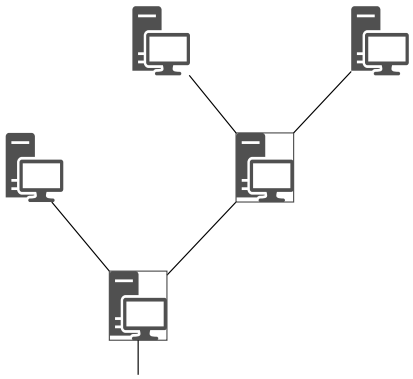


Figure 3.4: The tree network topology is a combination of star network topology and bus network topology where a hierarchical order exists and where each node has a fixed arbitrary number of child nodes (here, two).

on a wide geographic zone. Several inter-connected LAN can be considered as a WAN. The internet is an example of a WAN.

3.2.2 Computer networks: communication protocol

Definition 3.2.4

Internet Protocol (IP) *is a family of communication protocols of computer networks used on the Internet. IP protocol enables a unique addressing service for all the inter-connected endpoints.*

Definition 3.2.5

An IP ***packet*** *is a structured unit of data consisting in the association of control information (Called* ***header****) and a payload. The header helps for the delivery of the payload, using source and destination IP addresses and ports and the detection of errors. The payload contains the actual message sent from the source to the destination. It is the only information that is received by the destination of the packet. A pcap is a file containing a packet capture of the network activity (pcap = Packet CAPture). It corresponds to the raw bytes that arrives on a network card. A pcap often contains several packets.*

Definition 3.2.6

Network flows *correspond to a sequence of packets generated by a fixed pair of source and destination IP addresses and ports. Network flow metadata can be extracted to measure, for example the number of bytes exchanged, the duration of the network flow or the number of bytes that traveled in the forward direction. Chinchani et al. [CB05] points out that*

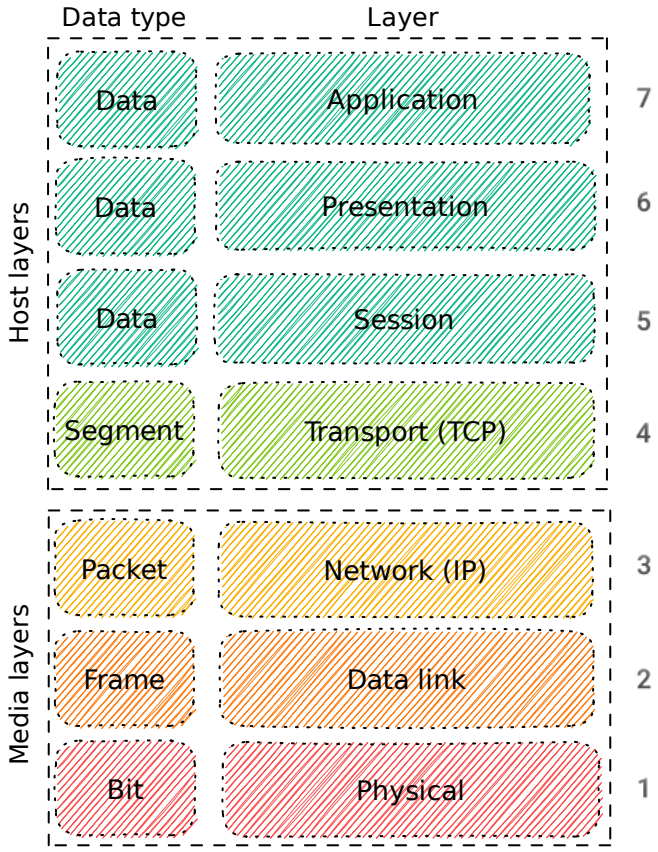


Figure 3.5: The OSI model and its seven layers. The layer 2 and 3 are interesting in LAN and WAN communication. The layer 2 (data link layer) transmit data frames between two endpoints connected by a physical layer. The layer 3 (network layer) send data packets and rely on addressing and routing.

flow-based Network Intrusion Detection Systems (NIDS) savings represents only 0.1% of data volume with respect to the storage of the packet's payload. This optimization works once the text header is removed (under the assumption that the flow corresponds to HTTP traffic) and relies on a data pre-processing. Network flows are extracted processing PCAP files or from the traffic. Tools to extract network flows include the CICFlowMeter [Las+17], Zeek logs [LAR13], etc...)

As seen in Definition 3.2.2, the inter-connected endpoints of a network exchange messages. Network communication is mainly based on the TCP/IP protocols. This protocol is used for both the communication in LAN and WAN. TCP/IP respects the OSI model [Sta87]. TCP/IP is used at large scale as it is reliable to send packets in order and without errors. The UDP protocol is connectionless. Although it is less reliable than TCP, it is used as it requires less processing at the network interface level. The LAN internal communication does not require an internet connection and communicates directly by sending Network frames (OSI layer 3, network layer, view figure 3.5). A packet contains a header and a payload (See Definition 3.2.5). While the payload contains the actual message, the header is used for rooting, containing among other information, destination information. The communication of the LAN with the exterior (Internet connection or WAN connection) can be seen as a connection between two routers added to the LAN internal connection. A router is a device that allows packets transfers (OSI layer 3, network Layer) between the inside and the outside of the computer Network.

3.2.3 Computer networks: a dynamic environment

The network is a dynamic environment. Many changes can occur inside of the LAN network such as:

- Changes in the topology of the network
- Insertion of new computers
- Allowance of new services
- Use of new applications
- Installation of new software
- System updates

Changes outside of a network can also have an impact on the network inside of the LAN:

- Appearance of new services, applications, websites used by the user inside of the network
- Novel Protocols (e.g. 5G)

Definition 3.2.7
We refer as the normal behavior *of the IS to all the behaviors that do not break or attempt to break the security policy of the IS.*

Definition 3.2.8
An ***asset*** *is a data, device or component of a network, that supports informational activities.*

Definition 3.2.9
An information ***security policy*** *is the action plan of a defined group to ensure the protection of its assets. We often refer as a security policy as the security of confidentiality, availability and integrity of a data, a service.*

Both the changes, inside and outside the LAN have an impact on the structure of the network as well as on the traffic generated in the network. These changes imply an evolution of the content of the data navigating on the network through the data packets and network frames that are exchanged by endpoints. These modifications have a consequence on the traffic and thus, have a consequence on the analysis of this traffic. Traffic analysis using static methods is thus intuitively forced to be updated frequently to remain relevant. Furthermore, the aforementioned changes are an opportunity for an attacker to bypass such static traffic analysis methods.

Figure 3.6 provides an example of a company network organization with several interconnected devices, in and outside the internal zone. Several networking zones are accessible through the router and a Demilitarized Zone (DMZ): a low trust zone where a lot of interactions with the internet can be performed. The internal zone contains the company computers and data servers. Its security is reinforced with another firewall. Traffic that comes from the internet is analyzed by both a firewall and an IDS/Intrusion Protection System (IPS).

3.3 Network security

3.3.1 Network security: attacks, intrusions, vulnerabilities

Cyber-security refers to all the protection measures against cyber-attacks [CDP14]. The security of the internal zone is bound to both its data and the available systems.

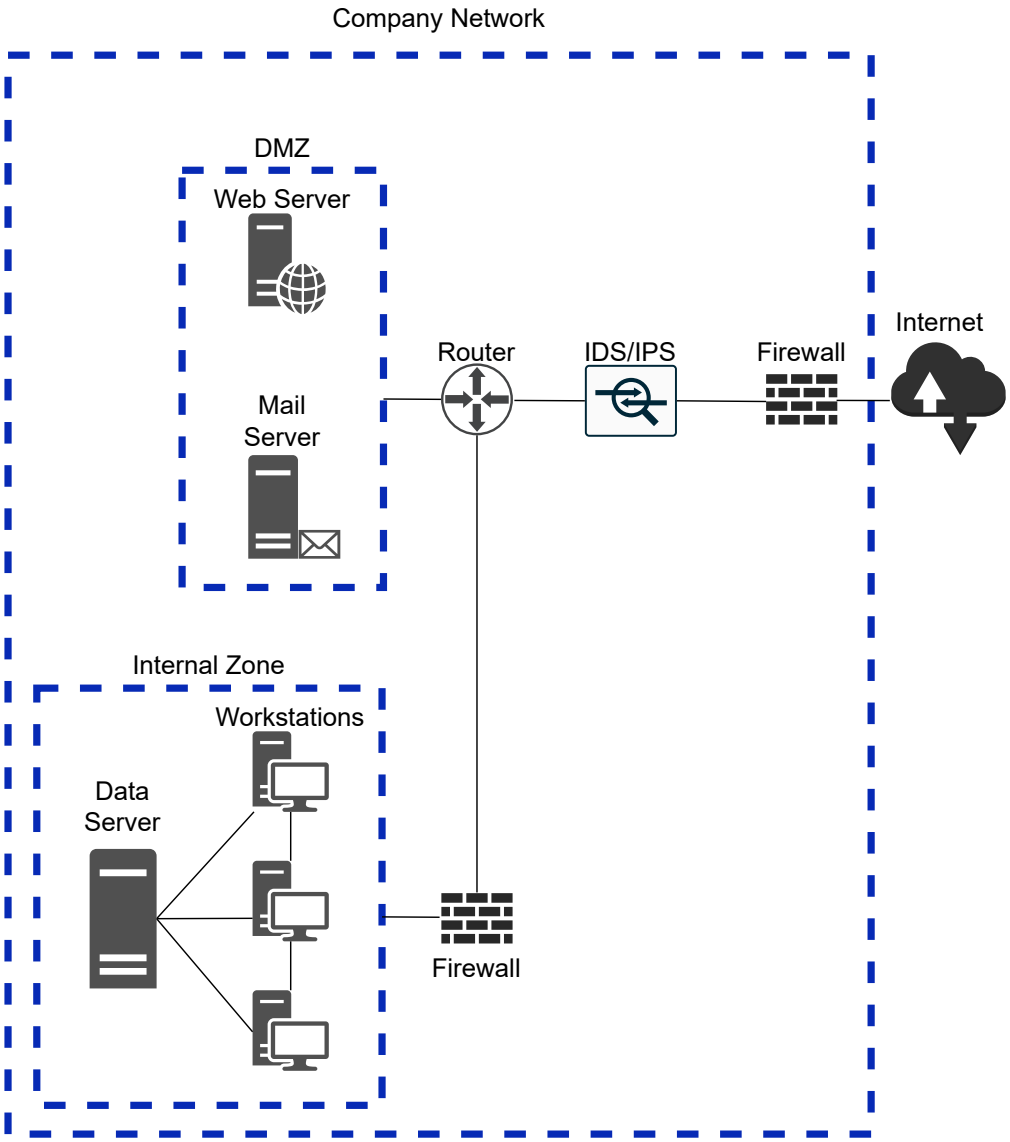


Figure 3.6: An example of a company's network. Traffic incoming from internet goes through a firewall, an IDS or Intrusion Protection System (IPS) and through a router. The incoming traffic is then routed either to the Demilitarized Zone (DMZ) or to the LAN. Diverse Security mechanisms are in place including several firewalls and a IDS/Intrusion Protection System (IPS).

Definition 3.3.1

We define a ***network attack*** *as a deliberate attempt from an unauthorized person to perform unlawful actions on a network. Those actions aim to compromise either the confidentiality, availability or integrity of an asset [HH05].*

The **Confidentiality** guarantees an asset from unauthorized misuse or access. The **integrity** guarantees an asset from unauthorized modifications. The **availability** guarantees the access to an asset to the authorized users in an uninterrupted way.

Example 3.3.1. The following behavior gives an example of confidentiality, integrity and availability compromise: an attacker able to steal credential to one's email address compromises the confidentiality of data. Using its email address to send a phishing link that installs a ransomware (a software that encrypts the data of one's computer) results in a data integrity compromise, once the ransomware is installed. The deletion, by the attacker of content in the mail box, is a data availability compromise.

Definition 3.3.2
We refer as an ***intrusion*** *as a successful compromise of confidentiality, integrity and/or availability of an asset.*

Definition 3.3.3
A ***vulnerability*** *is a security weakness that can be exploited by an attacker. The* ***attack surface*** *is the sum of all system vulnerabilities.*

Definition 3.3.4
An ***exploit*** *is a sequence of commands used by an attacker to take advantage of a vulnerability. An exploit is an intrusion.*

Definition 3.3.5
A ***zero-day attack*** *is the tentative of exploiting a vulnerability that is not publicly known when the attack occurs.*

Definition 3.3.6
An ***attack scenario*** *refers to a sequence of actions performed by an attacker that aims to reach the final intrusion goal of the attacker. An action itself can either be an* attack*, an* intrusion*, or a* benign action*, i.e. a legal activity such as a request for publicly available data.*

The notion of "attack scenario" is close to the "cyber kill chain" which is an abstraction of the attack scenario [HCA+11; PB17].

3.3.2 Network security: the cyber kill chain

Definition 3.3.7
*The **cyber kill chain** is the model of the intrusion process over time [YR15].*

The cyber kill chain is compound of 7 steps [AL15; MPB14]:

1. Reconnaissance: the target is selected and its potential vulnerabilities are explored,
2. Weaponizing: the attacker creates the software that exploits one or several vulnerabilities of the target system,
3. Delivery: the attacker transmits the malicious software to the system,
4. Exploitation: the software is executed and the vulnerability is exploited,
5. Installation: the malicious software installs a backdoor for the external attacker,
6. Command and Control: the malicious software grants a permanent access to the network,
7. Action on objective: the attacker takes measures to reach his goals.

This model is useful to propose different security steps to prevent the attacker to reach her/his objectives:

1. Detect the reconnaissance step and the exploration behavior of the attacker,
2. Reject: prevent the information leaks and un-authorized access,
3. Interrupt or stop the outgoing traffic toward the attacker,
4. Deteriorate the command control of the attacker,
5. Deceive the command control of the attacker,
6. Contain: change the memory representation of the network.

It is important to note that the attack is successful (i.e. there is an intrusion on the network) at the installation step of the cyber kill chain.

3.3.3 Network security: why is security required ?

Intrusions are costly for the individual and for an organization. These costs can be classified in three main categories:

- **Economical cost**: the economical prejudice can be high. It can happen through the theft of data or intellectual property, through a denial of services, through the reparation costs of a compromised service/system, etc. For example, the **Wannacry** ransomware [MP17] costed an estimated amount of 4B$, affecting more than 200,000 computers over 150 countries. The attacker received a cumulative amount of money in Bitcoins, estimated over 300k$.

- **Reputation cost**: intrusions can lead to loss of customer's trust, bad publicity in press,... [Lab15] evaluated the cost of reputation damage of an intrusion to an average of 8500$ for small businesses and over 200,000$ for enterprises.

- **Regulatory fines**: General Data Protection Regulation (GDRP) and other legal framework around cyber-security states that fines could be charged to an organization in the event of a cyber-attack. The GDPR states that some violations could result in fines of up to $12M$ [Pel21].

[BD12] gives an interesting statistic: zero-days attacks last a mean time of 312 days and by the time the vulnerability is publicly known, the amount of the vulnerability exploit is multiplied by a factor 10,000. Unfortunately, each of the vulnerability exploit can be linked with one or several of the aforementioned costs, showing the importance of intrusion detection.

3.4 Detecting intrusions on a network

There exist two main types of IDS :

- Signature-based Intrusion Detection Systems (SIDS) where known behaviors are used to compare with the occurring behaviors [GS08; HS14]

- Anomaly-based Intrusion Detection System (AIDS) where all the traffic is analyzed, and unknown and malicious traffic is spotted to raise alerts [Khr+19; DDW00; Lia+13].

3.4.1 Network data

While network messages circulate through packets, there exist other type of network data that can be used for the detection of intrusion.

Definition 3.4.1
We call network logs *the records of the events that occurred in an application. It contains information about the user and its actions (Access to objects, authentications attempts,...). Most of the time, each line of the network log is a description of a single event. Network logs can be extracted from a firewall, from an IDS/IPS and from most software present on the network.*

3.4.2 Detection: the signature approach

Definition 3.4.2
A ***signature*** *is a search rule used for the examination of the network traffic (a packet or a series of packets). This rule searches for matches in the packet header or/and in the payload.*

Definition 3.4.3
Signature-based Intrusion Detection Systems (SIDS) *are devices or software that filter a data source (traffic, network flows, logs) using signatures of the attacks, defining known characteristics of cyber-attacks [KT03; HS14; GS08].*

SIDS are widely used on nowadays networks. Their main advantages is that their detection is very efficient for known attack behaviors which makes with the firewall a solid first barrier for the attackers to pass. The main drawback of SIDS is the high number of patterns that need to be handcrafted, stored and analyzed, as each attack has a unique signature. Furthermore, the fixed signature base used for the detection can be bypassed by attackers (see example 3.4.1 (2)) by finding a sequence that is sufficient to exploit a vulnerability while being far enough of the matching signature to pass through the in-place security.

As an example, [Hol14] tests on the SNORT signature-based IDS a dataset of 356 severe attacks including 183 attacks seen as zero-days attacks to the rule set. This paper explains how SNORT is able to detect a conservative estimate of zero-day detection of 8.2% due to the fact that zero-days attack deviate from known behaviors. It also shows that more than 54% of the known attacks are accurately detected using SNORT. Event

if half of the attacks are identified by SNORT, this figure shows that the other half of the attack traffic is able to compromise the security of the network. Thus, it seems that a single SIDS is not sufficient for ensuring the security of the network.

3.4.2.1 How are SIDS built?

SIDS are based on user-defined signatures. The signatures are designed from several data sources.

- Common knowledge: using already known attacks and events from the past (years, months, days) to build signatures. Default signature datasets exist on open access.
- An alert raised by a SIDS can trigger a refining of an existing signature.
- An alert triggered by a SIDS can lead to the analysis of an attack scenario and thus, new signatures can be defined for the un-detected attacks.

Figure 3.7 illustrates a simple version of the functioning of SIDS.

SIDS generally use regular expressions as filters. The main drawback of SIDS is the high number of signatures that need to be handcrafted, stored and analyzed. The SIDS is generally paired with the router and filter incoming and outgoing traffic from the LAN.

Example 3.4.1. The **XSS attack** (Cross-site Scripting) is a code injection of HTML code into a website. For example, HTML code can be injected in an authentication field on a website. HTML code is bound to the use of HTML tags between angle brackets ("<" and ">"). A simple way to raise an alert preventing the XSS intrusion is to detect the angle brackets.

The following signature rule – SNORT rule – detects angle brackets and HTML tags:

```
alert tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS
(msg:"Paranoid XSS Cross-site scripting attempt"; flow:to_server,
established;pcre:"/((\%3C)|<)[^\n]((\%3E)|>)/i";
classtype:web-application-attack; sid:9000; rev:5;)
```

The rule is built as follow:

- `alert`: rule action. SNORT will generate an alert if the condition of the rule is met.
- `tcp $EXTERNAL_NET any -> $HTTP_SERVERS $HTTP_PORTS`: observe all the traffic coming from the outside of the network, toward the HTTP network server on the HTTP ports, using the TCP protocol.

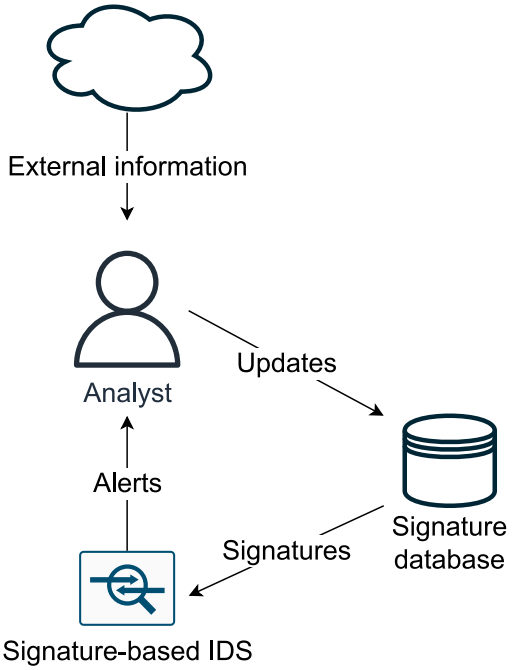


Figure 3.7: The analyst is at the heart of the SIDS. She/he creates a signature dataset which is exploited to correlate the incoming data source (IP packets, network flow, logs) with the known malicious characteristics. The SIDS raises alerts that comes back to the analyst. With the help of external data-sources (documentation, logs, alerts from other security mechanisms), the analyst updates the signature dataset.

- `msg: "Paranoid XSS Cross-site scripting attempt";`: an alert message that will be logged if the condition is met
- `flow: to_server, established;` observe the TCP flow destined to the server, once the session is connected
- `pcre:"/((\%3C)|<)[^\n]+((\%3E)|>)/i";` The actual payload filter. Looking for payload where a HTML tag is present.
  - `(\%3C)|<)` detects the HTML encoding of the character '<' or the character '<' itself.
  - `[^\n]+` detects any sequence of characters that do not contain the new line character.
  - `(\%3E)|>)` detects the HTML encoding of the character '>' or the character '>' itself.

- `classtype:web-application-attack;` classify the alert in a category that is in a more general category of attacks (here : web application attacks).
- `sid:9000; rev:5;` SNORT rule index and revision index.

This SNORT rule:

- is able to detect an XSS attack
- can be bypassed by an attacker
- can raise false positive alerts

1) the xss attack : `<script alert=document.cookies/>` would raise an alert as the HTML code contains a JavaScript code injection that displays the website cookies, it would be a true positive (even though the injected code is harmless).

2) An attacker can still perform an XSS attack embedding the following code into an uploaded file:
`javascript:/*--></title></style></textarea></script></xmp><svg/onload=
'+/"/+/onmouseover=1/+/[*/[]/+alert(document.cookies)//'>` Note that the here above JavaScript intrusion is performed on a specific application and is not generic. This code, embedded in an uploaded file is able to bypass the SNORT rule using JavaScript comment that shadows the HTML tags while still being able to execute the malicious code. (`alert(document.cookies)`).

3) A user using the password `iybzef<hjbaz>zd` would raise an alert, even though she/he is not attacking the information system. This alert is a false positive.

There are four things to learn from the example 3.4.1:

- A signature based intrusion detection can allow efficient attack detection
- Writing a detection rule is a complex task
- There is a way to bypass a detection rule but this way increases the complexity of the attack
- There are cases where the rule can raise false alerts

3.4.3 Detection: detecting anomalies

Definition 3.4.4

AIDS are devices or software that detect deviations of the occurring connections with respect to a known model of correct network behavior [Gar+09; JPP11].

Anomaly detection is a difficult task. It is required as attackers often make changes to already known attacks to evade SIDS and other security mechanisms. For instance, only in the first quarter of 2017, more than 55.000 attack variations were discovered in only 15 attack families [Sym17].

AIDS aim at producing a model of the normal behavior and to detect behaviors deviating from this model. Anomaly-based IDS use statistical inference with either unsupervised learning (a method of learning where no labels are required) or supervised learning (a method of learning where immediate feedback depending on label information is required). Unsupervised learning is widely used as it is possible to analyze the network frames without the need of labeled data [SGA20; ASS19; Don+19]. As stated in [SGA20], there exists a trade-off between a high accuracy using unsupervised learning and a time-efficient, low complexity model. While unsupervised learning is convenient due to its ability to train in real-time and to use unlabeled data, its performances are hindered by high false positive rates. False positives are particularly costly in IDS, as they require analysts to study in depth imaginary intrusions and eventually decrease the confidence of the analysts in the IDS. In practice, FPR needs to be extremely low, for an IDS to be operational.

In terms of practical implementation, retrieving labeled network logs (Definition 3.4.1) to detect an anomaly is a major issue. Indeed, such data require expert engineers to analyze vast amounts of network logs to pick the ones related to an attack.

3.4.3.1 How are AIDS built?

Figure 3.8 illustrates the basic functioning and updating of an AIDS. The AIDS is trained using a Machine Learning (ML) algorithm from a network traffic dataset [AMH16; SV17]. Supervised, Unsupervised or Reinforcement Learning (RL) based AIDS can be designed depending on the data used for training. This dataset contains either:

- only normal traffic: in this case the AIDS computes the difference between the incoming traffic and its conception of normal behavior and raises an alert if too

much difference is found [Abb+14; BMS13; Khr+19]. It is hard for supervised learning algorithms to learn from such dataset

- normal and attack traffic: in this case, the ML algorithm acts as a classifier [Sar+20; ASS19; Khr+19]. From such datasets, all ML methods can be applied

Once trained, the model is used in an **inference mode** and raises alerts that are sent back to the analyst. That is done by exploiting the obtained model on the real incoming data. The dataset can be updated, and the analyst can then re-train the AIDS in order to update it.

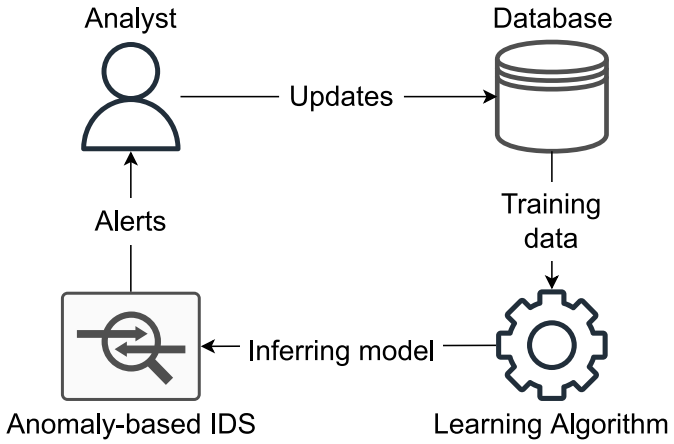


Figure 3.8: The AIDS is trained from a dataset. Once trained, it is used for inference on the network. Raised alerts go back to the analyst. She/he can trigger the re-training of the AIDS from more recent data. The analyst can update the dataset through re-labeling.

3.4.4 Network Security: defense in depth

Cyber-defense, as a security concept, promotes the use of security controls in multiple layers in an IS [Smi03]. For example, using a network firewall at the entry of the company network and before the internal zone (see Figure 3.6) constitutes two security layers. The security controls can be arranged to shelter a sub-network from some intrusions packets. Firewalls can be used to block packets based on their destination or source (IP addresses, TCP ports,...). Firewalls are often used as a router to create a sub-network (a demilitarized zone) separated with respect to the level of trust that is given to each network [Esc98]. An IPS can filter the data directly (if a matching pattern is found for example) or block a user.

3.5 Conclusion

Network security is a complex field due to the high number of variables of this perpetually changing environment.

On the first hand, the network itself changes, with the use of new hardware, new software, services, unfortunately introducing vulnerabilities. It makes the network security a complex task as the security itself has to evolve with respect to the network changes. On the other hand, the user's behavior adapts to these numerous changes and so does the attacker, exploiting the newly created vulnerabilities.

The goal of network security is to secure a network by identifying, protecting, detecting, responding and recovering, in accordance with the NIST framework [Bar+18]. To this end, there exist many security tools such as firewall or IDS/IPS. IDS are probes that study the network traffic and tries to detect malicious behavior, either using already known misbehaviors (Using a **signature** approach) or by finding **anomalies** in the user's behaviors.

The challenges of anomaly and signature probes are to be able to detect accurately those misbehaviors while not generating too many false alarms. Furthermore, those probes should be able to keep up with the incoming traffic. Finally, such probes should preferably be easy to update to remain relevant in the context of a dynamic network environment, with dynamic behaviors.

While SIDS are efficient to help an analyst to analyze known traffic, they perform poorly in the detection of zero-days attacks. AIDS examine the traffic and compare it with an empirical model of the normal and attack traffic. It is more relevant for the detection of the novel attacks but also bound to higher false positive alert rate. This manuscript explores the AIDS capabilities for the detection of intrusions on simulated network traffic. The issue of the false positive alerts is explored as well as the design challenges of such probes, facing biased imbalanced data and dynamic network environments.

CHAPTER 4

Training AIDS using IP network data

4.1 Introduction

Network Intrusion Detection Systems (NIDS) are required for the security of the network. As stated in Chapter 3, it exists two types of NIDS :

- Signature-based Intrusion Detection Systems (SIDS)
- Anomaly-based Intrusion Detection System (AIDS)

AIDS are mostly based on Machine Learning (ML) techniques. This chapter aims to discern the different ML techniques. The considered ML algorithms can be classified as supervised, unsupervised, reinforcement learning and Genetic Programming (GP). The drawback and advantages of each category of ML algorithms in an intrusion detection context are discussed. This chapter also introduces the classification and detection problems and set ground for the requirements of efficient AIDS based on ML. Finally, the Tangled Program Graph (TPG) learning framework is introduced as a technique that can be derived into an evolutive supervised classification task. The TPG is used in this thesis as an AIDS.

4.2 Machine Learning: Learning methods

ML is a wide domain where many methods exist. The here after methods differs by their learning principles, types of data or how data are used in order to train the method.

4.2.1 Supervised Learning

Supervised Learning is a family of training methods for which all training set data samples are labeled with their expected output data [HTF09]. The supervised Learning Agent (LA) is trained to **map the expected output (label) to the input data** using feedbacks under the form of fitness functions and back propagation or mathematical operations such as gradient descent, derivatives and linear algebra. Supervised learning algorithms include among others, **linear regression**, **Support Vector Machine (SVM)**, **decision trees** and most **Neural Network (NN)**.

4.2.1.1 Linear Regression

Linear regression is a simple data modeling method, reflecting a regression problem where the output of the classifier is a single real value. The model is described by a simple equation $Y = aX + b$. The model is mostly used for predicting the $Y \in \mathbb{R}$ from a given X using a and b as the parameters of the model [MPV21; Wei05].

Example 4.2.1. An estimation of the amount of rain (Y) depending on the average cloud cover (X) could be given using a linear regression model (See Figure 4.1).

The problem is to find the value of the parameters a and b that minimize the mean error of the model on the dataset. The example with two variables can be generalized to a larger number of variables.

Definition 4.2.1
Linear classifiers *are linear models that classify data by applying a set of linear combination on the samples.*

4.2.1.2 Support Vector Machines

SVM [Hea+98] is a generalization of linear classifiers [CV95; CDS19]. It builds a linear separation model where the average distance to the line of each sample from each class has been maximized. The SVM model idea is to embed the examples of the dataset into

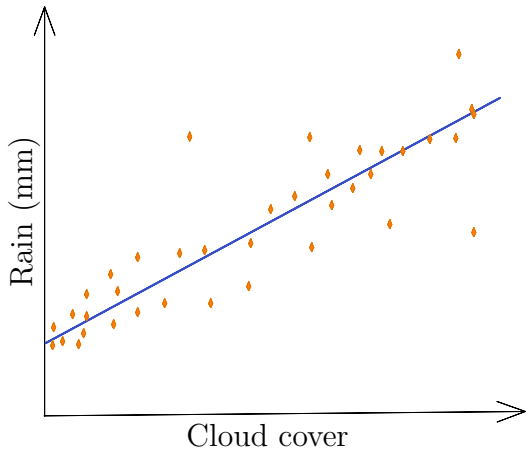


Figure 4.1: Example of a one dimension linear regression. The model can be used to predict the amount of rain depending on the average cloud cover.

a very high dimension space. We know that with a sufficiently high dimension space, any dataset is linearly separable (meaning that a hyper plan separates samples of two classes perfectly). The SVM trick is to do this operation without having to really build up those high dimension representations.

4.2.1.3 Decision trees

Decision trees [CP77] are models composed of a succession of splitting nodes. The entry point of the decision tree is called the root node. Nodes that do not have children are called leaves. In the learning phase, the tree is recursively built starting from the root node, processing the whole dataset. There are two main steps:

1. find the feature that discriminate the best the classes of the sample.
2. define the boundaries to split the dataset on the selected feature.

Then, each sub-dataset is recursively split as described above. In the inference phase, each sample data goes from the root node to one of the leaves, representing a decision. Each node contains a condition. Depending on whether or not this condition is met, the data sample travels to either the left node child or to the right node child. The process recurses until a leaf node is reached.

Example 4.2.2. Figure 4.2 is a simple decision tree model. Each sample data goes from the root node ("*Is it sunny?*") and follows the path depending on the condition. A single decision is taken when the leaf node is reached.

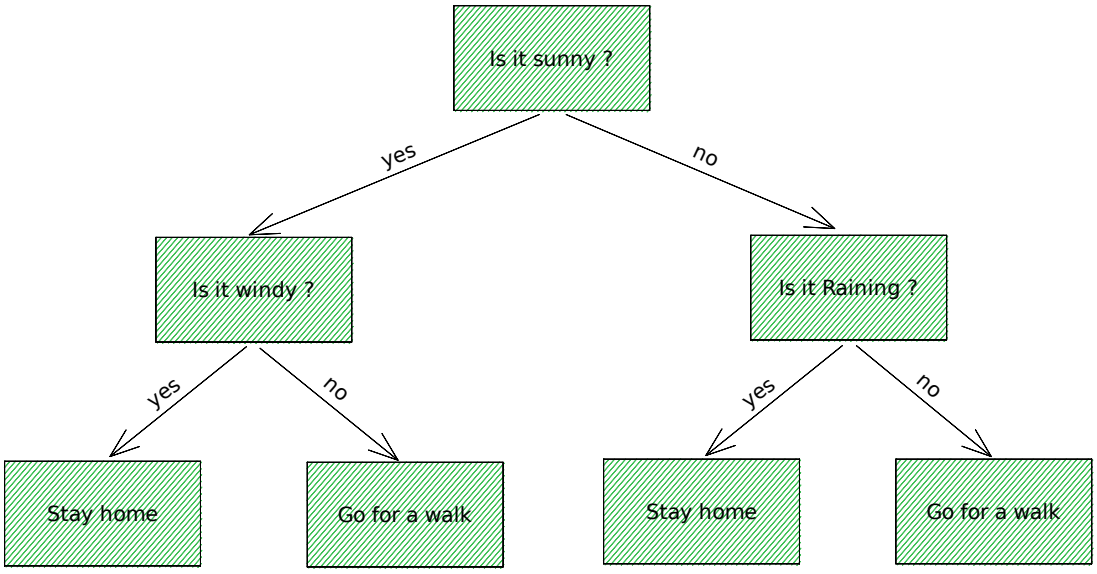


Figure 4.2: Simple example of a decision tree. Input data are weather data and output data is a decision in the set $\{Stay\ home, Go\ for\ a\ walk\}$.

The **Random Forest** algorithm [Bre01] is based on the use of multiple decision trees.

4.2.1.4 Neural Networks

NN are complex networks used for classification [MP43]. A layer is composed of several nodes. Nodes in a layer are interconnected with the nodes of the next layer. In the inference phase, a node receives incoming data, applies a weight and an activation function, and passes the results to the next layer. Depending on the activation of the different nodes, different decisions can be taken. In the learning phase, an optimization technique is used to improve the values of the weights based on the current errors produced by the NN. Convolutional Neural Network (CNN) are complex neural network where nodes can perform a convolution, detecting patterns in the data [LB+95]. CNN are widely used for classification problems.

4.2.2 Unsupervised Learning

Unsupervised learning learns regularities from unlabeled data. When data are represented by vectors in $\mathbb{R}^n$, the problem of unsupervised classifiers is to create groups of samples that are similar, groups being dissimilar from each other. In these algorithms, the notion of similarity between samples is of particular importance. Most of the time, information such as the number of classes is required for the learning process [CA16]. In the following, we introduce common unsupervised learning algorithms such as K-means and Singular Value Decomposition (SVD).

We notice that it also exists a domain in between supervised and unsupervised learning called semi-supervised learning, where the ML method is robust to the missing of one or several label information [Zhu05; OHT20].

4.2.2.1 K-means

K-means [Mac+67] clusters data samples into k groups (called clusters). The separation is based on the computation of the mean distance of the data samples to the separation limit. Although the resulting model is similar to SVM, k-means uses an iterative process to converge toward the optimal solution based on mathematical optimization. Furthermore, K-means do not use labels in its training and can lead to data separations that are not relevant in a classification problem, depending on both how the initialization phase of the learning process was done and the function chosen to measure the distance between samples in order to separate them.

4.2.2.2 Singular Value Decomposition

SVD [WRR03] is a mathematical operation used to find **eigenvectors** on complex matrix. Those eigenvectors are used to accurately separate the data samples. **Principal Component Analysis (PCA)** is a similar method that helps reducing the number of variables, selecting combinations of features containing as much information as possible.

4.2.3 Reinforcement Learning

Reinforcement Learning (RL) [KLM96] is an alternative to both supervised and unsupervised learning. RL is the problem faced by a ML agent that learns behavior through trial-and-errors interactions with a complex and dynamic environment (see Figure 4.3).

The actions of the RL agent have an impact on the environment. When an action is good for the expected problem to solve, the environment rewards the agent. The actions learned as reactions to a set of specific states of the environment is called the policy. RL is known to perform correctly when the environment works under the Markovian assumption that each state depends on the finite combination of the previous state and is a reaction to the agent's policy [NN98]. RL algorithms interact with their environment to find relevant solution to the problem they face. It implies that they are subject to the Explore/Exploit dilemma which consists in finding an equilibrium between the exploration of the environment (that can lead to greater rewards and thus, better policies) and the usage of the current agent's policy (that may cause a lack of observations of important characteristics of the environment). Using this feature, a RL method is able to keep including novelty in its policy and exploitation is particularly interesting for the design of an Intrusion Detection System (IDS) constantly facing new threats.

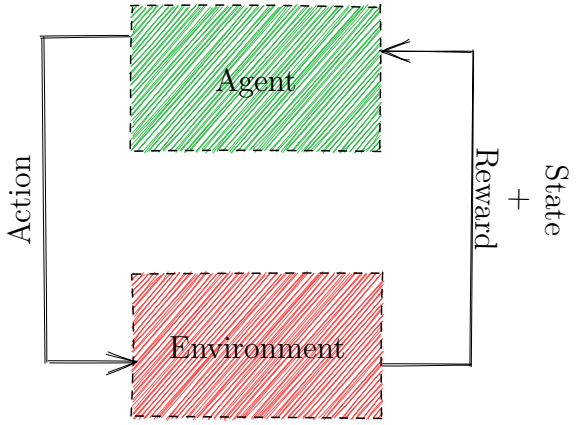


Figure 4.3: Reinforcement learning is the problem of an agent learning from a complex environment. The agent modifies the environment through the use of actions and the environment sends back its state and a reward.

RL training is based on the maximization of a reward. A reward can be seen as a signal, sent by the environment, that implicitly indicates the performance of the RL agent. The reward is sent after an action was taken by the RL agent. Common RL agents are Deep Q-Networks (DQN) [Mni+15], based on the Q-Learning algorithm [WD92].

4.2.3.1 Q-Learning

Q-learning [WD92] is efficient on smaller problems, where the action space (A) and the state space (S) is small. It associates a $S \times A$ sized matrix (the Q-table), storing the

expected reward values for a given state of the environment and for all actions. The chosen action is the one that maximizes the reward.

4.2.3.2 Deep Q-networks

DQN are an optimization of Q-learning based on the use of NN that generalize the agents for greater state spaces and action spaces [SB18]. In case of high dimensions, the Q-table becomes complex to fill with correct reward estimations and the exploration of the environment can be limited.

4.2.4 Genetic programming

GP [Hol92] is a programming technique where a population of individual (often randomly initialized) is set to face a complex task. The individuals that perform badly on this task are discarded, and the others are kept. That is the **selection** and **decimation** phases. The remaining individual goes through cloning and complex **mutation** processes. These mutations results in light modifications of each individual behavior. The selection, decimation and mutation altogether is called a generation. The population is then re-evaluated and the process iterates. The successive generation can be seen as an empirical fine-tuning of a population.

The aforementioned learning techniques can be tuned to be trained using genetic programming. For example, a NN population with random weights can be generated, and the best fitting individuals in that population can be cloned and mutated to improve the classification results of the method. In order to determine whether or not a model is correct, an evaluation is required. This evaluation consists in inferring the learned model on new data samples and to measure its performance.

4.3 Evaluation of classification and detection

The **classification** problem is formalized as a machine learning problem where sample belong to different categories (or classes). The classification task faced by a ML model is to associate a sample with its class. The **detection** problem is a binary classification task where one of the classes is the class to detect, often called Positive (P). Conversely, the other class is called Negative (N).

Accurate prediction of P by the model leads to true positives (tp) while predicting the Positive class on a negative sample results in a false positive (fp). Conversely, we refer as a true negative as the prediction of the Negative class on a negative sample and as a false negative as the negative prediction of a positive sample (fn). A detection model is commonly evaluated [Lev16] with four measures:

- Accuracy (see Eq. 4.1)
- Precision (see Eq. 4.2)
- Recall (see Eq. 4.3)
- F1-score (see Eq. 4.4)

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{4.1}$$

Accuracy is the measure of the amount of correct prediction with respect to the overall number of predictions. A high accuracy testifies about a model taking good decisions.

$$Precision = \frac{tp}{tp + fp} \tag{4.2}$$

Precision is the measure of the number of true positives with respect to the overall number of positive predictions. A high precision is correlated with a model rarely mistaking when predicting the positive class. A model with a high precision can miss Positive samples. Such a model is a good candidate for intrusion detection as false alarms are costly for an operational security analyst.

$$Recall = \frac{tp}{tp + fn} \tag{4.3}$$

Recall is the measure of the number of true positives with respect to the overall number of positive samples. A model with a high recall is a model that rarely misses a Positive sample. Such a model can generate false positive alerts.

$$F1 - score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{4.4}$$

F1-score is a measure that gives as much importance to the recall and the precision (i.e. to the false positive and false negatives). It compares the number of true positives to the number of total errors of the model.

The **ROC curve** is the representation of the True Positive Rate with respect to the False Positive rate. Area Under Curve (AUC) is defined as the area under that curve.

4.4 Learning from network data

Network anoaly detection is often done at the layer 3 and 4 of the OSI model (se Figure 3.5). The network frames and packets can be analyzed. Information on the network connection can be extracted under the form of features and be used for the training of learning-based IDS.

4.4.1 Network data: NIDS datasets

Network Intrusion Detection Systems (NIDS) datasets generally comes in two formats. Some datasets use network packets (or PCAP) while other use network flow or network logs. These types of data are structured and have standard formats.

4.4.2 Packets, Network flows and logs

We refer to Packets network flows and logs as defined (respectively) in Definition 3.2.6 and 3.4.1. Packets have a structured header and a variable payload size. Minimum packet size is 64B and its maximum size is 64KB. Network flows are structure data extracted from pcap files. The size and structure of a network flow depends on the processing. For example, the CICFlowMeter software extracts 80 features from PCAP files. An unoptimized storage of such a data through the use of double precision values would be 640B per network flow. Finally, logs are also structured data that vary depending on the source of the log or the application used to extract it, but its size is not necessarily bound as it contains textual description of events. The log file size has a high bound (for example 1GB under UNIX systems) in order to prevent process errors and operations such as log rotation exist in order to keep logs for a longer amount of time.

4.4.3 NIDS datasets

M. Ring et Al. [Rin+19] published a survey of NIDS datasets. Among them, some such as CIDDS-001 or CICDDoS19 are flow based and generally more compact as they sum up meta-information of the traffic. The DARPA dataset [MIT99] comes with PCAP

information. Some other datasets such as the CICIDS 2017 dataset [SLG18] gives both PCAP and network flow information.

The CICIDS 2017 is a recent labeled network intrusion dataset originally designed to meet the needs of having representative network data for Network intrusion Detection [SLG18]. Even though a revision of this dataset is proposed in [ERJ21], correcting some of the label information, the original CICIDS 2017 dataset is mostly used for the experimental parts of this thesis, in order to both, study the impact of the dataset errors on the ML method proposed and to be comparable with the state of the art methods.

4.4.4 Network data: the imbalance nature of the intrusion detection problem

Attack traffic coming from outside of the Information System (IS) toward the inside of the IS represent a small amount of traffic with respect to a normal amount of connection resulting of normal activity. Realistic network intrusion detection datasets should take into account that there are usually fewer attacks on a network than harmless traffic. This issue impact the ML training of the IDS as realistic data are by nature imbalanced and thus statically harder to learn from [HG09].

4.5 Using ML algorithms for Network security through the design of an AIDS

The ideal AIDS is a probe that

- triggers as few false alerts as possible (We saw in Chapter 2 that 99.67% precision is a minimum on the CICIDS 2017 dataset),
- is resilient to mislabeling and data imbalance as they are both present in NIDS datasets,
- enables continual learning, to fit the changes occurring on the network,
- enables an efficient learning phase, to keep pace with the incoming traffic,
- can learn with missing labels.

4.5.1 Supervised Learning

Supervised learning is intuitively a good candidate for network security. It permits efficient classification with the use of CNN and reaches high precision. The drawbacks of this method is that the learning depends on the presence of accurate labels which is hard to obtain. Thus, the training of supervised learning method faces at least one of the two issues:

- errors in label cause a degradation of the detection of intrusion and produces a model with degraded capabilities
- the model over-fits the dataset leading to costly errors under operational inferring conditions

Furthermore, ML-based models are static and thus, are not sustainable solutions for long-lasting inference.

4.5.2 Unsupervised Learning

Unsupervised learning major strength is that it does not depend on data labeling. It makes it easier to obtain qualitative data to train the model as unlabeled PCAP data are sufficient. Unfortunately as an attacker tries to be as close as possible to a normal behavior, it seems easier to bypass such methods. The challenge on the use of unsupervised learning is thus to find accurate data separation. Furthermore, learning from unlabeled data is computing intensive making it difficult to enable both flux processing and continual learning.

4.5.3 Reinforcement Learning

Reinforcement Learning is adapted to timely problems where the environment's state at $T = t + 1$ depends on both the environment's state at $T = t$ and on the action taken by the RL agent. Such dependence is met even when no actions are taken on the network for a fixed pair of source and destination IP addresses and ports. The supervision required by the RL agent is relaxed with respect to the one needed by supervised learning algorithms. RL algorithms are sometime used as classifiers, that usually converge slower than supervised learning algorithms as they require no immediate supervision.

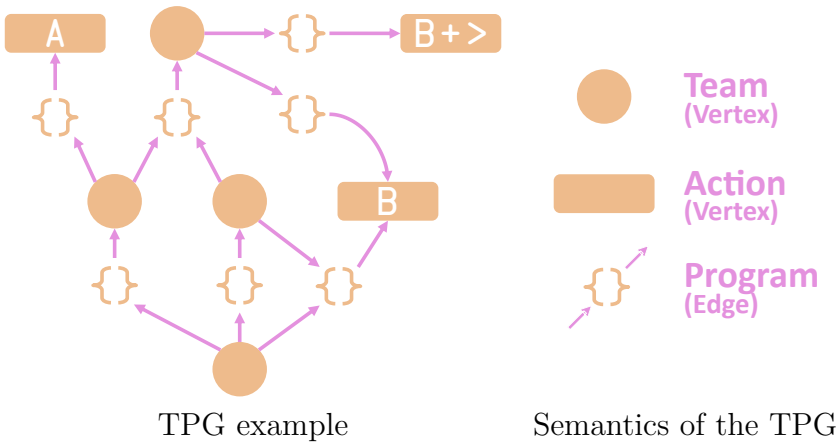


Figure 4.4: Example of a Tangled Program Graph (TPG)

4.5.4 Genetic Programming

Genetic programming used with any of the previously described methods will permit incremental learning, updating continuously the model and making it able to detect novelty along the monitoring.

4.6 Experimented GP method: Tangled Program Graphs (TPG)

The TPG framework [KSH] is a GP-based RL method which weightlessness and agility is interesting for the intrusion detection problem. The capabilities for intrusion detection of the TPG is deeply studied in this thesis. We describe here the TPG, as introduced by Kelly and Heywood [KSH].

4.6.1 TPG: Model and Learning Algorithm

The semantics of the Tangled Program Graph (TPG) model consists of three elements composing a directed graph: *programs*, *teams* and *actions*. The *teams* and *actions* are the vertices of the graph, *teams* being internal vertices while *actions* are leaves of the graph. The *programs* are associated to the edges of the graph that each connects a source *team* to either a destination *team* or a destination *action* vertex. Self-loops, that is an edge connecting a *team* to itself, are not allowed in TPGs.

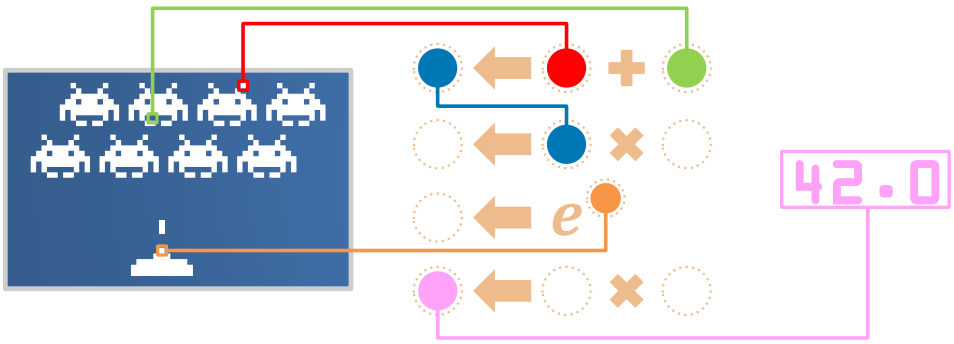


Figure 4.5: *Program* from a TPG. On the left, the learning environment state fed to the *program*. In the middle, the sequence of instructions of the *program*. On the right, the result produced by the program.

A *program* can be seen as a black box that takes the current state of the learning environment as an input, processes it, and produces a real number, called a *bid*, as a result. In more detail, a *program* is a sequence of simple arithmetic *instructions*, like additions or exponents. Each *instruction* takes as an operand either data coming from the observed learning environment, or the value stored in a register by a previous *instruction*. The last value stored in a specific register, generally called `R0`, is the result produced by the *program*.

The execution of a TPG starts from its unique root *team*, when a new state of the environment becomes available. All *programs* associated to outgoing edges of the root *team* are executed with the current state of the environment as their input. Once all *programs* have completed their execution, the edge associated to the largest *bid* is identified, and the execution of the TPG continues, starting from the destination *team* or *action* of this edge. If another *team* is pointed by this edge, its outgoing *programs* are executed, still with the same input state, and the execution continues along the edge with the largest *bid*[1]. Eventually, the edge with the largest *bid* leads to an *action* vertex. In this case, the *action* is executed by the learning agent, a new resulting state of the environment is received, and the TPG execution restarts from its root *team*.

The genetic evolution process of a TPG relies on a graph with several root *teams*. The initial TPG created for the first generation only contains root *teams* whose outgoing edges each lead directly to an *action* vertex. At a given generation of the learning process, each root *team* of the TPG represents a different *policy* whose fitness is evaluated. Evaluating a root *team* consists of executing the TPG stemming from it a fixed number of times, or until a terminal state of the learning environment is reached, like a game-over in a video

[1]If a team is visited several times, previously taken edges are ignored to avoid infinite loops.

game. The function used to evaluate a root *team* based on the action it took is called the fitness function. The fitness obtained after evaluating each root *team* of the TPG is used by the evolution process. Worst-fitting root *teams*, which obtained the lowest fitness, are deleted from the TPG.

To create new root *teams* for the next generation of the evolution process, randomly selected remaining *teams* from the TPG are duplicated with all their outgoing edges. Then, these new edges undergo a random mutation process, possibly altering their destination vertex, and modifying their *programs* by adding, removing, swapping, and changing their instructions and operands. Surviving root *teams* from previous generations may become the destination of an edge added during the mutation process, thus becoming internal vertices of the TPG. This mutation mechanism favors the emergence of long-living valuable sub-graphs of connected *teams*. Indeed, useful *teams* contributing to higher fitness have a greater chance of becoming internal vertices of the TPG which can not be discarded unless they become root *teams* again. Hence, complexity is added to the TPG adaptively, only if this complexity leads to better fitness for the learning agent. Finally, it is important to note that a *team* from the TPG may be used by several policies stemming from different root *team*. Hence, a valuable behavior discovered by a single root *team* at generation n may be reused by other root team in subsequent generations. A detailed description of this evolution process can be found in [Kel18].

The capabilities of TPGs have been demonstrated [KSH; Kel18] on automatically playing the 55 video games from the Arcade Learning Environment (ALE) [Bel+13]. In this learning environment, the adaptive complexity of TPG leads to models with diverse sizes, depending on the complexity of the strategies developed to play each game. For example, there are two orders of magnitude between the smallest and largest networks built within these learning environments. On the performance side, TPGs have been shown to reach a level of competency comparable with state-of-the-art deep-learning techniques on ALE games, for a fraction of their computational and storage cost. Extensions of the TPG model have been proposed to support continuous action spaces in order to target new learning environments, like time-series predictions [KH17b], and to GP-based support classification [KH17a]. This last GP-based classification proposal motivates the chapter on imbalanced classification.

4.6.2 Parameters of the TPG

In this manuscript, an effort is made for the reproducibility of the results. We present the parameters of the legacy TPG in Table 4.1 and provide a quick description of them associated with their default values. Additional parameters of the Generic Evolvable Graphs for Efficient Learning of Artificial Tangled Intelligence (GEGELATI) TPG library are described in Table 4.2

The parameters selected for each contribution will be detailed in the corresponding chapters.

Default parameters of the TPG has been studied and defined in [Kel18].

4.7 Conclusion

This Chapter explains the context of ML in the intrusion detection domain. ML exists under different forms and each of them presents drawback and advantages in a NIDS context. The learning of the ML techniques is conditioned by the available data. Thus, network intrusion data used for the training of an AIDS must be qualitative and represents as much as possible the reality of the network environment.

Through this chapter important intrusion detection problems are introduced. Indeed, obtaining representative and qualitative network data is a complex task. Then, once qualitative data is obtained, one must, while designing its ML model, use it in a fair way and deal with the imbalanced nature of the detection task where intrusions are less represented than normal traffic. Finally, obtaining a fair ML model trained on a NIDS dataset does not guarantees that the model is usable in operational conditions, knowing that the network environment is dynamic and zero day attacks emergence. To this end, the TPG framework is introduced as a way to provide an adaptive method based on the incremental nature of the genetic programming process. This thesis explores the capabilities of the TPG framework in an intrusion detection context.

Parameter Name	Description	Default Value
maxNbActionsPerEval	Maximum number of actions performed on the learning environment during each evaluation of a root	1000
maxNbEvaluationPerPolicy	Maximum number of times a given root is evaluated	1000
maxProgramSize	Maximum number of Line within the Program of the TPG	96
pAdd	Probability of inserting a line in the Program	0.5
pDelete	Probability of deleting a line in the Program	0.5
pMutate	Probability of altering a line of the Program	1.0
pSwap	Probability of swapping two lines of the Program	1.0
maxInitOutgoingEdges	Maximum number of Edge connected to each TPGTeam of the Graph when initialized	3
maxOutgoingEdges	Maximum number of outgoing edge during Graph mutations	5
nbRoots	Number of root Teams to maintain when populating the Graph	100
pEdgeAddition	Probability of adding an outgoing Edge to a Team	0.7
pEdgeDeletion	Probability of deleting an outgoing Edge of a Team	0.7
pEdgeDestinationChange	Probability of changing the destination of an Edge	0.1
pEdgeDestinationIsAction	Probability of the new destination of an Edge to be an Action	0.5
pProgramMutation	Probability of mutating the Program of an outgoing Edge	0.2
nbGenerations	Number of generations of the training	500
nbIterationsPerPolicyEvaluation	Number of evaluation of each root per generation	5
nbRegisters	Number of registers for the Program execution	8
ratioDeletedRoots	Percentage of deleted (and regenerated) root Vertex at each generation	0.5

Table 4.1: Default parameters of the TPG framework.

Parameter Name	Description	Default Value
Archive Size	Number of recordings held in the Archive	50
archivingProbability	Probability of archiving the result of each Program execution	0.05
maxConstValue	Maximum constant value possible	100
minConstValue	Minimum constant value possible	-10
pConstantMutation	Probability of each constant to be mutated	0.5
forceProgramBehaviorChangeOnMutation	Makes sure a Programs behavior changes when mutated	false
nbProgramConstant	Number of Constant available in each Program	0
nbThreads	Number of threads used for the training process	1

Table 4.2: Additional parameters of the GEGELATI TPG library.

Part II

Contributions

CHAPTER 5

Contribution 1: Assessing the biases of IP networks intrusion detection datasets and evaluating their effect on a Tangled Program Graph (TPG)-based Anomaly-based Intrusion Detection System (AIDS)

5.1 Introduction

Learning based Anomaly-based Intrusion Detection System (AIDS) are used in IP networks as an efficient method to reinforce the security of a network. These probes are trained on either real data collected on a company network or using synthetic data from generated or simulated datasets. In the design process of the learning based AIDS, the choice of the training dataset is important as it conditions the type of input data and its characteristics. The learning process and the inferring results of the produced model depend on the training data. The model produced by the Machine Learning (ML) intelligence, studying batches of specific data, is subject to the data biases that exist in the dataset [GD95; PKP21]. Choosing a Network Intrusion Detection Systems (NIDS) dataset implies choosing implicit biases that will condition the learning. Those biases can take different forms and have diverse effects on the learning process. It is important to consider that, for operational constraints, the training data used in the modeling should

mirror the network reality. Thus, training network data require to be similar to user generated network data in order to perform an accurate detection.

This chapter aims to examine the different data biases existing in the NIDS datasets, their potential impact on the training process and the mitigation that can be applied to the data for better consistency. Furthermore, this chapter shows the Tangled Program Graph (TPG) framework robustness to biases in the labeling of the dataset.

This chapter answers three main questions:

- what is the influence of biases on the learning process?
- how to fairly and impartially test, in an un-biased way, the produced decision model?
- how to create fair NIDS datasets while being realistic compared to a real incoming network traffic?

Finally, this chapter proposes a procedure in order to identify biased features. Furthermore, it is shown that the TPG, is a good candidate for intrusion detection, where labels are often subject to biases.

5.2 Related Work

When it comes to learning application for real-life use-cases, the choice of the training data must be relevant enough to produce a realistic training environment that mirrors the reality in which the application might be used. Considering the intrusion detection problem, many limitations can be found in training dataset that can lead to a decision model inaccurate in operational conditions. There exist eight of these limitations, called "biases" [GHF22; Meh+21].

- **Deployment bias**: The model is used for inference in a context differing from the one it has been built into [Meh+21; Olt+19]. A NIDS trained on a dataset and used in real-life condition is subject to a Deployment bias. The bias happens when characteristics of the training and inferring networks differ. In [MLK14], authors state that the dataset they use for training a Signature-based Intrusion Detection Systems (SIDS) (DARPA 1999 [**darpa_dataset**]) is not sufficient to prove operational performances and that another testing dataset should be used to get rid of this bias.

- **Representation bias**: This bias appears when the input data do not represent the reality [Lan+10]. For example, in the CICIDS 2017 dataset [SLG18] the simulated traffic is compound of intrusions and background traffic. The attack traffic results in successful intrusions scenario whereas no unsuccessful attack scenario exists in the dataset.

- **Measurement bias**: The features and labels available in the dataset are not real variables of interest useful to characterize the problem [SG21].

- **Label bias**: Errors in the labeling can be found in the dataset [BB17]. Label bias exists in NIDS dataset when the labeling of the data is performed by already existing Intrusion Detection System (IDS). It leads zero-days attacks to be labeled as "sane traffic". Some datasets are known to have incorrect or missing labels such as in the CICIDS 2017 dataset [ERJ21].

- **Algorithmic bias**: The construction of the model leads to deviations of the outcome [dOL17; FN96]. An algorithmic bias exists for example in the model trained on imbalanced datasets [MG09; Vet21]. As IDS datasets are mostly imbalanced, the model can tend to identify the different samples to the most represented classes of the dataset [AI19]

- **Evaluation bias**: Inappropriate testing population or evaluation metric is used [Meh+21; Olt+19]. The DARPA 1999 dataset is compound of two parts. One part is made of background traffic, and the other is made of attack traffic. [McH00] points out several evaluation biases. For example, in this case, the performance measurement of the IDS is based on the number of sessions. In this special case multi-sessions attacks can reduce in an optimistic way the number of false positive alarms as the denominator would be larger when a false alarm is reported.

- **Social bias**: The population of experts that create the dataset is biased [Olt+19]. For example in the design of a NIDS dataset, a ratio above normal of attacks can be recorded.

- **Feedback bias**: This issue appears as feedback on the ML method can be sent only for samples that have been analyzed by the algorithm [Meh+21; Olt+19].

The existing bias can be classified in two main categories:

- Biases of the data
- Biases of the algorithm

We refer as fairness as the property of a ML-based model that ensures that biases in the data do not lead to biases in the model output [OC20]. In order to increase fairness in the learning process of IDS, one can mitigate the biases of its algorithm and take into account the biases of the data. Biases of the algorithm are deployment, algorithmic and feedback biases whereas the remaining (representation, measurement, label, evaluation and social biases) is linked to the data.

5.2.1 Reducing biases in the algorithm

Methods exist in order to reduce the bias of the algorithm used for intrusion detection. [MLK14] states that testing operational functioning of an IDS on a sub-part of the training dataset is not sufficient to prove the efficiency of the solution. This is mainly due to the differences that can exist between the training and operational network environment. Pessac et al. [PS22] state that fairness in ML can be measured in five different ways (Disparate Impacts, demographic parity, equalized odds, equal parity and individual fairness). Among them, the Individual fairness [DI18; Jos+16] measure seems relevant in the intrusion detection context, as it requires similar samples to be treated similarly. As there exist several types of attacks, and as some are totally different from the others, creating a single IDS able to identify with the same precision all attacks is a complex task. Although it seems obvious, fairness in comparison of several methods comes with identical input data, and identical testing conditions as stated in [Che+21].

5.2.2 Pre-processing datasets for fairness

The KDD dataset [**kdd_dataset**] is based on a feature extraction of the DARPA dataset.

In [Tav+09] KDD dataset [**kdd_dataset**] is pre-processed in order to remove all the duplicated records of the dataset and to produce the NSL-KDD dataset. Abdulraheem et al. [AI19] has a similar approach and reduce the size of the KDD dataset by up to 97%. These pre-processing can be seen as a representation bias mitigation as the redundancy of the features can lead the ML to unfair decisions based on to much weight given to these redundant features.

Sarhan et al. [SLP22] propose a set of 43 features to fairly train IDS. The pre-processing for the extraction of these features results in a reduction of deployment bias as it promotes the use of a standardized feature set for four different NIDS databases. They promote the use of Network flow based features as they are light and can be easily extracted by a router or a switch. Authors show that the proposed feature set is adequate for the design of NIDS. Although they are important features for operational intrusion detection, the conservation of the IP source and destination address keeps a representation bias inside of the dataset, as most datasets are simulated with one or several fixed attacker. This study is complementary with the work of [MLK14] as the standardization of the useful features for intrusion detection helps to train and to infer on different datasets.

Similarly, the information gain of the features of the dataset can be measured [KZH05]. The information gain results from the computation of the entropy of samples split by a given feature. If a split separates the population in two groups of different classes, the information gain is high. Conversely, low information gain indicates that splitting a group with respect to the chosen feature does not helps in the classification of the samples. The information gain is often used for the training of classification trees. A biased feature with high information gain is intuitively dangerous for the training of an IDS. Other studies perform feature extraction through the use of Auto-Encoders [YH18; MK18] or other ML techniques [MK18; ASS20]. Zhang et al. [ZD21] try to increase fairness using adversarial networks. This study to de-correlates the representation bias from the dataset sensitive attributes. In [DR20], representation and algorithmic biases in anomaly detection are explored. The outcome fairness is measured with respect to "protected status variables" and by finding combination of these variables in the outlier class and in the normal class. Similarly, Deepak et Al. [DA20] explain the importance of sensitive attributes in the design of fair outlier detection algorithms and propose three heuristics (Neighborhood diversity, Apriori distribution and Attribute asymmetry) to enhance fairness. Although the former works focused on fair outlier detection for people, the idea of measuring fairness in the output with respect to protected status variables is useful for intrusion detection. For example, in the original version of the CICIDS 2017 dataset [SLG18], the attacks comes from a set of four computers generating only attack traffic. The attack detection using simply the source IP addresses of the attackers would be effective in this context, but biased.

5.3 Impact of learning biases in NIDS

This section aims to define the impact of the label biases on the learning and the fairness of a TPG-based AIDS. We here formalize the representation bias and the label bias as they are the main focus of this chapter. This discussion is mostly illustrated on the CICIDS 2017 dataset as a recent study proposes a re-labeling of the dataset in order to increase its fairness [ERJ21].

5.3.1 Problem definition

The problem of learning from biased data can be defined as the following:

Supposing two datasets:

- A dataset B divided in a training set B_{tr} and a test set B_{te}
- the ideal fair dataset F representing the same data as B divided in a training set F_{tr} and a test set F_{te}

Samples are named $\sigma_{F_{tr}} \in F_{tr}$, $\sigma_{F_{te}} \in F_{te}$, $\sigma_{B_{tr}} \in B_{tr}$ and $\sigma_{B_{te}} \in B_{te}$.

Samples $\sigma_F \in F_{tr} \cup F_{te}$ and $\sigma_B \in B_{tr} \cup B_{te}$ correspond in the way that there exist indexes i and j, such as σ_{F_i} and σ_{B_j} represent the biased and un-biased records of the exact same event occurrence on the network.

Each sample σ belongs to one of two classes $\{P, N\}$ where P denotes the positive class and N the negative class. The class associated with the sample σ_i is $l_i \in \{P, N\}$. A classifier is a function m that associates a sample with a single label l'. $m|\sigma(i) \mapsto m(\sigma_i) = l'_i$.

Typically, label bias exists when $l_{F_i} \neq l_{B_j}$.

Supposing two classifiers identical but trained on different data: $m_F|\sigma_i \in F_{Tr} \mapsto m_F(\sigma_i) = l'_{F_i}$ and $m_B|\sigma_j \in B_{Tr} \mapsto m_B(\sigma_j) = l'_{B_j}$, there is no label bias present when for $\sigma_i \in F_{Te} \cup B_{Te}$, $m_B(\sigma_i) = m_F(\sigma_i) = l'_i$.

5.3.2 Learning with representation biases

Representation biases are present in a dataset when the available features do not truly represent the learning problem. Intuitively, it results in setting too much attention to these specific features that facilitate the detection of the positive events without being necessarily relevant for the detection of positive events in a real context. In the CICIDS

2017 dataset, the positive events are generated from a set of four attack computers during specific time frames. The identification of the network flow logs (using the destination and source ports and IP addresses) or the timestamp of the network flow logs are both examples of representation biases. In the first case, the identification of the log is a clue for the detection but is not a required condition to qualify the traffic as an intrusion or as sane traffic. For example, the use of a computer to generate attack traffic once does not imply that all traffic coming from this computer is harmful. In the second case, the presence of the timestamps of the attack in the training dataset is questionable since the generation of the dataset simulates attacks at fixed time. Intuitively, a model detecting intrusions based on the timestamps of the network flow logs is really likely to be **over-fitting** the data rather than being an adequate classifier.

5.3.3 Learning with label biases

Label biases are biases existing in datasets where one or several labels are incorrect. In intrusion detection, this bias exists in datasets that qualify safe network traffic as attack traffic or inversely. This issue also exists in the CICIDS 2017 dataset [ERJ21] where unsuccessful attacks are considered as intrusions and some of the succeeding attacks are considered as safe traffic. Furthermore, in [ERJ21], a distinction in the labels is made between intrusions, i.e. successful attacks, and unsuccessful attacks. In this chapter, we study the impact of label bias on the learning of a Genetic Programming (GP)-based NIDS. In particular, we aim to measure the differences of identical classifiers per-class results having correct and incorrect labels.

5.4 Experimental Setup

We focus on the observation and mitigation of data biases. Among them the representation bias is particularly interesting. A prior pre-processing of the dataset can contribute to its reduction.

5.4.1 The CICIDS 2017 dataset

The CICIDS 2017 dataset is one of the intrusion detection datasets with the most diverse and realistic range of cyber-attacks. It addresses recent attacks that are not available in other datasets using a range of different computers, operating systems and security

features [PB18]. It has been generated using two networks. The first one, the victim network, is a set of five servers and 10 computers using different operating systems (Windows, Linux and Macintosh) and necessary devices such as routers, firewalls and switches. The attacker's network includes four computers using Windows 8.1 and Kali operating systems, one router and one switch. The CICIDS 2017 dataset contains a week of generated traffic network frames. In this traffic several anomalies labeled in 14 different categories can be found. Most of the traffic is labeled as "normal traffic". The attacks are highly unbalanced, but this disparity does not reflect a usual behavior on an Information System (IS). Indeed, more than 16% of the traffic summaries in the CICIDS 2017 dataset represent attack traffic whereas it is believed that the attack ratio is under 0.002% which is the amount of attack traffic in the KDD dataset. Table 5.1 summarizes the different classes and the amount of data per class.

The dataset is separated in two subsets. 25% of the dataset is randomly extracted from the legacy dataset and constitutes the testing dataset. The remaining samples are kept for training.

Table 5.1: Distribution of classes in the CICIDS dataset in network flow logs. Each network flow log corresponds to 78 fields and 312 Bytes of raw data [SLG18].

Classes	Amount of network flow logs
BENIGN	2.359.087
Dos Hulk	231.072
Port-Scan	158.930
DDoS	41.835
Dos Golden-eye	10.293
FTP-Patator	7.938
SSH-Patator	5.897
Dos Slow-loris	5.796
Dos Slow-httptest	5.499
Bot	1.966
Brute force	1.507
XSS	652
Infiltration	36
SQL-Injection	21
Heart-bleed	11

The data used in our study is taken from the fully labeled CICIDS 2017 dataset. It sums up, in a .csv file, 78 network flow features from the captured network traffic (PCAP files). Information such as the destination port, the number of bytes per second or flags can be found in the dataset. Those information are represented as 32bits integers, floating-point numbers and Boolean. Detailed information about the 78 extracted features have been defined and explained on the CICFlowMeter web-page [Cyb].

As a first experiment, a legacy TPG [KSH] is trained on the CICIDS 2017 dataset. A second training on the same dataset is performed with a TPG adapted for classifications problems by taking as a reward the F1-score of the classification (this will be used for comparison in Chapter 6).

5.4.2 Experiment 1: representation bias of an IDS

As a first part of the experimental study of biases in intrusion detection, the representation bias of an IDS is studied. To this extent, a TPG is trained on the CICIDS 2017 NIDS dataset. The legacy TPG operating mode is based on the observation of specific parts of the state of the environment (i.e. in our case specific features of the network flows). We first train a TPG using the original CICIDS 2017 NIDS dataset and then observe the features that are used for classification. The traffic simulated for the dataset generation, as described in [SLG18], uses an attack network made of a set of four computers intruding a local network at regular time, it is expected that the TPG focuses on the identification of the malicious network flow features preferably based on the source IP and port and destination IP, as well as using the timestamp of the connections rather than using other available features. The experiment aims to train three different ML classifiers.

- $M1$ is trained on the full version of the CICIDS 2017 dataset and infers on a similar test dataset.
- $M2$ is trained on the full version of the CICIDS 2017 dataset and infers on a test dataset where white noise is synthetically added to the supposedly biased features.
- $M3$ training and test sets are edited with additional white noise to these particular features.

The attention of the agent to these specific features is studied as well as the classification results.

5.4.3 Experiment 2: label bias of an IDS

To reveal the impact of label bias, the legacy CICIDS 2017 and its revision, where labeling bias has been corrected, as described in [ERJ21] datasets are used. In order to enlighten the label bias, a specific focus is put on the DDoS and DoS Hulk attack as it is described as the attack class with the most incorrect labels.

Two classifiers are used. $M4$ is trained and evaluated on the legacy CICIDS 2017 dataset. $M5$ model's results after inferring on the corrected version of the CICIDS 2017 dataset are studied. Conversely, $M5$ is trained and evaluated on the corrected version of the CICIDS 2017 dataset. It is important to note that biased features as described in Section 5.4.2 are removed from the training set. The detection rates of each evaluation are studied.

Finally, to show the impact of the label bias on the TPG model, we study the inferring performances of $M4$ where a fraction of the intrusions are progressively re-labeled as sane traffic during the training phase and show the resilience of the TPG framework to inaccurate data labels.

The performance function used to train the TPG for Experiment 1 and Experiment 2 consist in a $+1$ score for each correct decision and a -1 score for each incorrect decision. This score is normalized on the overall number of decisions taken to give a single floating point value between -1 and 1.

5.4.4 Parameters of the TPG

Table 5.2 displays the parameters used in this Chapter to train TPGs.

These parameters are chosen so as to provide a fast prototyping TPG that converges rapidly. In particular, 10 000 samples are evaluated per generation raising the probability of each team to observe samples from different classes.

The parameters of the TPG as defined in [Kel18] are used. The number of actions taken, the number of root teams and the number of evaluation required are designed to widen the exploration space of the TPG. Due to the consequences of these parameters, the size of the programs have been reduced to reduce the execution time of the programs. Finally as a feature developped by our team, we force the program behavior to change if the program goes through mutations.

Parameter Name	Value
maxNbActionsPerEval	**10000**
maxNbEvaluationPerPolicy	**500**
maxProgramSize	**20**
pAdd	0.5
pDelete	0.5
pMutate	1.0
pSwap	1.0
maxInitOutgoingEdges	**2**
maxOutgoingEdges	5
nbRoots	**500**
pEdgeAddition	0.7
pEdgeDeletion	0.7
pEdgeDestinationChange	0.1
pEdgeDestinationIsAction	0.5
pProgramMutation	0.2
nbGenerations	**100**
nbIterationsPerPolicyEvaluation	**10**
nbRegisters	8
ratioDeletedRoots	0.4
archive Size	**500**
archivingProbability	**0.01**
maxConstValue	100
minConstValue	-10
pConstantMutation	0.5
forceProgramBehaviorChangeOnMutation	**true**
nbProgramConstant	0
nbThreads	**16**

Table 5.2: Parameters of the TPG framework used in this Chapter.

5.5 Experimental Results

This section discusses the experimental results of the exploration as well as the mitigation of learning biases for the design of an AIDS.

5.5.1 Preliminary training of a TPG on the CICIDS 2017 dataset

As a preliminary work, and as the TPG method is used in each chapter of this thesis, a legacy TPG is trained on the CICIDS 2017 dataset. The TPG is trained using a classical Reinforcement Learning (RL)-liked fitness function where a correct classification results in a $+1$ score and a bad classification a -1 score. Figure 5.1 represents the learning curve of such a TPG. In particular, this basic IDS is bound to a TPR of 99.53% and a TNR of 75.46%. This IDS is rather good considering that it does not raise too many false positive alerts (0.47%). In operational conditions, it is required that there are as few false alerts as possible as alerts imply strong activity of cyber analysts to examine the connection and to apply counter measures to an eventual intrusion. In practice, a FPR of 0.47% represents more than 4600 false alerts per millions of connections. Supposing that the network records 1M connections per days, and that an analyst can study between ten and twenty alerts per day, 230 analysts would be required to analyze those 4600 false alerts. As a consequence, the FPR is a crucial parameter for AIDS. Hence, such an IDS is not precise enough to be used under operational conditions.

For a second part of this preliminary study, the F1-score fitness function, more suited to classification problems is used as a reward function for the TPG agent. F1-score is defined in 4.3. Figure 5.1 is the learning curve of this revised version of the TPG. In particular, this IDS is bound to a TPR of 77.55% and a False Positive Rate (FPR) of 1.72%. The F1-score function further balances the classification results of the IDS finding a better equilibrium between True Positive Rate (TPR) and True Negative Rate (TNR). This equilibrium is interesting for classification tasks but not relevant for intrusion detection where the minimization of the false positive alerts is a more important criterion.

The FPR reached by the legacy TPG are too high and not sufficient to draw conclusions on the TPG capabilities in an intrusion detection context. We study the impact of the dataset biases on the TPG-based AIDS.

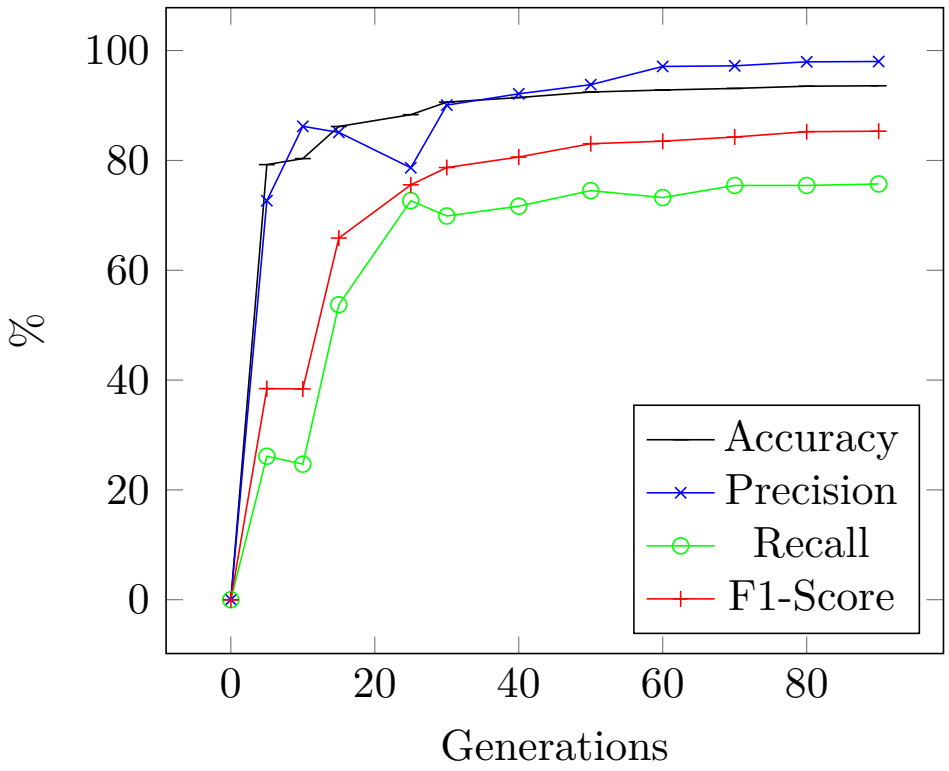


Figure 5.1: Machine learning evaluation metrics using the legacy TPG on the CICIDS 2017 dataset depending on the number of training generations.

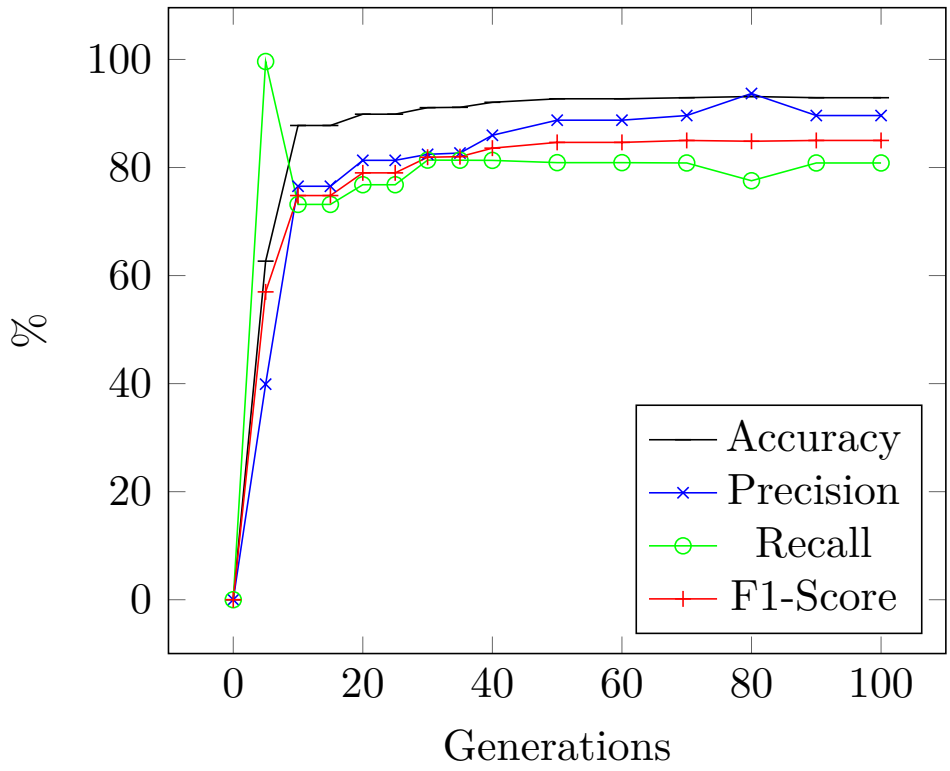


Figure 5.2: Machine learning evaluation metrics using a TPG using the F1-score fitness function as a reward depending on the number of training generations.

Model	source IP	source port	destination IP	timestamp
$M1$	1	0	0	5
$M3$	1	0	0	0
Information gain	0.8285	0.8420	0.7874	0.712606

Table 5.3: This table sums up the amount of observation of the supposedly biased features in the programs of the best TPG teams). The model $M1$ is trained and tested using those features. $M3$ is trained and tested with noisy features. High Information gain is present in the used and supposedly biased features.

5.5.2 Experiment 1: impact of the representation bias on the learned NIDS

As a first part of this study, the representation bias of the IDS is studied.

This bias exists in a dataset where the available features are not representative of the problem faced by the ML algorithm. As described in [SLG18], the CICIDS 2017 dataset has been synthesized using a sub-network of four computers to generate attack traffic. The source IP and ports and the source IP are thus consequent clues for the detection of attacks or the identification of normal traffic. Furthermore, as the intrusions are simulated at a given time, the timestamp feature of the dataset constitutes an important clue for the unfair detection of the intrusions.

Firstly we compute the entropy $E = 0.8433$ of the Benign and Attack traffic in the CICICDS 2017 dataset. This entropy bounds the information gain of the features. Information gain of each feature is between 0 and E. The information gain for a perfect split values E, meaning that the feature alone is able to separate the intrusions from the normal traffic of the dataset.

Table 5.3 sums up the observations of the models observing or not those features. From this table we can observe really information gain. These values, correlated with the knowledge on the data synthesis confirm the hypothesis of biased features in the CICIDS 2017 dataset. The model $M1$ is biased as it uses the Source IP and timestamp features. Table 5.4 sums up the results of the three classifiers. $M2$ is trained with the legacy CICIDS 2017 dataset, but infers on a dataset where white noise is synthetically added to these previous features. The inferring results raise of 18% FPR as the observed features are noisy, proving the use of the features in the decision making. In operational terms, a raise of FPR implies an over-solicitation of the analysts. Fewer correlations can be made between the identification features and the presence or not of an intrusion while inferring.

Model	TPR	TNR	Learning score
$M1$	99.8	77.6	0.87
$M2$	99.8	59.7	0.87
$M3$	96.3	83.2	0.88

Table 5.4: This table sums up the detection results of the three models built using (or not) the supposedly biased identification features. The learning score is obtained during training and the TPR and TNR are inference results. The learning score used comes from the legacy TPG as described in Section 5.5.1.

$M3$ results (view Table 5.4) differ from results of $M1$ and $M2$:

- Firstly the TNR is affected. For $M3$, the identification and temporal features are not available anymore, correlations using other features are made. The TNR of $M1$ is inferior to the one of $M3$ which uses features supposedly more relevant than the connection identification or temporal feature.

- Secondly, the TPR is slightly inferior for $M3$, proving that the Timestamp and identification information were helping the identification of intrusion. Table 5.3 confirms that the variables are not of interest when additional white noise is added.

The complexity of the different models (described by the number of teams, programs and depth of the graph) is similar for the three models. This is mainly due to the determinism of the used TPG framework.

It appears that the identification features of the logs constitutes, in the CICIDS 2017 dataset, a representation bias and should be removed for a fair exploration of the intrusion detection problem. We recommend not to use the features Source IP and Timestamp to reduce representation bias while training on the CICIDS 2017 dataset.

5.5.3 Experiment 2: impact of label bias on the ML-based AIDS

This section uses the results of the previous experiments. The features containing representation bias (Source IP, Source port, Destination IP and timestamp) are removed from the training datasets.

We here study the labeling bias of in the CICIDS 2017 NIDS dataset. In order to identify the label bias, two models are used:

- $M4$ is trained on the original version of the CICIDS 2017 dataset

Class	Detection rate of M1 (%)	Detection rate of M1' (%)	Detection rate of M2 (%)
Benign	99.53	99.67	99.98
Brute Force	0.00	0.00	0.00
XSS injection	0.00	x	x
SQL Injection	x	x	x
Botnet	0.00	0.00	0.00
Infiltration	0.00	x	x
FTP-Patator	0.00	0.00	100.00
SSH-Patator	0.00	0.00	0.00
Port Scan	99.13	0.11	99.71
DoS Slow Loris	0.00	16.67	0.00
DoS Slow HTTP test	52.99	3.70	0.00
DoS Hulk	62.64	13.31	99.97
DoS Golden-eye	25.00	0.58	94.12
Heart-bleed	x	x	x
DDoS	87.99	4.21	100.00

Table 5.5: Inferring results of the Model $M4$, trained on the original CICIDS 2017 dataset $M4'$ is the same model as $M4$ inferring on the revised version of the CICIDS 2017 dataset. Conversely, $M5$ is trained and infers on the revised version of the CICIDS 2017 dataset. The model $M4$ loses a significant part of its detection capabilities, comparing the per-class results with $M4'$. This can be explained by the re-labeling of the dataset. The detection rate drops on most intrusions. $M5$ reaches higher detection rates than $M4$. 'x' indicates that no samples from this class exist in the test dataset.

- $M5$ is trained on the corrected version of the CICIDS 2017 dataset, as described in [ERJ21].

Both models are used inferring on both datasets and the per class detection rate are studied. The revised version of the CICIDS 2017 dataset contains fewer intrusions than its original version as some of the intrusion traffic as been re-labeled as "Attack" as some of the intrusion traffic was interrupted. As we focus on the intrusion detection problem, no alerts should be raised by the NIDS when the "Attack" label is present. Thus, the "Attack" traffic is considered as sane.

The important information to note from Table 5.5 is that inferring with $M4$ does not raise the amount of false positive alerts. Even with the re-labeling, the IDS is able to detect some intrusions. The mislabeling of the CICIDS 2017 dataset has an impact on the learning. Furthermore, we can see that, in line with the results of [ERJ21], $M5$ trained

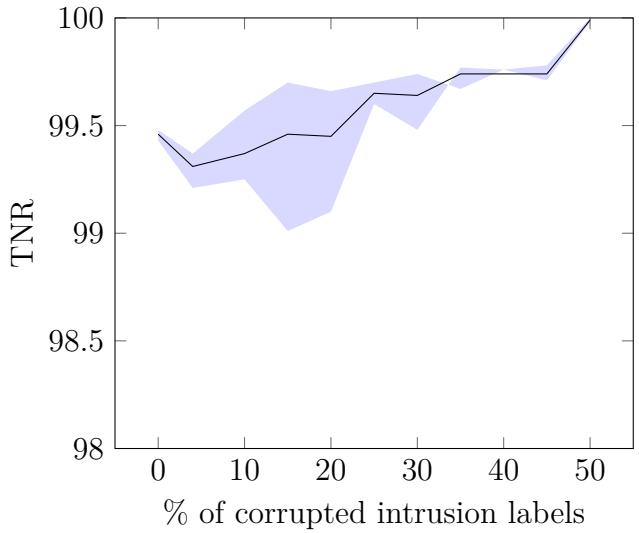


Figure 5.3: TNR of $M4$ with respect to the percentage of corrupted intrusion labels. The TNR does not vary when the amount of corrupted intrusion label evolves. It is because the corrupted labels correspond to intrusion traffic. When inferring, the intrusion traffic does not trigger an alert an is thus a False Negative, not impacting the FPR. When 50% of the intrusion labels are corrupted, the TNR eventually reaches 100%. The intrusion traffic is corrupted, and the traffic is qualified as sane all the time.

on the revised version of the CICIDS 2017 datasets converges easily and obtains better detection rates than $M4$.

5.5.4 Experiment 3: cost of mislabeling of the data

It is a complex task to correctly label a dataset. Under operational conditions, unknown traffic can be labeled as sane (at $t = t_0$) while it is in fact intrusion traffic. At $t = t_0 + \delta t$, the label of the incorrect samples are modified and the model must adapt. To simulate this issue, a fraction of the intrusions labels are corrupted, with a uniform probability, from the training dataset. Figures 5.3 and 5.4 displays respectively the TNR and TPR of $M4$ while inferring on the CICIDS 2017 testing dataset. As we can see on Figure 5.3, the TNR is slightly raising. It is due to the fact that the corrupted labels raise the probability of having negatives samples in the training set and thus affect the learning. Indeed, in such conditions, maximizing the number of true negatives helps the TPG to reach higher scores. Conversely, the true positive rate drop when more than 25% of the labels are corrupted.

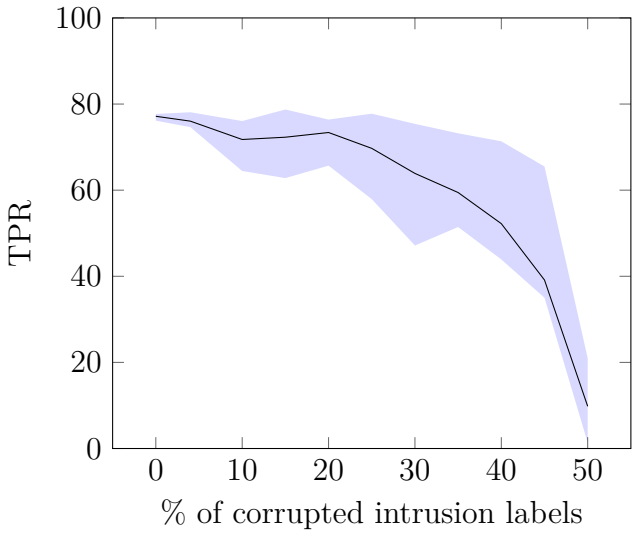


Figure 5.4: TPR of $M1$ with respect to the percentage of corrupted intrusion labels. The TPG model for intrusion detection is under 90% of its detection capabilities when reaching 25% of corrupted labels. Its detection capabilities drop drastically when the quantity of corrupted labels raises.

5.6 Discussion: mitigation of the representation and labeling biases of NIDS datasets

Although the mitigation of data biases require, at one point an expert knowledge, we can propose a procedure to reduce the representation biases of the datasets.

5.6.1 Assessing representation biases of a NIDS dataset

As seen as experimental results, reducing the biases in the dataset does not immediately lead to better classification results. Indeed, the biases facilitate the classification task by providing a simpler problem with inadequate features. This issue is known as a representation bias. We have identified features bound to a representation bias in the CICIDS 2017 dataset. The main drawback of the approach followed in this chapter is that this identification requires a deep knowledge of the data and its synthesis. Other features of the CICIDS 2017 dataset may be subject to representation bias.

In order to evaluate whether or not a feature is subject to a representation bias, we propose the following procedure :

1. Compute the information gain of all features

2. Sort the features on descending information gain order

3. From the first features (i.e. the feature having the greater information gain), study their specificity and whether or not their variations are realistic with respect to a real-world use-case.

4. Remove the risky features and verify that both TPR and FPR are not decreasing too much, even if this impact can be explained by a reduction of the model over-fitting on the biased features.

Such a procedure can not be followed on the labeling bias. Errors in labels require a costly expert analysis to be identified. In a real-life scenario, zero-days intrusions exist, implying that all attack traffic is not known and thus, considered as normal traffic.

5.6.2 Mitigation of labeling bias using TPG and future work

Expert analysis is required to reduce the label bias of a dataset by re-labeling the erroneously labeled samples. In order to mitigate the issue, we show in Section 5.5.4 that the use of the TPG is robust to up to 25% of missing attack labels. This robustness comes from the RL-based reward system. Through belated rewarding of a whole sample batch, the TPG lowers the supervision. We believe that lowering the supervision of the TPG could help in increasing the robustness of the method facing label bias. A semi-supervised prototype of the TPG relying on internal reward is described in Appendix A. The study of the intrusion detection capabilities of a semi-supervised version of the TPG and the impact measurement of data biases on it are kept as future work.

5.7 Conclusion

This chapter has explored both the representation and label biases existing in the CICIDS 2017 dataset. This chapter measures the impact of the learning biases and evaluates the resilience of the TPG framework to those data biases. For the first part, it is shown that some of the features of the CICIDS 2017 dataset contain a representation bias that is costly for the model. Indeed, the CICIDS 2017 dataset generation uses a cluster of attackers that are the source of all the intrusions at pre-set times. Thus, the information of the source IP and port, the destination IP as well as the timestamp give clues that do not allow to generalize if a given traffic constitutes a threat for the network or not.

Experimental results show that these features are used by the ML agent in order to decide whether or not an alert is to be raised. Furthermore, adding noise in those variables for inference leads to a drop of 18% of the TNR of the TPG. A procedure that consists in the correlation of expert knowledge about the data synthesis for high information gain features is proposed. This procedure, prior to a pre-processing of the dataset can be used to reduce the impact of representation bias on the model. Secondly, the impact of the label bias on the TPG is measured using both the original and the corrected labels of the dataset. Experimental results show that inferring on the corrected dataset having learned on the biased one made the results drop by up to 99% on the Ports Scans attacks and by a mean of 61% on the attack traffic. Finally, the resiliency of the TPG is tested on the original dataset where some of the attack traffic is synthetically biased. Results show that the TPG method is resilient to up to 25% of inaccurate labels.

CHAPTER 6

Contribution 2: Study of the impact of data imbalance on TPG performance

6.1 Introduction

Results of this study have been published in [Sou+22a] Imbalanced classification is the problem of training a classifier on a problem where one or several classes are represented by fewer samples than other classes. Imbalanced classification is necessary in a wide set of use cases, in particular when rare events [WRM03], or anomalies [CBK09], need to be detected in a large amount of *normal* data. The imbalance can reach various orders of magnitude making it particularly difficult to efficiently identify minority classes while avoiding false positives, i.e. false identification of the minority class.

While a lot of real world use cases are imbalanced, most recent studies on machine learning based classification focus on learning from balanced datasets. The studies of imbalanced learning problems themselves mostly manage the imbalance by over-sampling or under-sampling training and test datasets [YL06; ZL14].

Genetic Programming (GP) has recently been shown to perform well on balanced image classification [KH17b] by using the framework of Tangled Program Graph (TPG). With respect to competitors, TPGs have the advantage of a lightweight inference and a support for both high data cardinality and efficient diversity maintenance [SAH21].

When used for classification, TPG actions translate into classes while they originally refer to environment modifying actions in a Reinforcement Learning (RL) environment. Further details on the TPG framework are available in Section .

This chapter aims at adapting the TPG GP framework so as to improve its capacity to perform imbalanced classification. We first show that the current TPG framework does not perform well in an imbalanced context. We then propose new semantics to be integrated within TPGs in the form of an adapted selection system.

In this chapter:

- we evaluate the decrease of classification performance induced by the dataset imbalance in a TPG genetic programming framework.
- we compare different fitness functions and evaluation metrics; and we evaluate their effect on the training of a TPG-based classification.
- we demonstrate on an example that GP can adapt to imbalanced classification problems, provided that the fitness and selection phase is adapted using statistics on training sub-set performances.
- we report experimental studies conducted both on the MNIST handwritten digit image classification problem and on a network intrusion detection problem.

6.2 Related work

State of the art methods to deal with imbalanced learning problems can be classified into:

- *data level methods*, using dataset over and under-sampling or specialized feature selection, and
- *algorithm level methods*, including e.g. cost sensitive methods and hybrid/ensemble learning [Lee+18].

Data level methods aim to modify the input data to mitigate the classification performance degradation due to imbalanced datasets whereas algorithm level methods aims to modify the classifier in order to make it robust to the imbalanced data. Ensemble methods are a combination of both data level and algorithm level mitigation. The methods proposed in this chapter concentrate on algorithmic-level imbalance mitigation.

Over-sampling [ZL14] and under-sampling [YL06] methods re-balance the dataset to make an unmodified classifier robust to imbalance. Over-sampling methods select or generate more samples from the minority classes than from the other classes. This method adjusts the amount of data available in the training dataset and introduces a controlled bias. Conversely, under-sampling methods consist in selecting fewer samples from the majority classes, voluntarily omitting data. Synthetic Minority Oversampling Technique (SMOTE) [Cha+02; Fer+18], used for imbalanced classification, is a combination of over-sampling and under-sampling. The main drawbacks of these methods are that in case of over-sampling, artificially duplicated samples risk causing over-fitting of the classifier. In case of under-sampling, important components from the majority class can be deleted, and it is a shame to ignore samples potentially costly to obtain.

At the algorithm level, the mitigation of the imbalance issue is mostly addressed through adapting the fitness function. In [PBC15; Pei+20; Wan+15; Lóp+13], authors use custom or cost-based functions to counteract the effects of data imbalance. In [Vie+18], a fitness function is derived from the F1-Score and adapted to the problem of analyzing textual and biological data. The main disadvantage of the method is that the cost of error of a misclassification is not initially known by the algorithm but estimated through the learning. In [DH08], the Area Under Curve (AUC) metric is coupled with an active sub-sampling, to improve the results of the GP binary classification.

In these previous works, no comparison is performed on the performance of the training using one fitness function or another. This observation motivates the current study on the impact of imbalance on genetic programming-based classification. There exist generic fitness functions [JCD13] that are relevant candidates for learning from imbalanced data:

- Matthew's Correlation Coefficient (MCC) [Zhu20] is a correlation coefficient used for binary classification problems or for the detection of positives samples among negative samples. Yet, the study recommends not using MCC as an evaluation metric but only as a fitness function.

- G-mean [Aur+19] is a geometric mean of sensitivity and specificity. It is more reliable than F1-score to evaluate the quality of a training on an imbalanced dataset because the true negatives are injected in its computation while they do not appear in F1-score.

- Cohen's Kappa (κ) [VKS10] represents the probability of the correct classification not being random. It is interesting for high imbalance rates because the computation of the random classification is influenced with the imbalance ratio.

Section 6.4.2 will discuss these functions and other more common Machine Learning (ML) metrics for their use as fitness functions.

Classification with GP is an active research field [SAH21; Sá+20; LBX20]. In [SAH21], a TPG agent is specialized for performing balanced classification through the comparison of learning schemes. Among them, the "Lexicase", a method where each class is introduced one after the other as the agent's input, appears to be the most efficient. As this study aims at improving the imbalanced classification task results in order to contribute to the intrusion detection domain, we work under the constraint that we should not alter the incoming data to support a flow of network data as they come. These modifications of the Learning Agent of the TPG motivate the specifications of the Learning Agent to suit the imbalanced learning classification task.

GP includes a selection phase where the best individuals of the generation are kept for mutations and crossovers. There exist several selection algorithms such as:

1. **Roulette wheel** selection: The individuals with the highest fitness has higher probability to be kept.

2. **Rank** selection: The individuals are ranked and attributed a probability of being kept depending on how well they performed. The selection is a Roulette wheel selection with non-uniform probability.

3. **Steady State** Selection: Best individuals go through crossovers and mutations. The children replace the worst performing individuals.

4. **Tournament Selection**: Some individuals from the pool are randomly selected and evaluated in a tournament. The individual that performs the best is kept for mutation and crossover, the others are killed. The process iterates on a pool of individuals.

Although the impact of selection method for imbalanced classification has not been shown at algorithmic level, those methods have been used to improve the performance of the sampling method for hybrid classifiers. Roulette Wheel selection (1) has been used for imbalanced classification sampling methods in [MNN21; Gon+19]. [Le+21; Poz+21;

LP18] are hybrid learners implementing steady state selection to the data-level mitigation method. Similarly, [PBC15] uses tournament selection to improve the sampling process.

The legacy TPG implements a rank selection (2) where the best individuals go through crossovers and mutations. The children replace the worst performing individuals. [SAH21] demonstrates that rank selection is an adequate candidate for balanced classification.

6.3 Impact of imbalance on the learning

The imbalance nature of the classification problem has two different forms of impact. The learning process requires the use of a fitness function in order to evaluate the performances of the built model. When the number of samples from the positive class tends toward zero, we observe that the range of the fitness function is reduced, making it difficult, during the learning process, to distinguish which model fits the data in a better way.

The imbalance of the considered problem is expressed as $card(s^P) \ : \ card(s^N)$. In order to compactly express this degree of imbalance that can reach very high values in practice, we propose the notion of Imbalance Order of Magnitude (IOM). Formally, we define iom of a sample set s as:

$$\begin{aligned} &iom : s \to \mathbb{N}, \\ &iom(s) \mapsto log_{10}\left(\frac{card(s^N)}{card(s^P)}\right) \end{aligned} \tag{6.1}$$

For example, an IOM of 3 in a training set s_{tr} means that there are 1000 times more negative samples than positive samples in the set. The same imbalance is expressed as a ratio $1:1000$, where a ratio is defined as $1 : \frac{card(s^N)}{card(s^P)}$. In this chapter, we hypothesize that the IOM of both the training and test sets are equivalent: $iom = iom(s_{tr}) \approx iom(s_{te})$ and we thus refer to this IOM as the global imbalance of the problem.

Furthermore, the scarcity of the data makes it difficult for the learning agent to produce a realistic model of the minority class. Both issues are discussed in this section.

6.3.1 Cardinality: range of the fitness function

The range of values of the fitness function is affected by the number of positive samples in the dataset tends toward zero. In order to illustrate the problem, Example 6.3.1 depicts several situations at different IOM.

Example 6.3.1. Supposing two classifiers having constant True Negative Rate (TNR) and False Positive Rate (FPR), learning at different IOM and analyzing a constant number of 1,000 samples.

- The first classifier (C_1) has a 90% TNR and 10% FPR.
- The second classifier (C_2) has a 99% TNR and 1% FPR.

We study the evolution of the Fitness function for different True Positive Rate (TPR).

Considering the κ fitness function at different IOM, we observe (see Figure 6.1):

- For high IOM, the used fitness function tend to be less influenced by the detection of a positive sample. In the example, the κ fitness function is used. Figure 6.1 shows that even though the classifier detects accurately all the positives, its κ stays relatively low (around 0.4)

The genetic process is affected by this issue. When the cardinality of the positive samples is low, the probability of having a positive sample in the training set is low. Hence, the GP algorithm maximizes its result by identifying correctly the negative samples. The seemingly good results obtained by the GP algorithm can cause the loss of important information by deleting important information of the Positive sample model.

6.3.2 Imbalance: modeling of the data

The TPG builds a model from trial and errors of the different classes existing in the dataset. The training phase of the learning is based on the observation and classification of samples from a training set (also called a batch) and its classification. Once a batch is processed, the overall fitness is computed and the teams are sorted from the most performing to the worst. The teams that performed the best are kept. As a result, the model of the observed samples becomes better. When the dataset is imbalanced and independently of its fitness function, much effort is put on the majority class. The model of the majority class is thus quite accurate while the model of the minority class is poor. In the worst cases, all samples are classified to belong to the majority class.

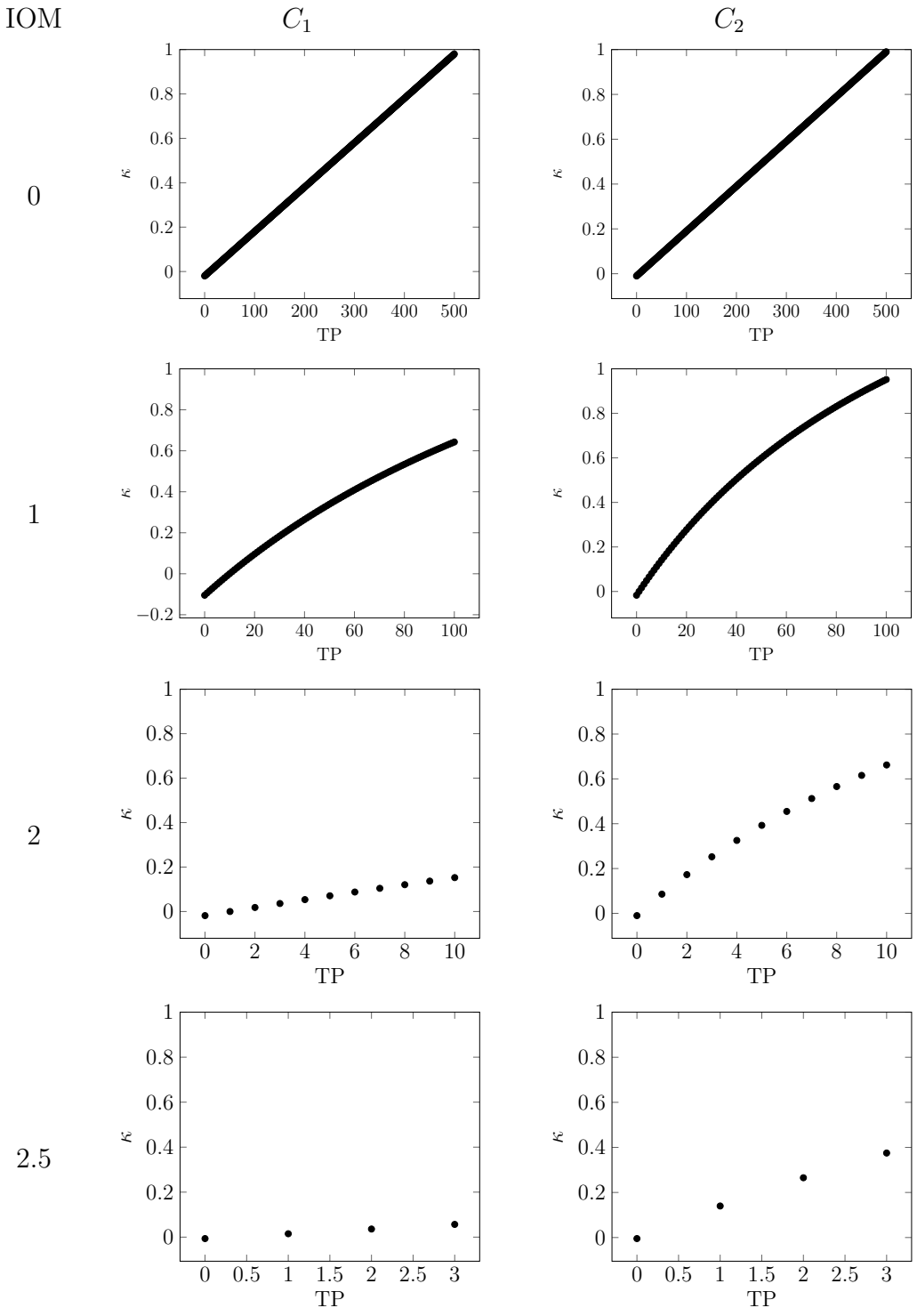


Table 6.1: The range of the fitness is highly affected for high IOM. The sensibility of the fitness function is high with respect to the TNR and low with respect to the TPR. It causes an issue when the probability of having a positive sample in the training set is low.

6.4 Fitness functions and genetic selection phase for imbalance classification

This section aims to define the imbalance classification problem and to specify the hypothesis and intuitions for this work. In particular, Section 6.4.2 details the fitness and evaluation functions compared in this work and Section 6.4.3 explains the proposition of the selection phase to ensure the robustness of the learning GP algorithms.

6.4.1 The imbalanced classification problem

We consider the following problem: a classifier m is trained on a training set s_{tr} and tested on a test set s_{te}. Each sample $\sigma \in s_{tr} \cup s_{te}$ belongs to one of two classes $\{P, N\}$ where P denotes the positive class and N the negative class. The oracle function $o : s_{tr} \cup s_{te} \rightarrow \{P, N\}$ associates to each sample σ its true class $o(\sigma)$ while the classifier $m : s_{tr} \cup s_{te} \rightarrow \{P, N\}$ associates to each sample σ its predicted class $m(\sigma)$.

Formally, given a set of samples s, the subset of true positive samples is expressed as $s^P = \{\sigma \in s | o(\sigma) = P\}$. Conversely, the subset of true negative samples is expressed as $s^N = \{\sigma \in s | o(\sigma) = N\}$.

In the rest of this chapter, we will use TP, TN, FP and FN as intuitive cardinality expressions of respectively true positives, true negatives, false positives and false negatives in the classifier produced results, on the test set. We formally define these expressions as:

$$\begin{aligned} TP &= card(\{\sigma \in s_{te} | o(\sigma) = P \wedge m(\sigma) = P\}), \\ TN &= card(\{\sigma \in s_{te} | o(\sigma) = N \wedge m(\sigma) = N\}), \\ FP &= card(\{\sigma \in s_{te} | o(\sigma) = N \wedge m(\sigma) = P\}), \\ FN &= card(\{\sigma \in s_{te} | o(\sigma) = P \wedge m(\sigma) = N\}). \end{aligned} \tag{6.2}$$

A classification problem is qualified as imbalanced when the training and test sets contain much less positive samples than negative samples. The choice of setting the positive class as the least represented one is motivated by the detection problem that consists in detecting a rare type of positive events among a vast number of (negative) samples. If data is received as a sample, imbalance corresponds to a low frequency of appearance of samples of one class, making each of them of a high importance.

Intuitively, the imbalance of a problem systematically tends to complicate the task of the classifier because an ML algorithm has difficulty to fit a model with few samples of the minority class. In addition, an ML algorithm favors the accuracy on the majority class to achieve an overall high accuracy and thus neglects the minority class. We want to study this hypothesis with a GP classifier, and characterize the link between imbalance ratios and classification decrease of performances.

Secondly, many metrics exist to evaluate the quality of a classifier and these metrics themselves are affected by the degree of imbalance of the problem. Our objective is to provide a multifaceted characterization of the classification decrease of performances.

Finally, we aim at adapting the considered GP to our imbalanced problem classifier by adapting its fitness and selection phase.

6.4.2 Selection: Choosing fitness and evaluation metrics

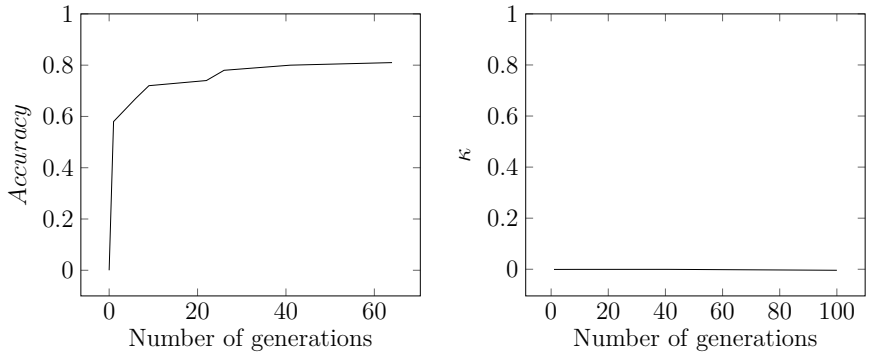


Figure 6.1: Using accuracy classification metric on an imbalanced problem. With an IOM of 1.5 and using accuracy as TPG fitness function, the TPG agent is able to maximize its accuracy (plot (a)) but produces a bad classifier. Indeed, its κ remains null (b) because the TPG agent is predicting the majority class for all samples.

Two functions need to be selected to train and test the TPG: the fitness function and the evaluation function. While the fitness function generates the inputs to the comparison of two teams, the evaluation function evaluates the capacity of the model to perform the classification. For both fitness and evaluation, functions from the state-of-the-art can be used, such as classification accuracy and F1-score. We here discuss the different metrics that will be evaluated in the experimental results.

There exist many methods to evaluate the efficiency of a model, the method to be used for a given classifier depending on its objective and on the requested trade-off between missing true positive samples and detecting false positive samples.

On balanced classification problems, i.e. $iom = 0$, a classifier efficiency is primarily evaluated through its accuracy, precision, recall and F1-score. On imbalanced problems, these statistical metrics shadow the minority class misprediction and profit to the majority class [War+19]. Consequently, some fitness functions lose their utility on imbalanced classification problems. In this section, we compare, for different imbalance ratios, a list of classification metrics. These metrics are then used to both evaluate the classifier and to build a fitness function for the training of the classifier.

Accuracy, used in most ML publications today, is the arithmetic mean of the correct decisions, defined as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.3}$$

The accuracy metric is not an adequate method for evaluating learning-based methods in highly imbalanced contexts. Figure 6.1 illustrates this metric on the imbalanced MNIST dataset. It depicts the evolution of the accuracy and the κ score (see below) with respect to the successive generation of a GP algorithm. Despite the constant increase of the accuracy, the κ score remains steadily null.

$$F1\text{-}score = \frac{TP}{TP + \frac{1}{2} * (FP + FN)} \tag{6.4}$$

Similarly, the F1-score does not take into account TN. F1-score is arguably more adapted to imbalanced classification than Accuracy but a model mistaking half of the time on the prediction of the minority class results in a seemingly fair F1-score of 0.667.

The G-mean (a geometric mean of the product of the sensitivity and the specificity of the agent) is more adapted to imbalance cases. It is the geometric mean of the two per class accuracy:

$$G\text{-}mean = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}} \tag{6.5}$$

The value of G-mean is equally sensitive to the ratio of TP versus FN and to the ratio of TN versus FP.

MCC (also known as the phi-coefficient) is the computation of the correlation between true and predicted values, leading to a maximal correlation coefficient of one in the case

of a perfect classifier. MCC is interesting in the case of imbalanced classification because its formula is completely symmetric. Similarly to the G-mean, high MCCs can be reached only for good predictions of both majority and minority classes.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{6.6}$$

The computation of the MCC depends on a product of sums as a denominator. When the IOM is high, there is a probability that both TP and FP are zero, leading to an improper operation.

Finally, the Cohen's Kappa score, measures the reliability of a decision by taking into account the probability that a good decision happened by chance.

$$\kappa = \frac{2 \times (TN \times TP - FN \times FP)}{(TP + FP)(FP + TN) + (TP + FN)(FN + TN)} \tag{6.7}$$

Similarly to MCC, the computation of the κ score is sensitive to the number of available positives in the dataset. The *kappa* score will unavoidably tend toward zero when the number of positives tends toward zero, unless in the specific case of a perfect classifier.

In addition to the use of different evaluation functions, it is also possible to modify the fitness function of the classification policy when it is suitable. In [SAH21], the inner fitness system for the training of the classification policy is $+1$ for a good classification and -1 when the policy is mistaking. To mitigate the imbalance of the dataset, this fitness system can be weighted e.g. using balanced accuracy. The principle of *balanced accuracy* is to weight the accuracy fitness function with coefficients inversely proportional to the ratio of minority samples. In an imbalanced context, the fitness function and the evaluation metric must be seen as functions to maximize and do not tell the whole truth on the accuracy of the classifier. Indeed, each learning problem is specific and has specific requirements in terms of FN or FP. The metrics discussed here-over are assessed as both fitness and evaluation functions in Section 6.6.

6.4.3 Selection: Comparison of the individuals

When learning from an imbalanced dataset, few samples of the minority class are contained in the training batch. The risk for GP is that teams with good performance (according to the fitness function) on the majority class take the lead on teams predicting

accurately the minority class. The teams able to detect the minority class are likely to be deleted at the end of the generation step causing a loss of knowledge.

To mitigate this issue, we propose, inspired from [Bon+18], to modify the inner selection phase of the TPG to preserve the teams that both perform well, and have a low variance on different subsets of samples. Our proposition is to change the evaluation process of a team. Instead of evaluating the team on a whole training set, we divide the training set in N independent subsets and test each of them separately. Instead of having a global score for the whole set, we obtain N scores, one for each subset. We can then build confidence intervals for each team and compute a mean and a standard deviation of the agent's performance. Note that at this point, the mean of the confidence interval is identical to the legacy TPG and corresponds to the score of the agent on the whole test set. A comparison of the means of different teams would not change the selection.

Our hypothesis is that when data is imbalanced and subsets are chosen to be small, some subsets will contain very few positive samples. The teams trained on these subsets will thus under-perform on the minority class. When a sample of the minority class is present in the subset, these teams will reach a score significantly lower than the mean. Our proposition is to change the selection phase so as to maximize the lowest bounds of the confidence intervals of each team. Selecting individuals this way, the variance of classification quality is likely to be reduced, and the classifier should be more robust to imbalance.

The comparison of the bounds of the confidence intervals can lead to diverse results. Similarly, comparing individuals confidence intervals (wide or tight) can influence on the selection of the best candidate. In order to evaluate the proposal, experiments are performed by selecting individuals based on the optimization of a composite function using the mean and the width of the interval (γ). The problem is formalized hereunder, and experimental results are discussed in Section 6.6.2.

The selection of the best teams is influenced by the position of the lowest bound of the confidence interval and is weighted by a coefficient inversely proportional to the width of the confidence interval.

To formalize the problem, the test set S is divided into N subsets $\{S_0, S_1..., S_N\}$. The TPG is evaluated on each subset with a fixed evaluation function, generating evaluations $E = \{e_0, e_1, ..., e_N\}$. μ_e and $\tilde{S}$ are respectively E empirical mean and standard deviation.

Equation 6.8 computes the empirical mean μ_e while Equation 6.9 computes the empirical standard deviation $\tilde{S}$.

$$\bar{\mu_e} = \frac{1}{N}\sum_{i=0}^{N} e_i \tag{6.8}$$

$$\tilde{S}^2 = \frac{1}{N}\sum_{i=0}^{N}(e_i - \mu_e)^2 \tag{6.9}$$

According to the Central Limit Theorem, provided that subsets $\{S_0, S_1..., S_{29}\}$ are drawn from the same probability set, and are independent and identically distributed, Equation 6.10 is verified if the number of samples N_{μ_e} is higher than 30. Consequently, no assumption has to be made on the distribution of the population. In Equation 6.10, $\mathcal{N}(0, \sigma)$ represents a Gaussian distribution whose mean is 0 and standard deviation is σ .

$$\sqrt{N_{\mu_e}}(\bar{\mu_e} - \mu_e) \xrightarrow{Law} \mathcal{N}(0, \sigma) \tag{6.10}$$

The confidence interval $IC^p_{\mu_e}$ is computed in Equation 6.11 and contain μ_e with a probability p. The term $a^\alpha_{\mu_e}$ embodies the accuracy of the estimation and is computed as in Equation 6.12. z_α is given by the table of the standard normal distribution given p. N_{μ_e} is the minimal number of samples to simulate to get an estimate respecting the user-defined parameter $\{p\}$.

$$IC^p_{\mu_e} = [\bar{\mu_e} - a^\alpha_{\mu_e}; \bar{\mu_e} + a^\alpha_{\mu_e}] \tag{6.11}$$

$$a^\alpha_{\mu_e} = z_\alpha \cdot \frac{\tilde{S}}{\sqrt{N_{\mu_e} - 1}} \tag{6.12}$$

Considering a set of two teams $\{T_0, T_1\}$, we define the comparison of the teams score as follows:

$$T_0 > T_1 \Leftrightarrow (\bar{\mu_{e_{T_0}}} - a^\alpha_{\mu_{e_{T_0}}}) + \frac{1}{2a^\alpha_{\mu_{e_{T_0}}} + 1} > (\bar{\mu_{e_{T_1}}} - a^\alpha_{\mu_{e_{T_1}}}) + \frac{1}{2a^\alpha_{\mu_{e_{T_1}}} + 1} \tag{6.13}$$

Equation 6.13 results from the optimization of two elements:

- The lowest bound of the confidence interval $(\bar{\mu_{e_T}} - a^\alpha_{\mu_{e_T}})$
- The width of the confidence interval $(a^\alpha_{\mu_{e_T}})$, where the smallest confidence interval is to prefer $(\frac{1}{2a^\alpha_{\mu_{e_T}}+1})$.

By selecting in priority the teams generating tight confidence intervals, we expect to increase the robustness of our model. Selecting the lowest bound is believed to help preserving the individuals that observe positives samples.

6.5 Experimental Setup

To study the effect of imbalance and test the selection proposition, we use the MNIST dataset [Den12]. MNIST is a balanced collection of 50k training images and 10k test black-and-white images of handwritten digits (from zero to nine). In order to use the dataset for an imbalanced training, we adjust it as follows:

- we select a positive class among the 10 available classes
- we select the imbalance ratio
- we build a training set of N images (N vary between 10 000 and 45 000) by selecting images from all the negative classes and images from the Positive class respecting the imbalance ratio
- we build, using the same method, a test set of 2 000 images.

We use a sampling method not to counter the imbalance effect, but to put ourselves in conditions to deal with imbalanced data. The imbalance ratio is tuned from (1:1) (i.e. an IOM of 0), to (1:10000) (i.e. an IOM of 4) with steps of 0.5 IOM. In order to see if the agent is able to detect the minority class for high imbalance ratios, a Positive sample is forced into the test set when the imbalance ratio is over 1:1000 (IOM of 3).

6.5.1 Parameters of the TPG

Table 6.2 displays the parameters used in this Chapter to train TPGs. The parameter *maxNbActionsPerEval* vary between 10 000 and 45 000 depending on the number of available samples. The number of available sample is not fixed in order to be able to create training sets with the required imbalance while conserving as much as samples from the minority class as possible. The parameters of the TPG as defined in [Kel18] are used. The number of actions taken, the number of root teams and the number of evaluation required are designed to widen the exploration space of the TPG. Due to the consequences of these parameters, the size of the programs have been reduced to reduce the execution

Parameter Name	Value
maxNbActionsPerEval	Vary between **1000 and 45000** depending on the imbalance ratio
maxNbEvaluationPerPolicy	**2000**
maxProgramSize	**20**
pAdd	0.5
pDelete	0.5
pMutate	1.0
pSwap	1.0
maxInitOutgoingEdges	**2**
maxOutgoingEdges	5
nbRoots	**500**
pEdgeAddition	0.7
pEdgeDeletion	0.7
pEdgeDestinationChange	0.1
pEdgeDestinationIsAction	0.5
pProgramMutation	0.2
nbGenerations	**180**
nbIterationsPerPolicyEvaluation	5
nbRegisters	8
ratioDeletedRoots	**0.9**
archive Size	**500**
archivingProbability	**0.01**
maxConstValue	10
minConstValue	-100
pConstantMutation	0.5
forceProgramBehaviorChangeOnMutation	**true**
nbProgramConstant	0
nbThreads	16

Table 6.2: Parameters of the TPG framework used in this Chapter.

time of the programs. Finally as a feature developped by our team, we force the program behavior to change if the program goes through mutations.

6.6 Experimental Results

This chapter primarily aims to demonstrate that modifications on a GP framework can contribute to improve imbalanced classification. Firstly, we run a legacy TPG agent on a balanced classification task and study its convergence. Figure 6.2 represents the training for each class seen as the positive class. This figure is the result of balanced training with a legacy TPG agent. This plot shows that:

- TPG converges for each class of the dataset.
- A mean convergence of $\kappa = 0.61$ is reached.
- A wide range of κ are reached, suggesting that all the considered problems have a different complexity. The best results are obtained for the detection of the number “1", reaching $\kappa_1 = 0.85$. The worst results are obtained for the detection of the number “5", reaching $\kappa_5 = 0.4$
- Convergence is reached in most cases after 180 generations. We fix the number number of generations to this value in next experiments.

This section details the experimental results on the imbalance mitigation using the TPG.

6.6.1 Fitness function and evaluation metric

In this subsection, we investigate which fitness function and evaluation metric to use for training on an imbalanced dataset. The fitness function is used as an inner mechanism for selection of the best individuals in the pool. The evaluation metric is the criterion with which we evaluate a root team outside the learning environment. This evaluation is performed on the best root team of a generation. It enables a saving of the model if the evaluation is greater than the evaluation previous generations. When inferring, the best root team of the generation that maximized the evaluation is restored.

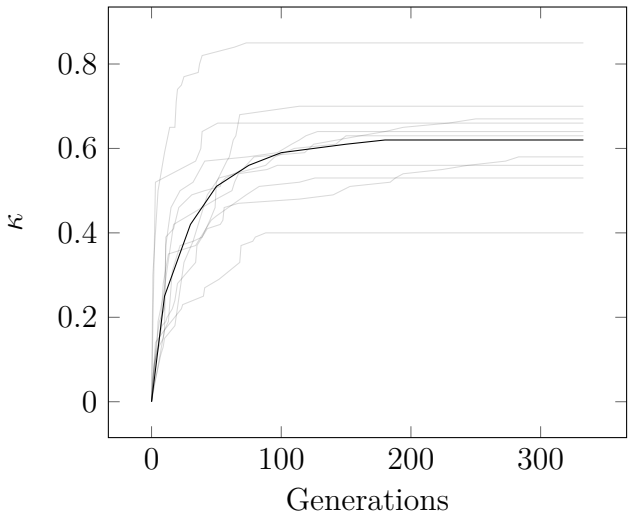


Figure 6.2: Learning a TPG on the MNIST dataset with balanced class ratio (IOM of 0). Grey curves show performance evolution with one digit class as the minority sample. The bold curve shows the mean and converges toward $\kappa = 0.61$.

6.6.1.1 Evaluation metric

A good evaluation metric for imbalanced classification is an evaluation metric that:

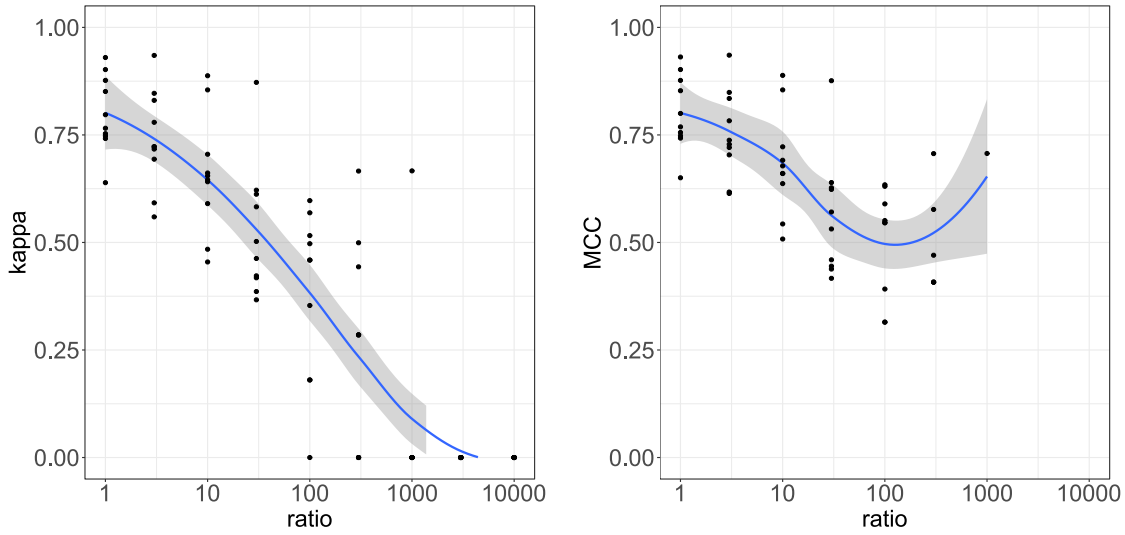
(i) Reaches a maximum for the considered best model,

(ii) Reaches a minimum for the considered worst model,

(iii) Takes into account the imbalance of the dataset,

(iv) Is humanly interpretable.

As explained in Section 6.4.2, F1-score, State-of-the-art TPG fitness and balanced accuracy are poor evaluation metrics on an imbalanced problem. Furthermore, it has been shown that G-mean is not a good evaluation metric as the influence of the Positive class is shadowed when the number of positive samples is low. Figure 6.3 compares the evaluation results of the same models with the two remaining considered metrics: κ and MCC. One expects for the mean evaluation of the classification quality to decrease when the IOM increases due to the cardinality issue described in Section 6.3.1. Curve on the right of the Figure represents the evolution of the MCC stops decreasing at an IOM of 3 as the other computed points are irrelevant. Indeed as seen in Section 6.4.2, the MCC computation can lead to an improper operation (division by zero) if two values among $\{TP, TN, FP,$



(a) (b)

Figure 6.3: κ (a) and MCC (b) versus imbalance ratio of a model. A point represents an evaluation for a fixed minority class of the dataset, for a fixed imbalanced ratio. The blue line and gray area represent the conditional mean of the evaluation results associated with confidence intervals. Each plot goes from IOM 0 to IOM 4. MCC is shown to badly capture the classification decrease of performances due to imbalance.

$FN\}$ are null. The adequate evaluation metric for imbalanced problems among the standard metrics is thus the Cohen's Kappa function. Each of these functions merges several parameters into a single value and, as so, only give a partial view of the problem. The class-wise accuracy or confusion matrix are still more complete indicators but can not be used easily in order to compare classification algorithms.

6.6.1.2 Choosing the right fitness

A right fitness function shall:

(i) Reach a maximum for the best model's predictions,

(ii) Reach a minimum for the worst model's predictions,

(iii) Produce a model that maximizes the aforementioned evaluation metric.

For the same reasons evoked on evaluation, F1-score and State-of-the-art TPG fitness are poor evaluation metrics for high IOM. Figure 6.4on page 110 shows the influence of κ, MCC and G-mean used as fitness functions. For low IOM, from zero to one, the

G-mean fitness function is the most performing. The MCC Fitness function is to prefer when the IOM is above one. This figure also shows that a TPG can train, with degraded performance, on problems with an IOMs greater than 4.

6.6.2 Evaluation of the Proposed Selection Algorithm

Figure 6.5 compares the method with the legacy TPG that selects individuals based on their means. The fitness function is G-mean for low imbalance and MCC for high imbalance. We can see that the result's variance is significantly reduced and that for IOMs from zero to 1.5, higher κ are reached. However, due to the evaluation metric and fitness function reaction to the vanishing presence of Positives in the dataset, the modified TPG fails to converge on IOMs over 3. Reaching low variance Kappas at very high IOMs will be our next objective.

Finally, Table 6.3 on page 111 shows the impact of the sub-batches size with respect to the imbalance ratio. The sub-batches are even more impacted by the cardinality issue described in Section 6.3.1. To this extent, it is normal that this selection mechanism does not improve the results in case of a low cardinality.

6.6.3 Testing on a Network Intrusion Detection Systems (NIDS) dataset

The CICIDS-2017 dataset [SLG18] used for testing the method on the intrusion detection problem. These results are to be compared with the Figures 5.1 and 5.2, on pages 79 and 80. The TPR of this method is at most 93.94% and the TNR, 94.13%. The model is more balanced using the G-score as a fitness function and thus, the TNR and TPR of the method are higher than in Figures 5.1 and 5.2.

Table 6.4 compares the results of the models described in Figures Figures 5.1, 5.2 and 6.6 after 100 training generations in terms of Accuracy, Precision, Recall and F1-score. The model optimized to deal with imbalanced performs better in terms of Accuracy, Recall and F1-Score, losing precision in order to give more balanced and higher overall results than the other two methods. The method produced a better classification model, although it is not the best model to perform intrusion detection as the lower precision implies a higher amount of false positive alerts.

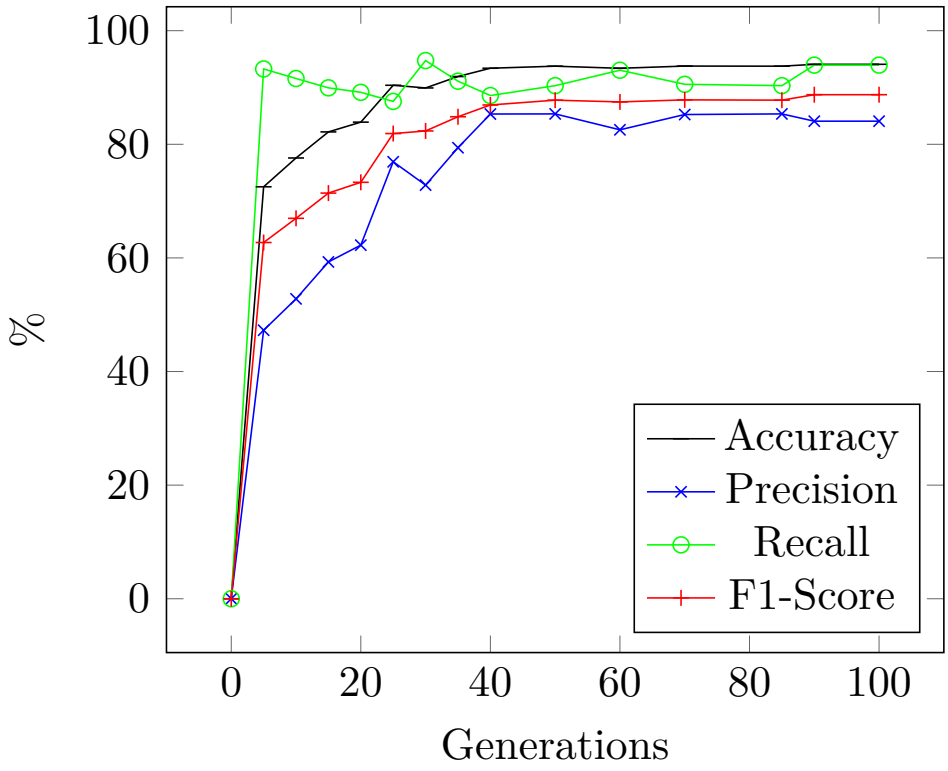


Figure 6.6: Machine learning evaluation metrics using the imbalanced learning adaptations for the TPG on the CICIDS 2017 dataset depending on the number of training generations.

6.7 Discussion and future work

The proposed modification of the TPG helps in increasing its resilience to imbalance data. Although we are able to learn from imbalanced data, the issue of the quantization and attenuation of the fitness function is not resolved. Actual fitness are functions of the TP, FP, TN and FN. The use of other variables should be explored to reduce the impact of imbalanced data. Furthermore, this study focuses on algorithmic modifications. This is motivated by the necessity of applying stream processing for operational IP network intrusion detection. Conversely, imbalanced problems are often simplified using hybrid methods containing both data processing and algorithmic mitigation. As a future work, finding methods for stream sub-sampling could lead to an hybrid mitigation of the imbalance in live operational NIDS.

6.8 Conclusion

This chapter has studied the effect of imbalance on a binary classification task using genetic programming. We demonstrate that the Cohen's Kappa is, among standard metrics, the one to use to evaluate a classifier on imbalanced problems. We also propose algorithm modifications on fitness and selection phases to support imbalance with genetic programming. Indeed, this problem is bound to a training phase where selection and fitness functions are key components, as the genetic programming algorithm uses selection phases to keep the individuals that perform correctly, based on their respective fitness. This chapter shows that the G-mean fitness function is a good candidate for low IOM while MCC fits better the highly imbalanced problems. In practice, it is shown that learning in an environment with an IOM of 4 is possible but with very degraded performances. Finally, we show that the robustness of GP agents facing moderate IOM problems can be increased through the use of the proposed selection phase, at the cost of lower performance on higher IOMs.

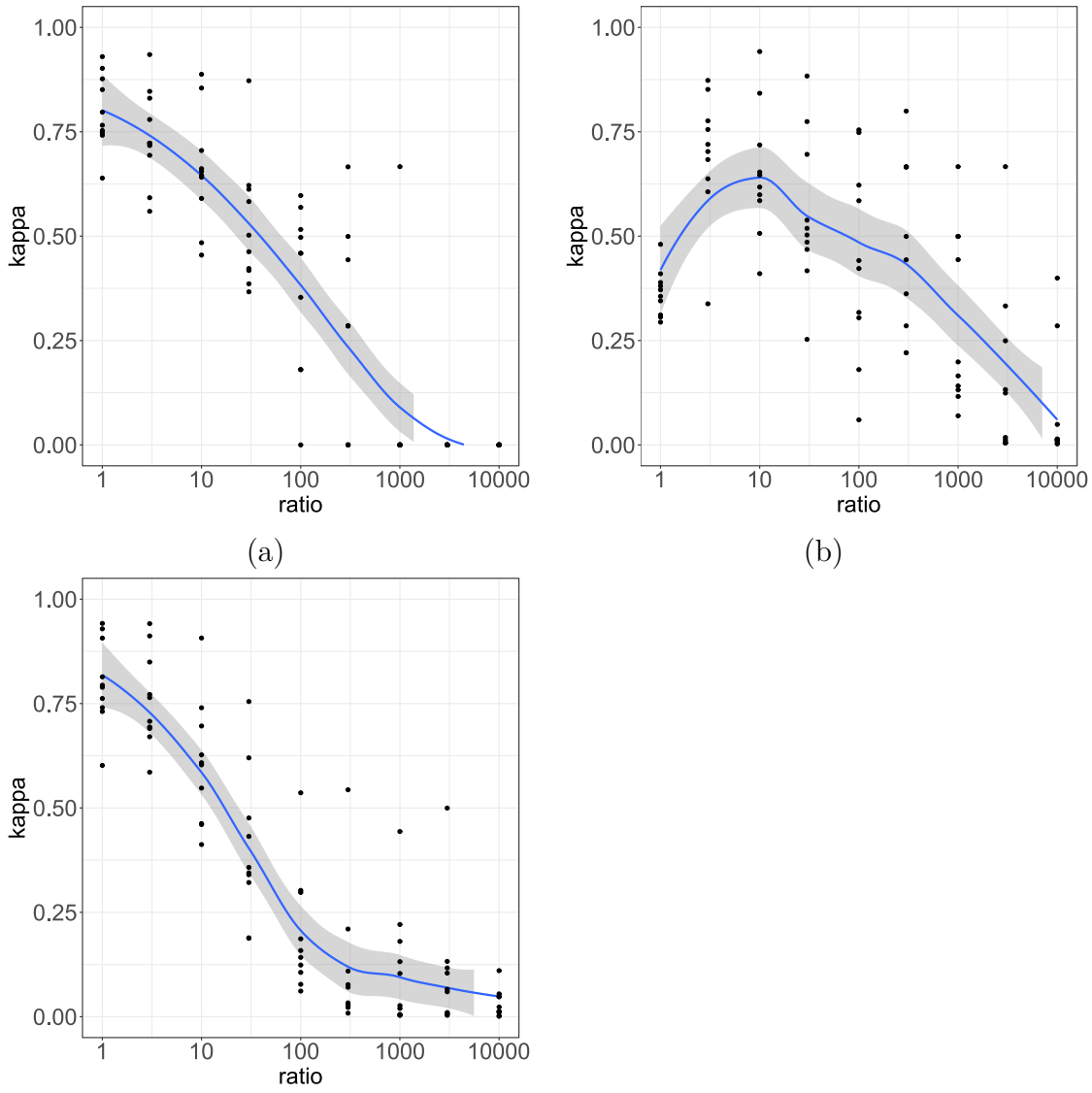


Figure 6.4: κ versus imbalance ratio for the fitness functions κ (a), MCC (b) and G-mean (c). A point represents an evaluation at a fixed imbalanced ratio. The blue line and gray area represent the conditional mean of the evaluation results associated with confidence intervals. Each plot goes from IOM 0 to IOM 4. G-mean is the best fitness function for low imbalance and MCC for high imbalance.

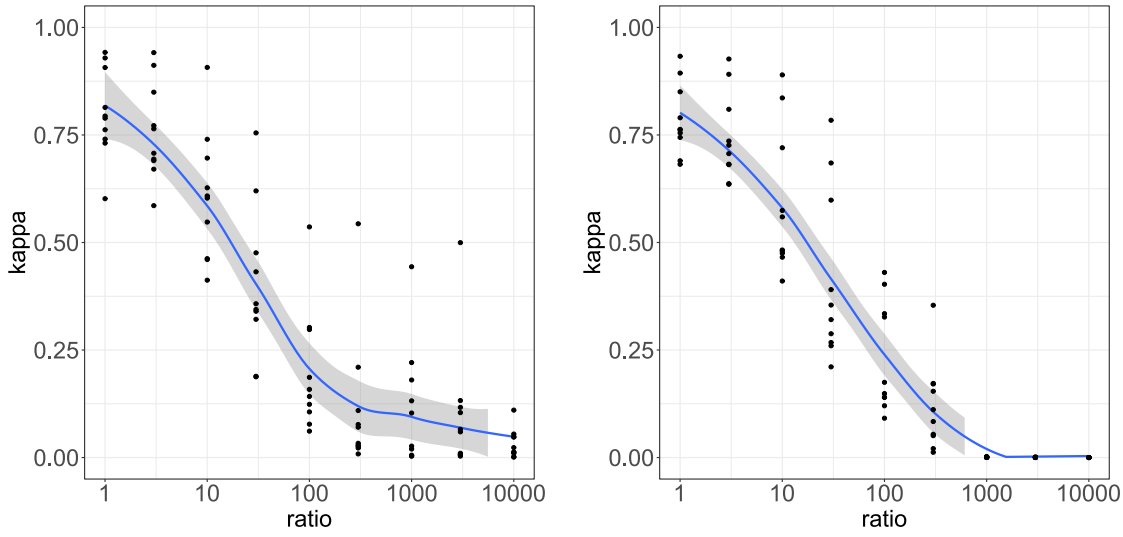


Figure 6.5: κ versus imbalance for the ranked selection phase from legacy TPG (a) versus the proposed selection phase (b). A point represents an evaluation result for a given minority class of the dataset for a fixed imbalanced ratio. The blue line and gray area represent the conditional mean of the evaluation results associated with confidence intervals. The proposed method, as expected, reduces the variance of the κ and slightly improves κ on moderate IOMs (lower than 2.5) but has lower performance on very high IOMs.

IOM \ Sub-batch size	50	100	200	500	1000	1500
0	0.88	0.89	0.90	x	x	x
0.5	0.70	0.71	0.78	0.79	0.81	0.73
1	0.66	0.76	0.75	0.75	0.79	0.76
1.5	0.63	0.71	0.79	0.79	0.84	0.79
2	0.57	0.66	0.75	0.79	0.72	0.72
2.5	0.23	0.33	0.34	0.42	0.38	0.51
3	0.18	0.11	0.20	0.20	0.40	0.42

Table 6.3: Impact of the sub-batch size when dealing with imbalanced data. The table sums up the κ values obtained with the class "0" as the minority class, for different IOM and batch size for 100 training generations. Firstly, we can observe the impact of the cardinality on the highest values of the κ measure when the IOM increases. Secondly, it appears that in most cases, data from large sub-batches are more easily classified than data from smaller batches. A big sub-batch is more likely to have one or several Positives in it and thus, the score attributed to this sub-batch is more likely to represent the real capabilities of the classifier. The $\times$ represent values that can not be obtained as the number of available data to explore these conditions can not be reached.

	Legacy TPG	TPG using F1-score	TPG using G-mean and Optimized selection
Accuracy	93.59	92.92	**94.08**
Precision	**98.01**	89.62	84.07
Recall	75.71	80.84	**93.94**
F1-Score	85.32	85.01	**88.73**

Table 6.4: Comparing classification results using different fitness functions. Although the learning on imbalanced data is more efficient using the revised selection mechanism and the G-mean function, the operational constraint on the false positive rate makes the legacy TPG more suitable to the intrusion detection problem.

CHAPTER 7

Contribution 3: Evaluating TPG for stream processing, incremental learning and high efficiency Anomaly-based Intrusion Detection System (AIDS)

7.1 Introduction

Results of this study have been published in [Sou+22b]. Intrusion detection consists in spotting the actions of attackers attempting to compromise the integrity, confidentiality, or availability of a computer resource [Zam01]. To enhance the security of a network, connections need to be analyzed for countermeasures to be deployed. In a realistic information system context, the connection rate is high, making it impossible to be analyzed by a human analyst in real-time. Thus, Network Intrusion Detection Systems (NIDS) have been created to facilitate the tasks of the analyst. As the network is changing over time, facilitating the update of Anomaly-based Intrusion Detection System (AIDS) is important [Mor11; Rai12; Sym17]. NIDSs work under the assumption that attacks packets share a common basis and similarities. This chapter tackles challenges for AIDS such as the diminution of the false-positive alert rate and the production of ancillary information when an alert is raised. Indeed, false alarms are usually very time consuming and can easily cause a detection system to be rejected by analysts. Furthermore, AIDS based on statistical inference require an (often costly) initial training that hinders system adapta-

tion. Finally, encrypted data packets can bypass the Intrusion Detection System (IDS) and prevent an attacker from being detected, motivating the choice of network flow logs analysis.

From the difficulties that are faced by current AIDS, Genetic Programming (GP) have interesting properties. Indeed, GP are incremental learning agents, able to create novelty from their evolution process at each generation. Solutions that emerge from this process are relevant candidate to face the ever-changing nature of the network environment. The Tangled Program Graph (TPG) intelligence, as designed by Stephen Kelly [KSH], is a Multi-Agent Reinforcement Learning (MARL) algorithm based on GP showing interesting properties of emergence. This emergence results in an interaction between a set of agents and the environment. The agents observe the environment and become more complex (tangled) in their response to observations. We also hypothesize that the properties of the TPG can also bypass some of the current difficulties of AIDS. Firstly, the TPG can train continuously and complex action behaviors have been demonstrated to emerge from its training, making it adapt over time to a dynamic environment. This online training also limits the offline training time required to obtain a correct model and the need of massive training data as it learns on incoming data without the necessity to store them. Furthermore, the online training is bound to the issue that incremental learning implies forgetting some of the knowledge. As demonstrated in Chapter 6, the TPG can be used as a classifier, raising alerts and giving classification information if needed. The current study focuses on the detection of attacks and thus, their classification is kept out of its scope. We use as training data the network flow information from both the CICIDS 2017 [Cyb] and the CSE-CIC-IDS2018 [SLG18; CC] datasets.

We developped a high performance monitoring system, named Secure-GEGELATI based on the TPG. IT aims at progressively automating the detection of intrusions in an Information System (IS) by observing the incoming data. It constitutes an AIDS probe which helps a security analyst to early detect an attack. The monitoring system is required to have nearly perfect precision, potentially trading it off for a low recall. Indeed, false alarms are extremely time consuming, as they bring analysts to finally uncover that no attack was currently performed. Conversely, missing intrusions is less costly, as it brings the security analyst back to the situation where no live monitoring system was present.

To be useful in practice, a live IDS monitoring system must:

- keep up with the pace of the IS incoming data,

- be embeddable in an always-on device, and thus energy efficient,
- raise extremely rare false alarms in a context where attacks are very rare events.

This chapter proposes the following:

- we study the GEGELATI TPGs learning capacities and energy efficiency on a realistic and complex application,
- we demonstrate that the properties of TPGs are well suited for building a learning-based NIDS. In particular, its energy efficiency and incremental learning capabilities on new incoming attack flows are studied,
- we introduce the open source Secure-GEGELATI Tangled Program Graph (TPG) system for learning-based intrusion detection.

The TPG learning method has a lightweight structure, its training and inferring processes are known to be fast and adequate for online training [KH17b]. This lightweight structure helps for online training on an embedded platform. The considered scenario is the following:

- Training phase: the analyst provides tagged data to train the stream-processing monitoring system, specifying whether an attack is ongoing or not.
- Monitoring phase: the live monitoring system is switched into a monitoring mode, continuously observes the IS and triggers an alarm when an attack occurs. With GP, the analyst can switch back the monitoring device into a training phase at any time and improve its sensitivity to new attacks.

Section 7.2 introduces state of the art methods for adaptive and stream processing intrusion detection and situates Secure-GEGELATI among them. We explain in Section 7.3 how the TPG can be adapted into a NIDS stream processing probe (The TPG algorithm was previously described and defined in Section 6.1). Finally, in Section 7.5, we compare the performance of Secure-GEGELATI with the state-of-the-art methods in terms of precision, stream processing capabilities and energy efficiency.

7.2 Related Work

This chapter is about creating an efficient and adaptable solution to the intrusion detection problem.

As described in Chapter 5, the network environment is dynamic making it a challenge to determine accurately which traffic is harmless and which is not. Moreover, due to the changes occurring on the network, new harmless activity may emerge. New emerging malicious activities may go along with these new harmless activities.

Some research papers focus on method to adapt to these changes over time, using incremental methods [Liu+19; Liu+20; Con+19; Wu+21]. For example, [Liu+19] applies incremental learning to deep neural networks in order to train an AIDS. The method is able to detect unknown attacks sample, even though the novel attacks belong to already known attack categories. The same critics can be made about [Liu+20] where GP is used under the same testing conditions. On the other hand, some research articles [Moh+19; APN18] focus on the design of an IDS, using incremental learning to adapt to these changes, but without demonstrating the functioning of the method on a live network making difficult to compare their results. For example, [Moh+19] reached high classification score on the KDD99 dataset, using an incremental deep learning classifier but no tests were made on a changing environment. Finally, other methods [LCS20; ZCS20] claim their solution suitable for live network inferring without demonstrating the accuracy of the solution on a dynamic network environment.

For their part, [Liu+19; Con+19; WRM03; Moh+19; GMS00; SLP22] measure the efficiency of their IDS and provide either bit rate or analysis time of network packets. The most efficient method [SLP22] used extra trees and applied feature selection (using a 43 features dataset) for an efficient and accurate detection reaching a peak performance of 44.10 MB/s analyzed. [Liu+19] provides an interesting comparison measure as they used an incremental learning method applied to a changing network environment, while providing their results in terms of analysis time. Using this method, one is able to examine 2.4MB/s of data.

The main differences between these IDSs lies in the learning method used. Indeed, [Con+19] used both a Neural Network (NN) coupled with a Support Vector Machine (SVM) to design an IDS. Similarly, Mohamed et al. [Moh+19] used Deep Neural Networks (DNNs) resulting in less processing capabilities. The feature selection and sparse neural structure used by Liu et al. in [Liu+19] made their IDS between 100 and 1000 times more

efficient in terms of stream processing capabilities than the aforementioned [Con+19] and [Moh+19]. On the other hand, lighter learning algorithms such as extra trees [SLP22] give the best performance in terms of stream processing capabilities. This comparative study of learning algorithms motivates the use of the lightweight structure of the TPG to face the intrusion detection problem.

Table 7.1 sums up the different research papers that designed AIDS to either fit the changes occurring on the network or that reaches high network packets processing performances.

7.3 The Secure-GEGELATI stream processing prototype

Secure-GEGELATI is a TPG-based system that has been designed to perform intrusion detection on an IS.

In Section 7.3.1, we present the changes applied to TPGs to use them in an IDS. Section 7.3.2 details reinforcement learning in a NIDS context. Section 7.3.3 presents the embedded version of our system. Finally, we describe in Section 7.3.4 a practical use case of the probe.

7.3.1 An Anomaly-based Intrusion Detection System

Secure-GEGELATI is a TPG system that has been adapted in the following ways:

1. Secure-GEGELATI is tailored to perform an inference task. To build Secure-GEGELATI, the TPG has been adapted to classification problems by increasing the probabilities of mutations of programs, and increasing the diversity of explored solutions. In particular, the mutations that cause the change of the outgoing edges to a different action are useful to discover samples from all the represented classes, including rare intrusion classes. After modification, Secure-GEGELATI is able to create a model for rare events.

2. It is tailored to produce classification with low False Positive Rate (FPR) and incremental learning. Originally, a TPG is designed to solve Reinforcement Learning (RL) challenges, i.e. to choose the actions to be applied to an environment so as to increase a reward. A NIDS requires primarily a high precision to get a low false

paper	Incremental	GP	accuracy	unknown attacks	novel attacks	changing traffic	changing topology	stream processing capabilities (MB/s)
[Liu+19]	x		98%	no	yes	no	no	2.4
[Liu+20]	x	x	87%	no	yes	no	no	x
[Con+19]	x		83%	no	no	yes	no	0.002
[Wu+21]	x		85%	no	no	yes	no	0.003
[Moh+19]	x		98%	no	no	no	no	0.02
[APN18]	x		89%	no	no	no	no	x
[LCS20]			96%	no	no	no	no	x
[ZCS20]			70%	no	no	no	no	x
[GMS00]			x	no	no	no	no	0.027
[SLP22]			>90%	no	no	no	no	44.10

Table 7.1: Adaptability of NIDS and performance measurement (MB/s) for stream processing. This table points out the published methods either able to adapt to changes on a network topology, traffic or attack traffic or methods used for stream processing for anomaly detection. A distinction is made between unknown attacks and novel attacks where the first one is a zero-day attack that is totally unknown for the network while the second one is a zero-day attack of an already known category.

alarm rate, and secondarily a high recall to detect as many intrusions as possible. As described in Chapter 5, the basic fitness function giving points for the true positives and true negatives and withdrawing points for each false positive or false negative alert suits the needs of an AIDS. Section 7.5.5 uses this fitness function with the additional constraint of not raising any false positive alerts during the training phase.

3. Secure-GEGELATI uses a selection mechanism that is more resilient to the presence of imbalanced data as described in Chapter 6.

4. It is tailored to high stream processing capabilities enabled by core parallelism and a training phase where each network flows are processed only once instead of analyzed by each TPG root teams.

Secure-GEGELATI faces two main problems:

- analyze network flow logs with high stream processing capabilities
- learn in a dynamic environment where new threats appear over time.

To keep pace with the input data stream while training, we studied two training modes. The first one trains each root team with the same input stream of data and the second one stacks the network flow logs into a FIFO and different root teams unstack data through their learning. In the second training mode, each root team is trained with different data making the imbalanced data problem more important. The two methods will be compared in results sections on Table 7.9.

Finally, in order to cope with changes on the network (see Table **??**) or with the appearance of novel threats (See Table 7.7 and Table 7.8), we altered the learning process in order to switch manually between training and highly imbalanced modes. This feature allows an analyst to update the probe when required. Figure 7.1 and 7.2 show the flow of data in both inferring and training mode.

7.3.2 A GP-based probe

We designed the Secure-GEGELATI system as a GP probe that:

- takes actions on a dynamic environment, similarly to the legacy RL TPG
- evaluates the internal model (inferring) and explores of the action space (training)

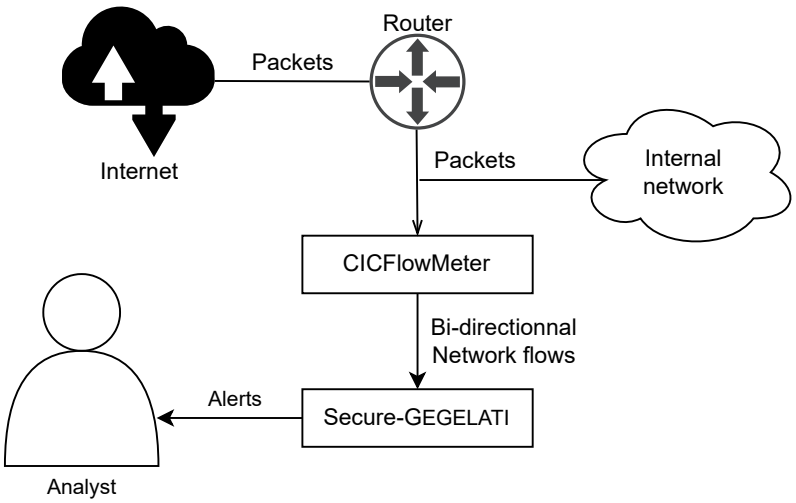


Figure 7.1: In inferring Mode, Secure-GEGELATI monitors Bi-directional Network flow logs (Network flows logs from both the request and the response) provided by the "CICFlowmeter" software from the raw packets logs. The analyst receives potential alerts.

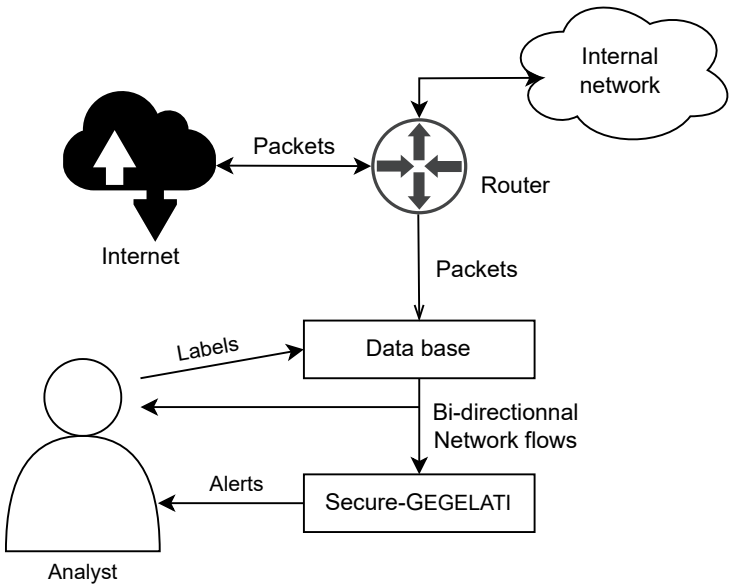


Figure 7.2: When Secure-GEGELATI is in training mode, it monitors network flows labeled by the analyst. The analyst labels this new training set of logs based on existing labels, expertise and other potential security mechanisms (such as signature-based IDS) already set-up on the network. The new training network flow logs are copied into to the previous training set. Secure-GEGELATI itself continues to raise alerts while training.

- learns through a fitness function, similar to rewards.

Secure-GEGELATI is a GP probe that exploits properties of RL to reduce the required amount of supervision, learn incrementally and provide a gradient free method.

We can compare Secure-GEGELATI with a RL-based probe as:

- it takes action on a dynamic environment, as we include the input data itself directly in the environment. When an action is taken (even though it is a classification action), the environment changes and a new line of network flow summaries is made available, enabling the agent to make new observations and update its action policy.

- its evaluation of the internal model is kept low to the benefit of a high exploration allowing the agent to fit to the novelty appearing on the dynamic environment. New attacks and threats can thus be detected.

- there is no immediate correction of the learning as in supervised learning, but instead a delayed score that is given to each agent root team at the end of a batch, fostering the selection of the best root teams for the cloning and the mutations.

7.3.3 A stream processing Embedded system

We use as a proof-of-concept target an octa-core Exynos 5410 SoC with four LITTLE ARM cores (A7) and four big ARM cores (A15), providing a set of 17 clock configurations from 200 MHz to 2.0 GHz for the A15 cores and from 200MHz to 1.4GHz for the A7 cores. The TPG graph being a highly dynamic and evolvable model, we choose to use this versatile platform and benefit from its high-energy efficiency, optimized for running in handheld devices. This platform constitutes a portable baseline for future studies on hardware acceleration. Accelerations of TPG programs on a SoC-FPGA is considered for future work.

Secure-GEGELATI exploits the platform parallelism. As a low-power device, the number of cores used for the application can be adjusted at initialization to keep pace with the incoming data flow while functioning at the lowest level of energy consumption. The clock frequency of the A7 cluster and the A15 cluster can be set at initialization and updated at runtime to prevent the probe from being flooded.

The determinism of GEGELATI helps in the validation of the embedded Secure-GEGELATI IDS on the octa-core Exynos 5410 SoC, ensuring that the training is occurring properly on the embedded platform.

To keep pace with the input stream of data in a training configuration, optimizations have been performed. Two different settings are considered:

- M1: during training, all root teams analyze the input data. The best root teams are selected at the end of the generation for replication and mutation. The least performing root teams are deleted. This is the legacy functioning mode of a TPG.
- M2: online training in a stream processing context forces us to analyze each network flow log only once. The best root team is not necessarily the one that will analyze a network flow log. The log is analyzed by the first root team available – depending on the parallelism/determinism of the application –.

The analysis during an online training of the probe is slower than while inferring. The results shown in next sections consider both approaches. In M2, network flow logs arrive in a stack and each root team unstacks one log and analyses it. Each root team thus trains on different samples.

7.3.4 Using Secure-GEGELATI as an IDS

At initialization, the analyst trains the Secure-GEGELATI IDS offline providing labeled data or online, using results from other signature-based or anomaly-based IDSs to provide the labels. After setting the parameters of the embedded device such as started cores and frequency of the core clusters, the probe starts training. At any time, the analyst is able to freeze the training to use the probe in inferring mode. To update the device, he only has to switch back into training mode. This is illustrated in Figure 7.3.

7.4 Experimental setup

The legacy CICIDS2017 dataset is used for training and inferring Secure-GEGELATI. The CSE-CIC-IDS2018 dataset is also used as it is more realistic in terms of rareness of malicious events and as it sums up generated traffic on a network topology similar to the one of a small-company. The datasets are presented in more details in the section 7.4.1

We want to show here that Secure-GEGELATI is a relevant contribution to AIDS design by measuring its performances on various experiments.

- Firstly, we measure its performances on the CICIDS 2017.

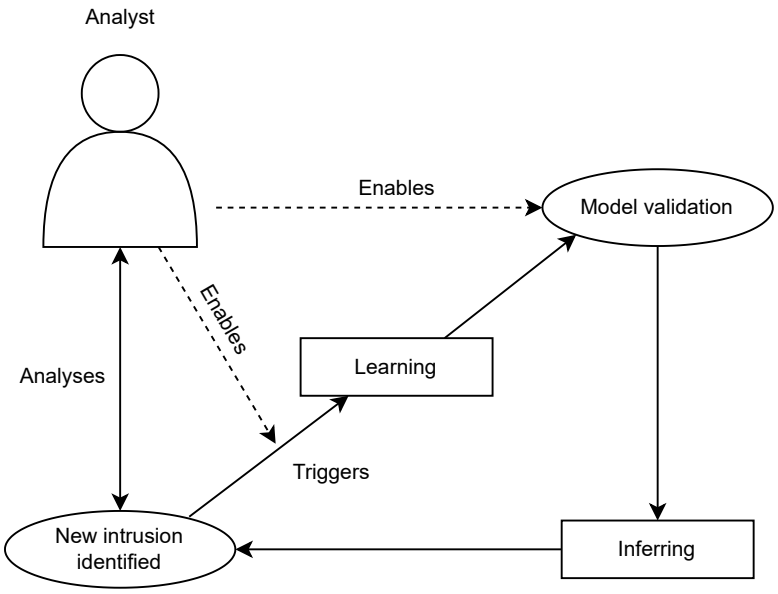


Figure 7.3: After an initial training, the Secure-GEGELATI IDS runs on the network. When a new intrusion is identified by Secure-GEGELATI (and confirmed) by the analyst, he updates the training set providing new labeled data and the probe is switched back into learning mode. New intrusion detection capability is checked on a validation set before turning the probe back into inferring mode.

- Then, to demonstrate the adaptability of the Secure-GEGELATI IDS, we run the following experiments:

  1. we measure the evaluation performances of a TPG previously trained on CICIDS 2017 on the CSE-CIC-IDS2018 dataset and show the adaptability of the TPG intelligence. The CSE-CIC-IDS2018 dataset represents changes in terms of network topology, available services, normal traffic and attack traffic. Furthermore, it introduces zero-days attacks from both known and unknown categories of attacks. We qualitatively compare the results to the state-of-the-art methods.

  2. We train a model without a category of attacks and study the reaction of Secure-GEGELATI when we introduce this attack in the dataset. We show that Secure-GEGELATI is able to evolve in order to discover the new attack and how it affects the learned model.

- We show that Secure-GEGELATI is a relevant solution in a stream processing context and that it is able to keep pace with the network flows of the CICIDS 2017 and 2018 datasets.

- We show that the Secure-GEGELATI IDS is useful on an embedded platform and measure its performance in terms of the amount of analysis per seconds. Details on the embedded platform are available in Section 7.3.3

- Finally, we measure the energy efficiency of the Secure-GEGELATI NIDS on the embedded platform.

7.4.1 Description of the datasets used in the experiments

The datasets CICIDS 2017 and 2018 are used in this study. Even though several drawbacks exists in both datasets, they both present advantages in terms of quality of generated traffic, uses of different operating systems and variety of services supported, diverse attacks while being recent datasets [Rin+19].

7.4.1.1 The CICIDS 2017 dataset

The CICIDS 2017 dataset is used as described in Section 5.4.1. The 15 classes dataset is turned into a two class problem characterizing the traffic as either normal or as an intrusion. The dataset is used with respect to the results provided in Chapter 5 in terms of representation bias mitigation. Except for the experiment under operational conditions in Section 7.5.5, the corrected dataset proposed by [ERJ21] is not used.

7.4.1.2 The CSE-CIC-IDS2018 dataset

The CSE-CIC-IDS2018 provides a similar dataset and is the result of a simulation of a much bigger IS. Indeed, the dataset generated comes from a set of 420 computers divided into five departments and thirty servers. The attacks are carried out from an “attacker" network composed of 50 machines using the Windows and Kali operating systems. The CSE-CIC-IDS2018 dataset is composed of the same categories of attacks as the CICIDS 2017 dataset but with slight differences due to the modernization of some attacks or the changes in operating system. The CSE-CIC-IDS2018 dataset is more representative of a realistic small company-sized IS.

Parameter Name	Value
maxNbActionsPerEval	**10000**
maxNbEvaluationPerPolicy	**500**
maxProgramSize	**20**
pAdd	0.5
pDelete	0.5
pMutate	1.0
pSwap	1.0
maxInitOutgoingEdges	**2**
maxOutgoingEdges	5
nbRoots	**360**
pEdgeAddition	0.7
pEdgeDeletion	0.7
pEdgeDestinationChange	0.1
pEdgeDestinationIsAction	0.5
pProgramMutation	0.2
nbGenerations	**100**
nbIterationsPerPolicyEvaluation	10
nbRegisters	8
ratioDeletedRoots	0.4
archive Size	**500**
archivingProbability	**0.01**
maxConstValue	100
minConstValue	0
pConstantMutation	0.5
forceProgramBehaviorChangeOnMutation	**true**
nbProgramConstant	0
nbThreads	16

Table 7.2: Parameters of the TPG framework used in this Chapter.

7.4.1.3 Adjustments

Some differences between the two datasets required pre-processing to have consistent data for training and evaluation of the solution. In particular, the CSE-CIC-IDS2018 dataset has two additional columns compared to the CICIDS 2017 dataset, and the CICIDS 2017 has information about header length whereas the other dataset does not.

7.4.2 Parameters of the TPG

Table 7.2 displays the parameters used in this Chapter to train TPGs. The parameters

of the TPG as defined in [Kel18] are used. The number of actions taken and the number of evaluation required are designed to widen the exploration space of the TPG. Due to the consequences of these parameters, the size of the programs have been reduced to reduce the execution time of the programs. The archiving probability is also reduced to limitate the memory impact of the probe on the embedded system. Finally as a feature developped by our team, we force the program behavior to change if the program goes through mutations.

7.5 Experimental Results

This section sums up the experiments results of the proposed protocol and evaluates the utility of the TPG algorithm in the design of a NIDS.

7.5.0.1 CICIDS 2017 Analysis

[SLG18] gives results on the CICIDS 2017 dataset in terms of Precision, Recall and F1 Score using seven Machine Learning (ML) algorithms. Those results have been reported in Table 7.3. Among those seven supervised learning algorithms, four stand out. Indeed, k-nearest neighbors (KNN), Random Forest (RF), Iterative Dichotomiser 3 (ID3) and the Quadratic Discriminant Analyzer (QDA) reach F1-scores above 90%. The Adaboost, Multi-Layer Perceptron (MLP) and Naive-Bayes algorithm reach lower detection rates.

Table 7.3: Results on the CICIDS 2017 dataset using various ML Algorithms as reported in [SLG18]

Algorithm	Precision	Recall	F1-score
KNN	0.96	0.96	0.96
RF	0.98	0.97	0.97
$ID3$	0.98	0.98	0.98
$Adaboost$	0.77	0.84	0.77
MLP	0.77	0.83	0.76
$Naive - Bayes$	0.88	0.04	0.04
QDA	0.97	0.88	0.92

A comparison metric for IDS is given by the Intrusion Detection Capability C_{ID} [Gu+06] based on information theory.

This C_{ID} is the result of the computation of the mutual information ($I(\vec{X};\vec{Y})$ having $\vec{X}$ being the inputs log class of the IDS and $\vec{Y}$ being the classifications given by the IDS on the input logs $\vec{X}$) normalized by the entropy of $\vec{X}$: $H(\vec{X})$. An IDS has to determine whether a log is normal or represents a threat. Secure-GEGELATI can be seen as a deterministic function that acts on the input stream ($\vec{X}$) and produces the output $\vec{Y}$ ideally being identical to $\vec{X}$ classes (i.e. "Benign" or "Attack"). The number of guesses represents $H(\vec{X})$ (i.e. the information content of $\vec{X}$) and the number of correct guesses represents $I(\vec{X};\vec{Y})$. More details on the C_{ID} can be found in [Gu+06].

The higher the C_{ID} is, the better the IDS performs. In practice, having a perfect model (FPR (False Positive Rate) and FNR (False Negative Rate) both value 0) leads to a C_{ID} of one . This value helps to choose the best trade-off between Precision and Recall.

7.5.1 Performance of the RF and the TPG algorithms on the datasets CICIDS 2017 and CSE-CIC-IDS2018

We compare the algorithms in terms of accuracy, precision, recall (or sensitivity) and F1-score. It is more interesting in this study to maximize both Precision and Recall (high F1-score) as we want to detect as many attacks as possible (high recall) while generating as few false-positive alerts as possible (high precision).

7.5.1.1 RF implementation

Even though the ID3 algorithm produced the best results (see Table 7.3), we use RF to compare our results as no open-source multi-threaded and low level implementation of ID3 was found for fair comparison. RF have results close to the ID3 algorithm and a multi-threaded low-level implementation is available in the Ranger framework [WZ15]. This implementation is useful to try the RF on the embedded platform. Table 7.4 sums up the precision, recall, F1-score and accuracy of the RF algorithm.

We can see that there are some differences with the results given in Table 7.3 and in [YSS19] using a state of the art RF algorithm. Those differences are due to a custom RF parameters tuning to reach a 100% precision. Such a precision is preferable because in the intrusion detection field, false positive alerts are costly time consuming.

Table 7.4: Performance metrics using RF on the CICIDS dataset for different training time + evaluation time.

Time (s)	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
183	94.5	97.6	74.1	84.3
539	95.5	99.7	77.3	87.1
1867	94.8	99.8	73.9	84.9
3543	92.2	100.0	60.3	75.3

7.5.1.2 Using the TPG to analyze CICIDS

Using the GEGELATI framework gives good results in terms of accuracy, precision, recall and F1-score as shown on Table 7.5.

Table 7.5: Performance metrics using Secure-GEGELATI on the CICIDS dataset depending on the total time (training + evaluating at the end of each generation on a disjoint testing dataset).
Comparing with Table 7.4, we can see that for a similar amount of time, the performances of both algorithms are close. In particular, we are not able to reach a 100% precision with Secure-GEGELATI but the higher recall makes it competitive.

Time (s)	Accuracy (%)	Precision (%)	Recall (%)	F1-score (%)
363	86.27	66.65	63.05	64.80
720	91.89	97.10	61.33	75.18
1701	94.74	96.72	76.32	85.31
3847	96.68	99.12	84.18	91.04

Comparing Table 7.5 with Table 7.4, we can see that for a similar amount of training time the results of Secure-GEGELATI are slightly better than the results of the RF algorithm. On one side, RF obtains no false positive alerts (really saving for an analyst). On the other side, the recall of the method is quite low. The RF implementation of the NIDS misses 40% of the attacks. Secure-GEGELATI is not able to detect attacks with a 100% precision but it is incorrect less than one percent of the time. On the other side, Secure-GEGELATI is able to detect over 84% of the attacks of the dataset. The learning of the RF being conditioned by statistics, a fast RF-based IDS will difficultly be able to detect the least represented classes. For example, Heartbleed or SQL-injections represent

less than 0.001 % of the data available in the CICIDS 2017 dataset. A RF-based probe able to detect those attacks could be designed by changing learning parameters and thus increasing the learning time of the IDS.

7.5.2 Adaptability of GEGELATI

7.5.2.1 Inferring the previous models to the CSE-CIC-IDS2018 dataset

In real-world conditions, the network environment is dynamic. The network topology can change, new services or tools can be installed and new user-behavior will be discovered. Even though switching from a tiny IS (represented by the CICIDS 2017 dataset) to a company-sized IS (CSE-CIC-IDS2018), we want our algorithm to be robust and to keep detecting as many threats as possible (low false-negative rate / high recall) while not generating false-positive alarms.

We sum up in Table 7.6 the evaluation of CSE-CIC-IDS2018 for both the RF algorithm and Secure-GEGELATI trained on the CICIDS 2017 dataset.

Table 7.6: Inferring models previously trained on the CICIDS 2017 dataset on the CSE-CIC-IDS2018 dataset

Algorithm	Accuracy	Precision	Recall	F1-score	C_{ID}
RF	70.58	100	0.08	0.16	4×10^{-3}
TPG	91.0	95.3	24.5	39.0	0.15

The difference between the results presented in Table 7.6 comes from the learning mechanisms. In the first case, with RF, the learning is done by studying the whole dataset as well as selecting the discriminating variables and thresholds that permit a classification without false positives. When the available information changes, that threshold and variables become less accurate for the classification of the new net-flow logs and leads to more classification errors. We keep detecting 1 threat over 4 using Secure-GEGELATI (see Table 7.6, recall column). Even if the content of the information changes, the TPG selects the discriminating variable and applies a program on those. The sequence of programs leading to the raise of an alarm might still activate, even though the observed values changed.

Although the use of RF does not generate false-positive alerts, it raises a mean of an alert every 1250 attacks whereas Secure-GEGELATI is raising a positive alert every

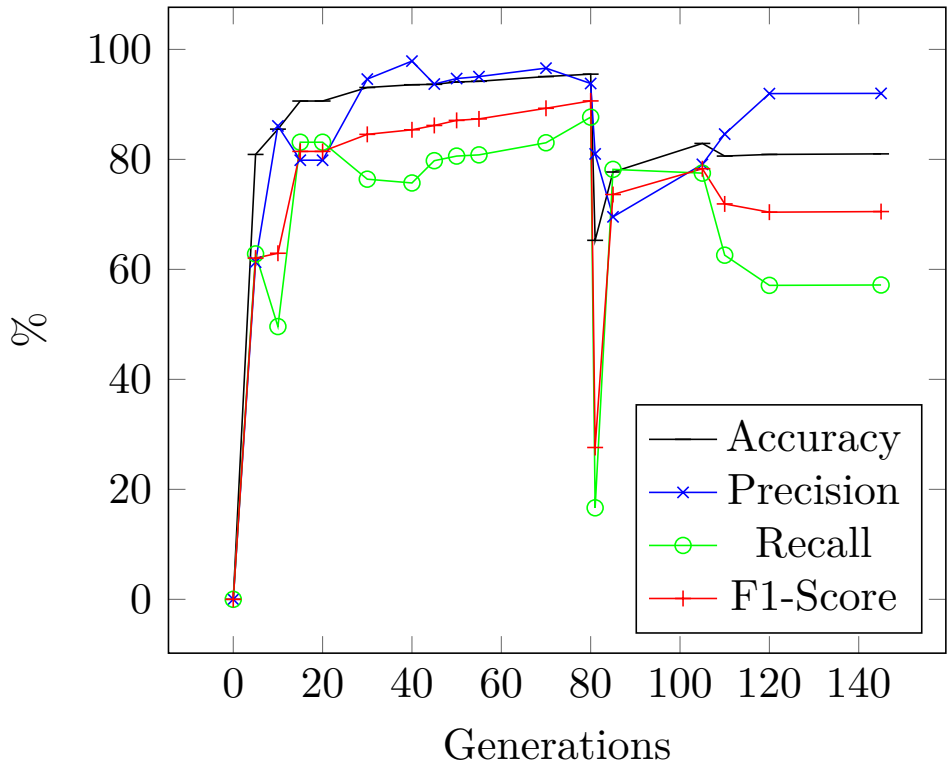


Figure 7.4: Performance metrics using Secure-GEGELATI on the CICIDS dataset depending on the total time (training + evaluation time. At generation 80, as Secure-GEGELATI reaches over 95% of its detection capabilities the analyzed connections are switched to CSE-CIC-IDS2018 instead of CICIDS 2017).
Even though drastic changes are applied to Secure-GEGELATI, the learning agent is able to adapt and detect intrusions with a high precision.

4 attacks with 95% of precision. We prefer the use of the Secure-GEGELATI IDS as $C_{ID}(TPG) = 0.15$ and $C_{ID}(RF) = 4 \times 10^{-3}$.

7.5.2.2 Discovering new categories of attacks

To demonstrate the detection of novel attacks, we carefully create two new data sets from the CICIDS 2017 dataset. The first one $D1$ is the CICIDS dataset without all the attacks labeled as “Port Scan" and the traffic in between Port Scan attacks. The second $D2$ is the CICIDS dataset without all DoS Slow Loris, Dos slow HTTP-test attacks and Port Scans and the Benign traffic in between those attacks. We train offline on $D1$ for 50 generations and add the port scan network flows to the existing dataset while Secure-GEGELATI worked in inference mode. After a while, we re-train the probe and write down the results

after 1 generation and after 50 generations of re-training. The same experiment is run with the second dataset $D2$. Table 7.7 shows the results of the experiment were Port Scan attacks are zero-days attacks whereas Table 7.8 shows the results of the experiment where the DoS attacks (all categories) are zero-days attacks.

We can see in Table 7.7 that most attacks are less detected after 50 re-train generations than after 1 re-train generation, marginally for some, significantly for others. This is partly due to the data imbalance described in Table 5.1 and also to the inner reward mechanism trying to prevent Secure-GEGELATI to raise false positive alerts. Before the modification of the dataset, Dos Hulk is the predominant attack class in the dataset. When Port Scans are added, it becomes the second most represented class, making it important for Secure-GEGELATI to detect it. Port Scans are known to have a signature and thus it is more likely that Secure-GEGELATI will come out with a good set of observation to detect them efficiently. Dos attacks are typically discovered through their volume which is something Secure-GEGELATI is not doing. Secure-GEGELATI can eventually come out with correct observations to reach back 100% of detection on the Dos-Hulk attack through an extended training.

Tables 7.7 and 7.8 sum up that the TPG is agile enough to detect new threats as they come and can be retrained online to update its knowledge base and keep performing a precise detection without generating too many false-positive alerts. Secure-GEGELATI tends to maximize its rewards and thus, attacks that have only a few samples in the dataset are more likely not to be detected.

7.5.3 Stream processing and energy efficiency of Secure-GEGELATI

The training time T of the TPG is conditioned by several factors:

- the number of samples or number of connections to analyze. A single policy takes a time $T = t$ to analyze a sample of the network flow logs and takes $T \approx n \times t$ to analyze n samples. The approximation is due to the depth of the chosen path. The

Table 7.7: This table sums up the per class true positives (and true negatives for the BENIGN class) when adding Port Scan attacks to the training set after 50 generations. The inferring results are very different from the results of the offline training due to the change of the evaluation set (required to insert the port scan attacks in the dataset). Note that without knowing anything about Port-Scans, the TPG model is able to raise an alert for 40% of them. Retraining causes an instant drop of the True Negative Rate and an instant raise of the True Positive Rate (TPR). Secure-GEGELATI tends to fit the most frequently occurring data of the dataset and thus becomes really good at detecting port-scan while keeping a low false-positive rate. × represents irrelevant data as they are not part of the evaluation set. Some attacks were not present in both evaluation sets or never detected.

class	train (50 gen)	Inferring	re-train (1 gen)	re-train (50 gen)
BENIGN	96.6	95.7	78.9	99.8
XSS	4.8	5.6	5.6	0
DoS slow Loris	29.1	42.6	38.0	36.4
Dos slow HTTP-test	58.9	66.0	63.8	59.6
Dos hulk	94.8	100	100	76.9
Port Scan	×	42.6	99.4	99.3
TPR	90.9	38.9	87.3	86

more relay-teams descended while running the algorithm, the more time it takes to take an action.

- The number of root-teams will have a direct influence on the training time. A root-team takes $T \approx n \times t$ to train and R root-teams take $T \approx n \times t \times R$.

To train the algorithm, we can choose to send the same sample to each root-team (M1). In this case, the analysis rate values: $A_{rate}(M1) \approx \frac{n}{T}$. Or we can train the algorithm sending samples to root-teams as they come (M2) which results in a boost of the analysis rate: $A_{rate}(M2) \approx \frac{R \times n}{T}$. We sum up in Table 7.9 the rates obtained at different stages of the training on the CICIDS 2017 dataset using a batch size of 10,000 samples.

Table 7.8: This table sums up the per class true positives (and true negatives for the BENIGN class) when adding Dos attacks to the training set after 50 generations of offline training. The inferring results are very different from the results of the offline training due to the change of the evaluation set (required to insert the DoS attacks in the dataset). This time the model is not able to detect any DoS attack in inferring mode. Retraining causes an instant drop of the True Negative Rate. Secure-GEGELATI tends to fit the most present data of the dataset and thus fits to detect Dos Slow-Loris and DoS Slow-HTTP test while keeping a low false-positive rate. × represents irrelevant data as they are not part of the evaluation set. Some attacks were not present in both evaluation sets or never detected. The data imbalance is described in Table 5.1

class	train (50 gen)	Inferring	re-train (1 gen)	re-train (50 gen)
BENIGN	100	100	86.1	99.9
Brute-force	75.0	67.5	67.5	67.5
XSS	86.2	94.4	94.4	94.4
DoS slow Loris	×	0.0	20.9	20.9
Dos slow HTTP-test	×	0.0	59.6	59.6
Port Scan	×	0.0	99.7	99.4
TPR	48.2	10.3	96.3	96.1

Note that Table 7.5 was obtained using the second method where samples are analyzed as they come and the root-teams do not learn on the same samples.

As seen in Section 7.4.1, the CICIDS 2017 dataset produces a mean of 50 connections per seconds and peaks to 170 connections per second. The Secure-GEGELATI algorithm is able to keep pace with the dataset using a X86 architecture. Through the generations, the graphs get more complex as relay teams are added. This is why the performance of the algorithm decreases with time.

As the peak number of connections per seconds on CICIDS values 170 connections per second, the embedded design must be efficient enough to perform online training at this rate. It is recalled that the four A7 cores of the Exynos 5410 platform can run at a maximal frequency of 1.4 GHz and the four A15 cores, at 2.0 GHz. We present in Table 7.10 the training times on the Exynos 5410.

Table 7.9: Measuring the number of connections analysis per seconds (A_{rate}) using Secure-GEGELATI. The first method (M1) trains a TPG by sending identical data to all teams whereas the second method (M2) stack all the data in a buffer and teams unstack the data one at a time. In method M2, teams are training with different data through time.

Gen.	Training time (s)	$A_{rate}(M1)$	$A_{rate}(M2)$
1	27.29	1832	916052
20	38.53	1297	642861
40	52.89	945	472643
60	58.61	853	426577
80	66.03	757	378615

Table 7.10: Reachable number of connection analysis per seconds using the Exynos 5410 with M2. We effectuate the training on TPG using 200 root-teams and training over a batch of 500 connection summaries (100,000 network flows analyzed). The frequencies F_{A7} and F_{A15} are in GHz.

Cores	F_{A7}	F_{A15}	Train (s)	A_{rate}
4A7 + 4A15	1.4	2.0	7.52	13294
1A7 + 4A15	1.4	2.0	10.49	9533
4A7	1.4	-	22.88	4371
3A7	0.2	-	239.21	418
2A7	0.3	-	258.00	387
2A7	0.2	-	614.39	162
1A7	0.2	-	1286.58	77

The embedded platform is able to process the entirety of the stream, even during the training phase. The Table 7.10 shows that the rate reachable using Secure-GEGELATI using 2 A7 cores at 300 MHz is superior to the connection rate on the CICIDS 2017 dataset.

7.5.3.1 Energy efficiency of the IDS

The x86 Intel Xeon W-2145 processor used for previous experiments in Section 7.5.2 has a 140 Watts peak thermal dissipation power (TDP). Using this architecture, Secure-GEGELATI analyses up to $378k$ connections per seconds during training and $480k$ connections per second using the graph on inference.

The Energy efficiency (E_{eff}) of the x86 platform is thus :

- $E_{eff}(Training) = 2.7k\ connections.W^{-1}$
- $E_{eff}(Inferring) = 3.4k\ connections.W^{-1}$

As a comparison, the RF IDS has an energy efficiency of $E_{eff}(Inferring) = 400\ connections.W^{-1}$ using a x86 architecture on inference. We used the framework RANGER [WZ15], a parallel framework to train RF for a fair comparison with a parallel TPG-based IDS. The Secure-GEGELATI software has thus 8 times the energy efficiency of RF-based IDS.

On the Exynos, the chosen solution (using 2 A7 cores at a 300MHz frequency) consumes 0.05 W. It results in an energy efficiency $E_{eff}(Inferring) = 200k\ connections.W^{-1}$.

Monitoring a theoretical system functioning at 1TBps and saturated in traffic requires at its maximal capacity the analyses of 700M connections. Under the hypothesis of perfect scalability, this probe can function on such a system at 35W.

7.5.4 Comparison with the state of the art

Table 7.11 sums up the state of the art contribution over adaptive and real time methods for the design of AIDS and compares it with our three different tests. During the first test (see Figure 7.7), Ports Scans attacks were added after a pre-training of the probe. At this time, the Ports Scan attacks are considered as zero-days attacks although the knowledge of DoS attacks helps the probe to detect some of the Port Scan attacks when inferring. The incremental learning process of the TPG is able to detect port-scans accurately after 50 generations. These results can be compared with [Liu+20] as both study use incremental learning and observe the reaction of the AIDS to the appearance of a zero-day attacks while having prior information on the attack. In the second test (see Figure 7.8), DoS attacks and Port Scans attacks were added to the training and testing dataset. No prior information is known by the probe when the attacks are integrated into the dataset. Once again, the TPG agent is able to fit the data and to detect the attacks. Finally (see Figure 7.4), a transfer is performed from the CICIDS 2017 dataset to the CSE-CIC-IDS2018 dataset. It results in changes of the network topology, of data traffic (cloud servers are added into the infrastructure) and of attack traffic, with some attacks where prior information is available and other attacks that are zero-days attacks. Although the TPG is able to keep a high precision (95.3%) and accuracy (91%), the method tends to miss a lot of attacks. It is mainly due to the short simulation time and to the extents of the changes occurring on the network. A complementary study is required. The Secure-GEGELATI

IDS is able to analyze as much as 149MB/s of network data (which represents a total amount of 100000 network flows per seconds).

7.5.5 Training an AIDS for operational conditions

AIDS are useful for an analyst to determine whether or not unknown traffic is dangerous for the IS. Analysts examine packets that triggered an alert through the AIDS. An analyst is known to be able to analyze between 10 and 20 events per day making it a key component for an AIDS not to generate false positive alerts. We train an AIDS using the corrected version of the CICIDS 2017 dataset as described in [ERJ21] with the constraint that no false positive alerts should be triggered during the training phase. The reduction of the representation bias as described in Chapter 5 is used, as well as the specification of the imbalanced learning TPG as described in Chapter 6. Table 7.12 shows the results in terms of True Negative Rate (TNR) and True Positive Rate (TPR) of this AIDS after inferring. The table also displays the total amount of alerts and the amount of false positive alerts of the AIDS.

Although the aforementioned AIDS suits the operational requirements in terms of false positive rate, it mainly detects the Denial of Service (DoS) and Port Scan attacks, representing large amount of network packets. Thus, subtle intrusions such as **Code Injection** or **Infiltration** are less likely to be detected.

7.6 Discussion and future work

TPGs and RF strongly differ in their learning mechanisms. While RF ingests all the training data at once to build a model, TPGs progressively incorporate it and can recover from badly labeled data by feeding the system with new correctly labeled data. The training time and performance of RF is strongly impacted by the amount of data and the chosen learning parameters (tree depth, number of trees, etc) that need to be tuned. With a fine tuning of these parameters and large computation time, the models of predictions can be very accurate. The drawback of RF is however that they need to be trained and fine tuned from scratch to incorporate a new intrusion type.

TPG training takes a longer time than RF to converge to an accurate model, as their model results from trials and errors. Parameterization of the TPG agent is rather easy, the state of the art parameters used by Stephen Kelly [KH17b] suit most learning application

Paper	Incremental	GP	Accuracy	Unknown attacks	Novel attacks	Changing traffic	Changing topology	Stream processing capabilities (MB/s)
[Liu+19]	x		98%	no	**yes**	no	no	2.4
[Liu+20]	x	x	87%	no	**yes**	no	no	x
[Con+19]	x		83%	no	no	**yes**	no	0.002
[Wu+21]	x		85%	no	no	**yes**	no	0.003
[Moh+19]	x		98%	no	no	no	no	0.02
[APN18]	x		89%	no	no	no	no	x
[LCS20]			96%	no	no	no	no	x
[ZCS20]			70%	no	no	no	no	x
[GMS00]			x	no	no	no	no	0.027
[SLP22]			>90%	no	no	no	no	44.10
[Sou+22b] - (1)			97%	no	**yes**	no	no	**149**
[Sou+22b] - (2)			**99%**	**yes**	no	no	no	**149**
[Sou+22b] - (3)			91.8%	**yes**	no	**yes**	**yes**	**149**

Table 7.11: Comparing the detection accuracy and stream processing capabilities of our three tests ((1) adding Ports Scans attacks, (2) adding Ports Scans and DoS attacks, (3) changing the network topology, services and data) using Secure-GEGELATI. The detection accuracy is above 90% even though massive changes have been applied to the training and testing dataset. The method fits the ever changing network data through time using the incremental property of the TPG. Finally, the probe is able to process a maximum of 149MB/s (around 100000 networks frames per seconds), overcoming the results of [SLP22]

.

TPR (%)	TNR (%)	Number of Positive alerts	Number of false positive alerts
28.83	99.96	2964	15

Table 7.12: Results of a AIDS trained with the constraint of not generating any false positive alerts during the training phase. The FPR is low (0.04%) which makes it suitable for operational conditions. Most positives are missed (71%).

and the sensitivity of the parameters is low. Light modifications of the parameters do not affect the learning process much but can bring interesting properties such as light graph structure or ability to detect rare events. However, the positive point is that TPGs are able to detect attacks while training. Online training makes TPGs more fit to detect novelty. Even if TPGs take time to converge, they finally adapt and are able to detect new threats as they arrive.

The Secure-GEGELATI TPG algorithm in its current form is still limited for the observation of rare events. Indeed, rarely activated teams and programs may be deleted, which can cause issues in real-world conditions for very rare intrusion detection. In the CICIDS 2017 dataset, there is 1 attack out of five connections so the test conditions overrate intrusions and the problem does not appear.

A false alarm rate of 0.2% is low but results in too many false alarms in the practical context of an IDS. Proposing a training with the constraint of not generating false positive alerts can result in lower FPR but also leads to detection of attacks having a higher impact on the volume of network packets. The current detection system needs to be complemented with temporal filtering, exploiting the multi-connection nature of most intrusions, to reach the extremely low false alarm rates required in practice. This objective is kept as future work.

The study of Secure-GEGELATI has proven that TPGs are well suited for adaptive network intrusion detection. However, the Secure-GEGELATI TPG model currently requires a large amount of labeled data to converge. As a future work, we intend to reduce the need of supervision by introducing semi-supervised learning into the TPG framework. Furthermore, being able to process the stream of data is interesting but real-time processing of the network flow would be more interesting as it would give guarantees on the ability of the probe to analyze the traffic.

7.7 Conclusion

This chapter has introduced the Secure-GEGELATI learning-based stream processing NIDS and has demonstrated the agility and energy efficiency reached by the resulting network probe. Secure-GEGELATI combines several capabilities required by a NIDS: rare events detection with very few false alarms, as Secure-GEGELATI detects more than 80% of the intrusions with a precision over 99%; high-energy efficiency, as Secure-GEGELATI is 8× more energy efficient than RF in the same inference conditions and can process 3× more data than its concurrent [SLP22]; high scalability as the speedup over 4 embedded cores reaches 96.9% of the optimum. Furthermore, thanks to the TPG intelligence, Secure-GEGELATI is a flexible tool that adapts to novel threats. The system can anytime be switched into a training mode and be fed with new labeled benign and intrusion data to improve its capabilities. Finally, a training for an operational AIDS is performed to show that the TPG can be trained to raise alerts without generating too many positive alerts.

Part III

Conclusion

CHAPTER 8

Conclusion

This thesis focuses on different aspects of Network Intrusion Detection Systems (NIDS) design. It has been showed that the data used for training is a key component for the production of qualitative Anomaly-based Intrusion Detection System (AIDS). Firstly, network intrusion datasets must be handled with care in order to prevent errors in the learned model. Indeed, we show that representation bias in a network intrusion dataset has great impacts on the model decisions. Furthermore, errors in the labels of the dataset are costly while creating a model of the network traffic. Experimental results show that a model learned on a dataset with inaccurate labels performs poorly on a corrected version of the same dataset. The Tangled Program Graph (TPG) capabilities are explored as a resilient framework to this exposed label bias. In a second time, the same approach is lead to show the resilience of the TPG to data imbalance. This intrinsic characteristic of the network intrusion datasets tends to degrade the quality of the models performance. The impact of the data imbalance on the TPG framework is studied and an algorithmic mitigation is proposed to enhance the robustness of the TPG framework to the imbalanced classification task. Finally, the TPG is used as an incremental learning framework, taking profits of its Genetic Programming (GP) algorithm. This incremental learning process helps in the reduction of the deployment bias issue. Indeed, training offline on a dataset which differs from the real network condition causes a drop in terms of the detection capabilities of the AIDS probe. The TPG-based AIDS is demonstrated to be usable as

an embedded probe capable of keeping pace with the incoming network flow of data and functioning at low power consumption. The three main contributions are over-viewed here after.

8.1 Research contributions

Chapter 3 has introduced key notions to progress on the global understanding of the network security applied to intrusion detection. Learned Anomaly-based Intrusion Detection System (AIDS) rely on Machine Learning (ML) techniques and on a rigorous use of the network data which is detailed in Chapter 4. These preliminary chapters are the backbone of the three research contributions proposed in this manuscript.

8.1.1 Assessing the biases of IP networks intrusion detection datasets and evaluating their effect on a TPG-based AIDS

In Chapter 5, both the label and representation bias are studied. These bias exist in the CICIDS 2017 database and have an impact on the learning process of the TPG. For the first part, representation label introduces easy correlations between connections and their associated classes providing identification features such as information on the source and destination IP and ports, as well as timestamps. This bias exist since the generation of the CICIDS 2017 dataset uses a cluster of attackers that are the only source of the attack traffic. Experimental results show that these features are used by the ML agent in order to decide whether or not an alert is to be raised. Furthermore, adding noise in those variables for inference leads to a drop of 18% of the True Negative Rate (TNR) of the TPG, a particular sensitive indicator. A pre-processing of the dataset to exclude the biased features is thus recommended. Secondly, using biased labels on the TPG framework has a strong impact on the learning. Experiments using a model trained on the biased dataset showed a mean drop of per-class detection rates of 61%. Furthermore, this drop reaches up to 99% on the Ports Scans attacks. The resiliency of the TPG is tested on the original dataset where some of the attack traffic is synthetically biased. Results show that the TPG method is resilient to up to 25 % biased labels where its True Positive Rate (TPR) still reaches 90% of its maximum.

8.1.2 Study of the impact of data imbalance on TPG performance

Chapter 6 has studied the effect of imbalance on a binary classification task based on genetic programming. We demonstrate that the Cohen's Kappa is, among standard metrics, the one to use to evaluate a classifier on imbalanced problems. The fitness measurement during the selection phase of the genetic process has a strong impact on the built model. We propose algorithm modifications on fitness and selection phases to support imbalance with genetic programming. Indeed, the imbalanced classification problem is bound to a training phase where selection and fitness functions are key components, as the genetic programming algorithm uses selection phases to keep the individuals that perform correctly, based on their respective fitness. This chapter shows that the G-mean fitness function is a good candidate for low Imbalance Order of Magnitude (IOM) while MCC fits better the highly imbalanced problems. In practice, it is shown that learning in an environment with an IOM of 4 is possible, but with very degraded performances. Although hard to measure, the natural IOM of intrusion detection in real network data is above 4.5. Finally, we show that the robustness of GP agents facing moderate IOM problems can be increased through the use of the proposed selection phase, at the cost of lower performance on higher IOMs.

8.1.3 Evaluating TPG for stream processing, continual learning and high efficiency AIDS

Chapter 7 has introduced the Secure-Gegelati learning-based real-time NIDS and has demonstrated the agility and energy efficiency reached by the resulting network probe. Secure-Gegelati combines several capabilities required in a NIDS: rare events detection with few false alarms, as Secure-Gegelati detects more than 80% of the intrusions with a precision over 99% on the CICIDS 2017 dataset; high-energy efficiency, as Secure-Gegelati is 8× more energy efficient than Random Forest (RF) in the same inference conditions and can process 3× more data than competing solutions [SLP22]; high scalability as the speedup over 4 embedded cores reaches 96.9% of the optimum. Furthermore, thanks to the TPG intelligence, Secure-Gegelati is a flexible tool that adapts to novel threats. The system can anytime be switched into a training mode and be fed with new labeled benign and intrusion data to improve its capabilities. Finally, a training for an op-

erational AIDS is performed to show that the TPG can be trained to raise alerts without generating too much positive alerts.

8.2 Prospects and Future Works

This thesis has focused on the following five main research questions:

- How can Intrusion Detection System (IDS) detect new intrusions while raising very few false alerts?
- How should network intrusion data be fairly used and can we improve the resiliency of intrusion detection methods to unfairness?
- Can we still learn from highly imbalanced data from algorithmic mitigation only?
- Can incremental learning help in fitting ever-changing network data over time?
- Is it possible to design a low power embedded intrusion detection system that keeps pace with the incoming data flow?

This thesis is a small step in these particular directions and opens the gate for more advances on those research topics.

8.2.1 Biases network data handling

The use of qualitative and labeled network data is a first step in the design of an IDS. Unfortunately, qualitative and representative network data is difficult to generate, costly to accurately label and tends to be outdated rapidly. These issues constitute a huge drawback when it comes to train an IDS from these data. Thus, we either want to have qualitative network data or robust algorithmic methods that help dealing with low quality data. As a future work, a first approach would be to create a method that unifies the export of network flow data features in order to give a representation bias free feature set from the network packets data. A first step in that direction as been conduced by Sarhan et al. [SLP22].

Network data are often generated by deterministic scripts that reflect a defined behavior. Recent research focusing on learning from human demonstration [BRK21] could be a next step for the simulation of representative human-like behavior in order to have

less deterministic data generation and thus more interesting behavior analysis from the ML-based IDS.

8.2.2 Imbalanced learning: algorithmic mitigation of GP methods

Imbalanced learning topic is a very challenging problem. The imbalanced data problem has a strong impact on the score and evaluation functions. This impact is more marked on gradient-based ML methods because of the quantization of the fitness functions that occurs when learning with high data imbalance. Thus, it motivates the use of a gradient-free TPG method to approach the imbalanced data problem. I strongly believe that the inner modification of known algorithms can conduct to better imbalanced data classification performances. Most of the work -including ours- focuses on an external feedback function that is a function of the correct positive classes found by the classifier. It has the advantage to testify the performance of the model, but has the drawback of shadowing several variables into one function. Our main idea is to think about "local" rewards that comes for different components of the TPG graph, at different times. At any time, a team, program and a root team can be rewarded (potentially with different functions) and the selection and decimation mechanisms intervene at all the different reward levels during the training. These "inner" rewards should not depend on the global performance of the ML agent, but from local appreciation of the graph components. A basic example would be to reward a team at each time it has been selected or a program when it reaches the higher bid of the pool.

8.2.3 Adaptive IDSs as high performance probes

We have shown that incremental learning can be used to create an adaptive probe that detects intrusions in a network flow. The network flow data can be analyzed as it flows thanks to the lightness and performance of the TPG framework. A first idea to go forward would be to enable continual learning in the IDS. Continual learning differs from incremental learning (where knowledge is incrementally added to the knowledge base during training) as it enables the exploitation of the model (e.g. triggering alerts) while learning from the network data. Of course, continual learning raises important questions of the action of an attacker on such a model. Indeed, one could try to "normalize" the intrusions traffic by sending requests that seems to be intrusions while being normal. We

showed that the TPG performance could allow to process the network frames as they come, consuming low energy. In order to keep the training and inference performances and to guarantee real-time processing on the network flows, an effort should be made on handling graph complexity. This study could be extended using code generation of the obtained policy in order to speed up the analysis of the network frame while functioning with even less energy. Finally, it would be interesting to implement a framework in which the TPG framework is used as its original Reinforcement Learning (RL) framework design. It would imply the use on a real network of an IDS probe that takes action on the environment such as {allow, alert, block} to handle the network packets. As it would be difficult to use such a system on a live network, an adversarial setup could be thought where an agent tries to intrude the system and the other tries to defend it.

8.3 Journal and conference papers

8.3.1 As the first author:

Journal of Signal Processing Systems, Springer, 2022:

SECURE-GEGELATI Always-On Intrusion Detection through GEGELATI Lightweight Tangled Program Graphs [Sou+22b]

GECCO 2022

Imbalanced Classification with TPG Genetic Programming: Impact of Problem Imbalance and Selection Mechanisms [Sou+22a]

Under Submission

Impact of data biases in Intrusion detection: The benefits of genetic programming

8.3.2 As a co-author:

IEEE Workshop on Signal Processing Systems 2022

Ultra-fast Reinforcement Learning through C Code Generation for TPG Inference [Des+22]

Workshop on Design and Architectures for Signal and Image Processing (14th edition), 2021

GEGELATI: Lightweight artificial intelligence through generic and evolvable tangled program graphs

APPENDIX A

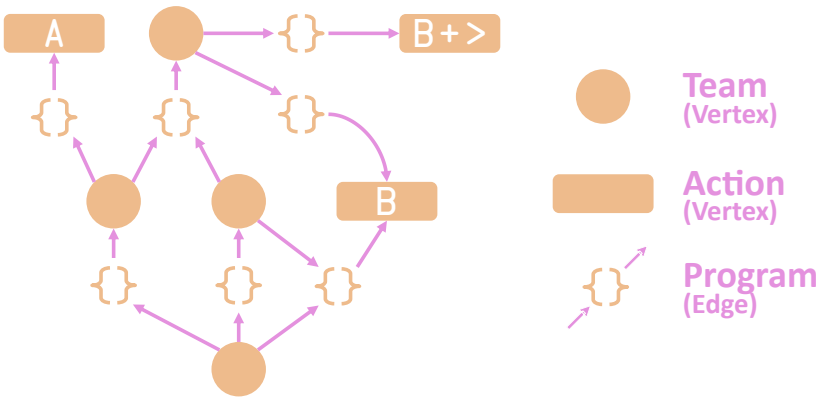
Prototyping and optimization of a Tangled Program Graph framework (GEGELATI)

A.1 Introduction

As an enabler of the work described in this manuscript, a consolidated open-source prototyping work has been made to provide adaptable Tangled Program Graph (TPG) features. Reproducibility of the results was ensured by the implementation of a deterministic behavior for the TPG while the framework has been sped-up using parallelism. A code generation of the best TPG-generated policy is shown to improve the inference performances. In order to train and switch datasets at any time of the training or inference, a saving/restoring of the TPG has also been added to the library. This feature comes with externalized parameterization in *json* files. Finally, as a way to extend the original TPG, program's constants are added and tested into the TPG as well as a inner reward system, in order to enable semi-supervised learning.

This chapter explains the motivation behind these modifications, overviews prototyping decisions and displays the benefits of the methods and their limitations.

The produced library is called Generic Evolvable Graphs for Efficient Learning of Artificial Tangled Intelligence (GEGELATI) and is available on Github [Tec].



TPG example Semantics of the TPG

Table A.1: Recalling the TPG semantics

A.2 The TPG as a deterministic and parallel agent

By determinism, we refer to the property of a program to always produce the exact same output when provided with the same input data and under the same initial state [Lee06]. This study is motivated by the lightweight properties of the TPG agent and have been conduced with K. Desnos [Des+21]. In order to increase both TPG learning and inference capabilities, providing parallel code for efficient learning is a key component. Furthermore, in order to be able to reproduce the obtained results easily on embedded platforms, the determinism of the application was required. It exists a natural conflict between the parallelism and the portability that constituted an implementation challenge to provide both features at a time.

How does the deterministic and scalable parallelism work?
During the learning process of TPGs, the most compute-intensive parts are the fitness evaluation of the *policies*, and the mutations of the *programs* added during the evolution process. The fitness evaluation of individual *policies* can be deterministically executed in parallel, on the conditions that: 1/ the learning environment can be cloned to evaluate several policies concurrently and 2/ any stochastic evolution of the learning environment state can be controlled deterministically. Under these conditions, the parallel evaluation of *policies* is possible, as the topology of the TPG, which is a shared resource for all *policies*, is fixed during this evaluation process. Similarly, the mutation of *programs* can be applied deterministically in parallel. Two kinds of mutations are applied to the TPG: mutations affecting the graph topology by inserting new root *teams* and edges; and mutations affecting *instructions* of the *programs* associated with the new edges. While

mutating the graph topology cannot be done in parallel, the graph being a shared resource, individual *programs* are independent from each other and can be mutated in parallel.

To control the stochastic process of mutation, random number must be generated. Pseudo Random Number Generator (PRNG) must be used each time a random number is needed. Given an initial seed, a PRNG produces a deterministic sequence of numbers. To ensure full determinism of the training of a TPG, a unique PRNG should be called in a fixed order during the whole training. The quality of randomness may impact much TPG performances. One could imagine using TRNGs for TPG inference, at the cost of a loss of determinism. This is kept as future work.

Letting the parallel parts of the training process call the PRNG directly is not possible, as the absolute order in which parallel computations occur is itself stochastic. It is also not possible to give a pre-computed list of pseudo-random numbers to each parallel task, as the number of random numbers needed for each task is itself stochastic. For example, when mutating a *program*, mutations are applied iteratively until the program behavior becomes "original" compared to pre-existing *programs* in the TPG. Hence, giving a fixed number of pre-computed random numbers for the *program* mutations is not feasible.

The parallelization strategy adopted in GEGELATI is based on the master/worker principle, with a distributed PRNG. The principle of the distributed PRNG is the use of two distinct PRNG instances: the $prng_{master}$ and the $prng_{worker}$. The $prng_{master}$ is exclusively used in the sequential parts of the learning process, which confers a deterministic nature to its usage, given an initial seed. Besides being used for stochastic tasks performed sequentially, like TPG topology mutations for example, the $prng_{master}$ is also used to generate a seed for each parallel worker task. In each worker task, a private $prng_{worker}$ is instantiated and initialized with the seed provided by the $prng_{master}$. Since all calls to the PRNG from the worker tasks exclusively use their private $prng_{worker}$, the random number sequences generated in each parallel task are deterministic.

The pseudo-code of the master and worker tasks for the policy fitness evaluation are presented in Procedures EvaluateAllPolicies and Worker, respectively. Communications between the tasks and load balancing of the computations are supported by a job queuing mechanism based on two queues: *JobQ* and *ResultQ*. Each policy evaluation job, prepared by the master procedure, encapsulates a unique job identifier `id`, a seed provided by the $prng_{master}$, and a root *team* from the TPG. All jobs are pushed in the *JobQ* queue before spawning as many worker threads as the number of secondary Processing Elements (PEs) in the target architecture. For each job it acquires from the *jobQ* queue, the worker

Procedure EvaluateAllPolicies

Input: TPG: $G = \langle Teams, Edges \rangle$
Data: PRNG: $prng_{master}$
Job queue: $JobQ$
Result Queue: $ResultQ$

```
/* Prepare jobs                                                    */
idx = 0
foreach root ∈ G.Teams do
    seed = prng_master.getNumber()
    job = { idx++, seed, root }
    jobQ.push(job)
/* Start parallel threads                                          */
for i = 1 to Num_PE − 1 do
    Spawn thread: Worker(G, JobQ, ResultQ)
Call Worker(G, JobQ, resultQ)
Join all threads
/* Post-Process Results and Trace                                  */
Sort ResultQ in result.jobId order
foreach result ∈ resultQ do
    Post-process result.trace // Archiving [KSH]
    ...
```

Procedure Worker

Input: TPG: $G = \langle Teams, Edges \rangle$
Job queue: $JobQ$
Result queue: $ResultQ$
Data: PRNG: $prng_{worker}$
Learning environment twin: LE

```
/* Poll for job                                                    */
while JobQ.hasJob() do
    /* Setup for policy evaluation                                 */
    job = jobQ.getNextJob()
    root = job.root
    prng_worker.reset(job.seed)
    LE.reset(prng_worker.getNumber())
    /* Evaluate policy fitness                                     */
    trace = evaluate(G, root, LE, prng_worker)
    result.jobId = job.id
    result.trace = trace
    resultQ.push(result)
```

procedure resets its $prng_{worker}$ using the seed contained in the job. Before evaluating the fitness of the root *team* contained in a job, the worker procedure resets its private copy of the learning environment, using a number given by the $prng_{worker}$. As a result of the policy fitness evaluation, described in details in [KSH], a result object encapsulating execution traces for the job is pushed in the *resultQ*. When all jobs have been processed, and all workers terminated, the master procedure is responsible for post-processing the traces stored in the *resultQ*. To ensure determinism of this post-processing, results stored in the *resultQ* are first sorted in ascending `job.id` order.

The master and worker procedures used for parallelising the mutations of *programs* are similar to the one used for policy fitness evaluation, with the difference that jobs encapsulate *programs* instead of root *teams*.

A.3 Generating code for fast TPG inference

This study explores the acceleration of the inference of the TPG through the code generation of the best TPG policy.
This feature has been developed by T. Bourgoin [Des+22; BOU+21] as a part of his internship at the IETR lab

A.3.1 Motivations

Parallelism and portability are bound to a consequent overhead during program execution. Furthermore, the convenience of using lambda expressions as instruction causes the slowing down of the execution of the TPG graph. In inferrence mode, the TPG graph can be freezed and transformed into an optimized fixed code. This is the purpose of this following code generation.

A.4 Code Generation for TPG Inference

The C language was selected as the target language for the TPG code generation as it is the de facto reference for programming embedded system. Hence, this choice ensures the portability of generated code on a wide variety of target hardware, ranging for ultra-low power micro-controllers, to high-end CPUs. The following sections describe how the

different parts of a pre-trained TPG from the legacy C++ GEGELATI framework are translated into standard C code.

A.4.1 Code Generation for Programs

In a TPG, a *program* is a list of *instructions* using data from the learning environment or results of previous *instructions* as operands. As presented in [Des+21], training a TPG with different *instruction* sets results in Reinforcement Learning (RL) agents with different complexity and fitness. For this reason, it is important to let developers customize the *instruction* set for each training with dedicated instructions. In the C++ framework, this customization can be supported using lambda functions to specify instructions with their operand number and types, and the lambda function to execute when this instruction is called. Listing A.1 presents two examples of C++ code declaring such custom *instructions* where template arguments define the number and data types of operands accepted by this *instruction*, and the lambda function is the code to execute.

```
auto addInst = LambdaInstruction<int, int>(
    [](int a, int b)->double {return a + b;},
    "$0 = $1 + $2");
auto accuInst = LambdaInstruction<double[2][1]>(
    [](double[2][1] t)->double {
        return t[0][0] + t[1][0];},
    "$0 = $1[0][0] + $2[1][0]")
```

Listing A.1: `LambdaInstruction` usage examples. Two instructions are added (*addInst* is the addition of two variables and *accuInst* defines the accumulation of variables contained in a 2-sized bi-dimensionnal array)

To generate the C code corresponding to a *program*, each *instruction* has to be translated into C code. For this reason, as shown in Listing A.1, a template string is used when declaring the *instruction*. This template string, which adopts the syntax of regular expressions, uses the `$0` placeholder for the register storing the result returned by the instruction, and the `$n` placeholder for the name of the n^{th} operand of the instruction.

```
1 double P0(int* in0    /* 1D array */,
2           double* in1 /* 4x4 2D-array */)
3 {
4   double reg[8] = { 0 };
5   { /* 1st Instruction: addInst */
```

```
6       int op0 = in0[0];
7       int op1 = reg[2];
8       reg[7] = op0 + op1;
9     }
10    { /* 2nd Instruction: accuInst */
11      double[2][1] op0 = {{in1[5]}, {in1[9]}};
12      reg[0] = op0[0][0] + op0[1][0];
13    }
14    ... // Following instructions
15    return reg[0];
16  }
```

Listing A.2: Program `P0()` generated code

For each *program* of the TPG, a dedicated C function is printed, as shown in Listing A.2. At lines 1-2, the printed function receives as arguments the pointers to the data sources used to observe the current state of the environment. At line 4, it declares the registers used to store the results of *instructions* throughout the *program*. Then, at lines 5-14, the *instructions* of the *programs* are printed one by one. For each instruction, the operands are first retrieved from the environment data source, for simple data types, this is achieved through simple pointer de-referencing, as done at lines 6-7. For complex operand types, the container class managing the environment data may provide more complex code generation schemes for fetching the operands, as shown at line 11 where a 2D sub-region of a 2D array of `double` is extracted automatically. Finally, the value held in the first register is returned as the *bid* the *program* at line 15.

A.4.2 Code Generation of TPG Structure

In this work, we choose to represent the traversal of the TPG graph directly in the generated code, exposing the graph structure to the compiler to perform additional optimizations.

We choose to represent the TPG as a Finished State Machine (FSM) using a **switch structure**. An extract of the TPG switch structure is represented on Procedure ExecuteTPG. Each case represents a *team*, containing *program* executions, as *Team1* at line 5. Transitions in the FSM represent the edges of the graph. The traversal of *teams* is saved in a specific array initialized at line 2 to avoid executing *programs* multiple times. Scores are set to $-\infty$ (e.g. line 13) to record edge traversals during graph execution in order to

avoid falling in an infinite loop. The traversal of a leaf *team*, returns the integer value of the corresponding action, as in line 18.

```
   Procedure ExecuteTPG
     Input: Data sources: data
     Output: Action
 1   team = rootTeam
 2   visited[] = { False }
 3   while True do
 4       switch team do
 5           case Team1 do
 6               if !visited[team] then
 7                   visited[team] = True
 8                   T1Scores[0] = P0(data)
 9                   T1Scores[1] = P1(data)
10                   ... // Outgoing programs of the team
11               best = bestProgram(T1Scores)
12               T1Scores[best] = −∞
13               team = T1Next[best]
14           ... // Other teams of the TPG
15           case Action0 do
16               return 1
17           ... // Other actions of the TPG
```

The speedups in inference time of the generated switch-based code with respect to the library are presented in Figure A.1, measured by ratio for the library and the generated code, of the total time spent executing the TPG for a complete game of the ALE framework [Bel+13]. On average on all games, the observed speedups are 44× on `xeon`, 24× on `laptop`, 45× on `jetson`, and 85× on `rpi2`. There are many possible causes to these differences in average speedup between platforms, notably: different hardware complexity (pipeline depth, bit-width, instruction & data cache sizes), different compiler versions, etc. Nevertheless, the obtained results are very good, especially for the `rpi2` which is the most lightweight CPU and benefits the most from the acceleration brought by the code generation.

Interestingly, the per-game variations of the speedups observed on every platform are very similar. For example, speedups obtained on all platforms for the `asteroids` game are on average 56% larger than speedups for the `fishing_derby` game. These results seems

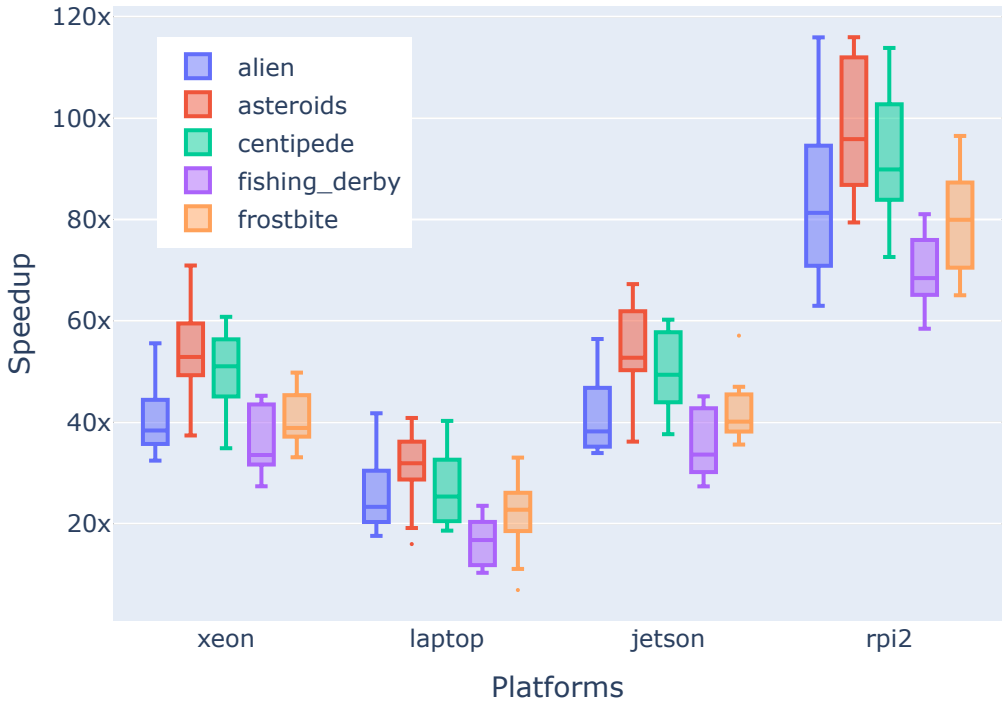


Figure A.1: Speedup in inference time of the generated code with respect to the library. Each box-plot represents the statistics for the 10 TPG trained for a given game, and inferred on a specific platform.

to indicate that the measured speedup per-TPG depends on the intrinsic complexity of the TPG itself, which derives from its number of *teams* or *programs*.

The average times per inference of the TPG, that is per action, for the different platforms are presented in Figure A.2. These results show the impressive absolute performance of the generated code, which on average performs an inference of the TPG within 782ns on `xeon`, 1.36µs on `laptop`, 2.41µs on `jetson`, and 8.60µs on `rpi2`. While these result confirm the benefit of using generated code for inference TPG, they also reveal the important spread of inference time on most platforms, with an average relative standard deviation of 34% for generated code (excluding `laptop`), and 42% for inference within the library.

Figure A.3 depicts the average execution time taken per line of *programs* of the TPG on the different platforms. With the library, these results show that the inference time of a TPG strongly correlates with the number of lines of *program* to execute, with a relative standard deviation of only 12%.

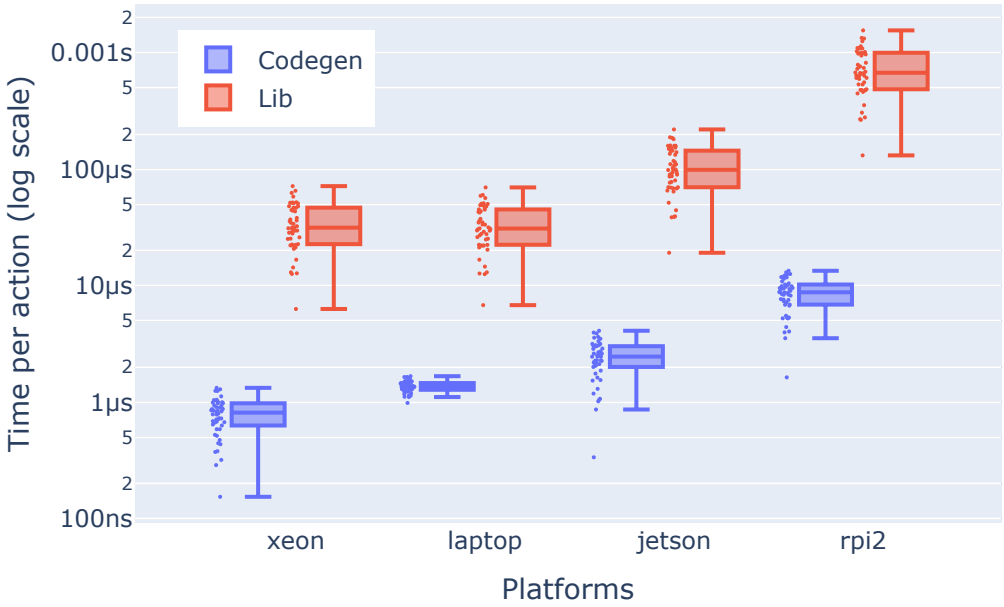


Figure A.2: Average time per TPG inference with the generated code and with the library. Each box-plot represents the statistics for the 50 TPG (5 games×10 TPG) run on a platform.

A.4.3 Discussion on the prototyping work

Acceleration of the TPG inference is achieved by getting rid of the algorithmic and software overhead needed for training TPGs, but dependable when focusing on TPG inference. Experiments on four computing platforms, ranging from embedded processors to high-end CPUs, result in a global acceleration by a factor 50 on average, of the inference time, compared to a traditional TPG framework. For a state-of-the-art visual RL environment and for performance equivalent to Deep Neural Network (DNN), the obtained inference time, range from hundreds of nano-seconds to a few micro-seconds on single-core CPUs, making this approach a very promising one for embedding RL agent in ultra-low power edge Artificial Intelligence (AI) systems.

A.5 Saving and restoring TPGs

This implementation aims at producing an output file that describes the TPG and that can be used as input for the GEGELATI framework to restore a graph to a stored state. This feature was developed with K. Desnos.

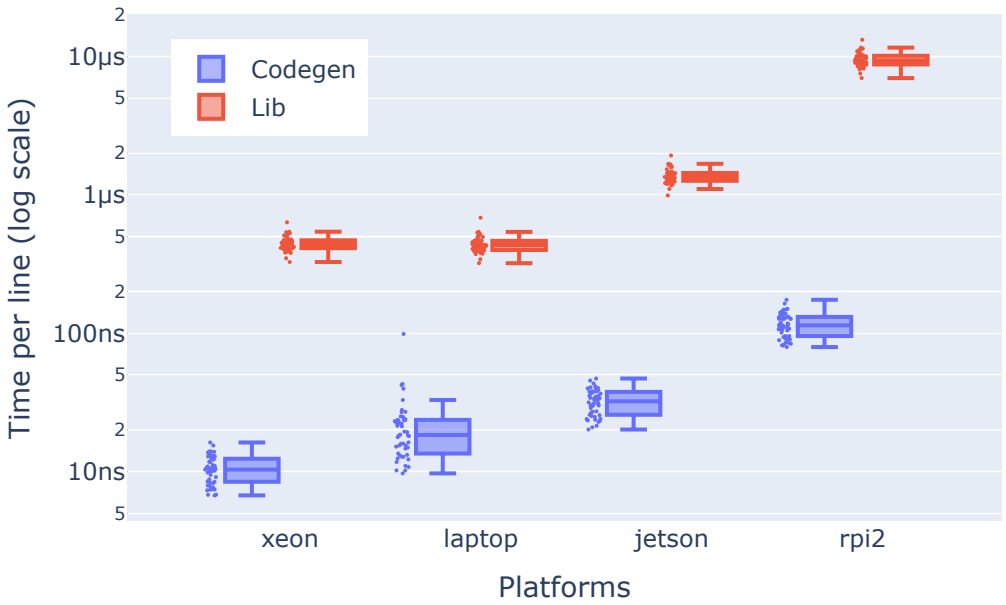


Figure A.3: Average time per instruction execution with the generated code and with the library. Each box-plot represents the statistics for the 50 TPG (5 games×10 TPG) run on a platform.

A.5.1 Motivations

On the one hand, saving a TPG graph at a given generation permits the exploration of its structure and offline extraction of basic statistics such as the number of teams, programs or information on what instructions are used. The humanly readable *.dot* format is used and can help generate TPG graph visualization. On the other hand, restoring a TPG graph is interesting when it comes to problems such as porting of a learned Anomaly-based Intrusion Detection System (AIDS) on a live network, the use of another dataset, or the deployment of a learned TPG graph on an embedded platform.

A.5.2 Prototyping choices for storing a TPG

Saving a TPG graph is rather simple. At the end of a generation, a graph is exported following the *.dot* file format. The algorithm iterates on the {teams, programs, instructions, action} and saves the components of each of them under text format tagged with specific indexes:

- T: Team – save the outgoing edges
- P: Program – save the list of instructions and the destination

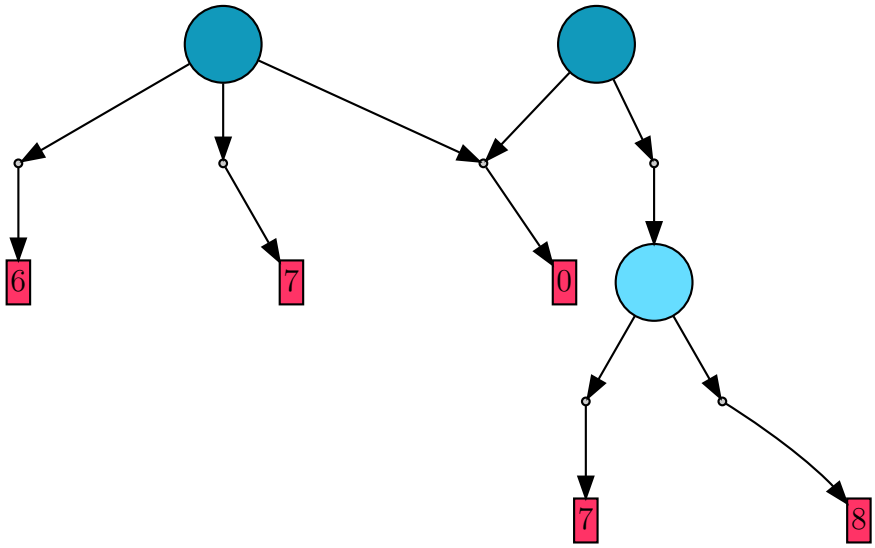


Figure A.4: A simple graph visualization of an exported TPG graphs, having two root teams, one team, six programs and four actions (action seven is duplicated for readability).

- I: Instructions – save the instruction and its operands
- A: Action – save the action

The rest of the tag is made of a number. Team's tags never change with the generations, implying that when a team is deleted, its tag is never used again.

The export of the graph follows the procedure ExportGraph.

Procedure ExportGraph

Input: TPG_Graph
Output: tpg.dot file

```
1 Print(TPG_DOT_HEADER) foreach team in TPG_Graph do
2     Print(team)
3 resetProgramTags() foreach program in TPG_Graph do
4     sourceTag = getSourceTag(program) Print(program) programTag =
        getProgramTag() instruction = getProgramInstructions() instructionTag =
        getInstructionTag Print(instruction) Print(Link(programTag,
        InstructionTag)) destination = getDestinationTag(program)
        Print(Link(sourceTag, ProgramTag, destinationTag))
5 Print(TPG_DOT_FOOTER)
```

The *.dot* format enables simple graph visualization using the **Graphviz** tool (see Figure A.4).

Restoring a TPG graph from a *.dot* file is the exact inverse procedure.

A.5.3 Discussion

This feature is a minor contribution to the TPG framework but is useful in many experiments of this manuscript. Furthermore it possible to compute statistics from the graph topology and a consequent save of time when using one trained model to infer on one or several other datasets. Furthermore, for reproducing the results, a single *.dot* file can be saved, saving the integrity of the graphs component.

A.6 External parametizing of the TPG

As a continuation of efforts to maintain TPG frameworks reproducibility, having external parametizing of the application is convenient. This feature was developed with P.Y. Raumer [Rau+20] From one part, it makes the modification of the behavior of the GEGELATI library possible, without the need of rebuilding it when the parameters were changed. Furthermore, it provides simple parameter exports that can be stored along with the resulting *.dot* file, saving both the TPG graph structure and information and its parameters.

A.6.1 Parameterized TPG implementation

The parameters of the TPG are stored in a JSON file. At initialization, a parser is called to read the JSON file and fill class information with the parameters it contains. The parameters are then stored in the TPG environment. Default values of each parameter are set in the JSON file to prevent errors and missing parameters values.

A.7 Mimicking decision trees and Convolutional Neural Network (CNN) with TPGs

This work provides constants, that belongs to a TPG program, and that can be used during both training and inference.

A.7.1 Motivations

The motivation behind this work is to be able to mimic the behavior of a decision tree by using constants and instructions such as {<,>,=} to compare the currently observed

state of the environment to one of the defined program's constant. Several constants can be used by the program to execute instructions such as a convolution, so as to mimic the behavior of a CNN. The implementation of constants in the TPG allows any instructions that require the use of constants. Common examples are comparison operators, filters and experts functions. For example in the intrusion detection problem, one can design an expert function to check which port is used for a connection. This function can rely on the use of constants and each program, through mutation could develop specific interests on listening connections on specific ports.

A.7.2 Implementation of TPG constants

The implementation of the TPG constants aims at providing, as part of the TPG environment a set of values that are fixed for a defined program. In the GEGELATI framework, constants are only integer values. At the initialization of the TPG, three new parameters are defined:

- $NbProgramConstants$: the number of Constants
- $MinConstValue$: the low bound of the constant value
- $MaxConstValue$: the high bound of the constant value

At initialization, $NbProgramConstants$ are randomly selected between their lowest and highest bound for each program. These constants can be used by the program as a part of the environment during all the execution of the TPG given that at least one instruction enables their use. At the end of a generation, programs subject to mutation have a chance of having one or several constants edited.

A.7.3 Discussion

The use of constants in a TPG is very use-case dependent. The number of constants and their range of values depend on both the use case and the instruction set used. Constants used in intrusion detection need to differ from constants used for image classification. For example, a 3×3 convolution filter applied to an image would require nine parameters and could not be applied on a single network flow log. The impact of the use of constants on the learning rate of the TPG has not yet been quantified. Preliminary results showed positive impact of the use of constants for long-run TPGs facing image classification problems.

A.8 Towards a semi-supervised TPG ?

This implementation work has been thought in order to lower the required supervision of the TPG.

A.8.1 Motivation

Intrusion detection is bound to the issue of having representative and labeled data. In this manuscript, we focus on the use of the TPG, as a supervised learning classifier, to detect intrusion in the network data. When performing continual learning on a live network, some feedback information are available from other defense mechanisms (Such as intrusions alerts from other Intrusion Detection System (IDS), known allowed traffic from the firewalls, ...). Some of this traffic is unfortunately unqualified and requires to be analyzed. The continual learning AIDS can help in its classification but can not, yet, learn from its inner reaction to the observation of the unknown observed network data.

A.8.2 Prototyping a semi-supervised learning based TPG

In order to build a TPG-based framework that provides semi-supervised learning, different fitness information should be used as a reward.

Common classifiers use labels to compute a fitness for the evaluation of a batch of data and this fitness is used in the core of the TPG to select the best performing teams.

After online training, the structure of the TPG graph is evaluated and validated on a data batch.

This prototyping work is based on two hypotheses:

1. It exists one or several correlation between samples from the same classes.

2. The chosen TPG path is similar for samples that are similar.

We use the confidence of the graph itself to introduce an inner reward mechanism. This inner reward mechanism R_i is added with a coefficient λ to the global fitness measurement of the team such as in equation A.1.

$$Fitness = Fitness(Batch + \lambda R_i) \tag{A.1}$$

The basic functioning of the TPG relies on the use of teams. Teams are associated with one or several programs (edges) that compute instructions and give a single bid at

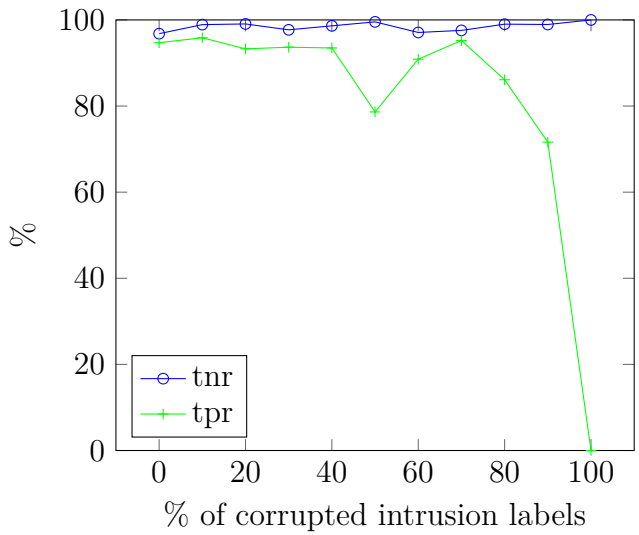


Figure A.5: TPR and TNR of the semi-supervised learning version of the TPG depending on the amount of missing labels in the dataset. These preliminary results show that the method seems to be resilient to up to 70% of missing labels.

the end of their execution. The program with the highest bid is selected and leads the path until its destination (either another team, having the same behavior or an action).

Programs observe specific parts of the environment and transform them, using instructions, into a bid. This implementation relies on the fact that a program behaves similarly when observing a similar data. Programs are added a target value for their bid which is calculated during the first execution of the program. The Mean Square Error (MSE) is computed using the programs target and the bid. If the MSE value is inferior to a defined threshold t, the program gets an additional inner reward. The extra points earned by the programs are added to the graph R_i.

At the end of the generation, the fitness of each graph is computed, and the selection process occurs as before.

Adaptations are made in the TPG instruction set to have real values at any time in the graph. More specifically, the instructions that can lead to infinity or Not a Number (NaN) such as {*, /, exp, log} have been removed.

A.8.3 Discussion

The first interesting result is that the modification of TPG, fully supervised, does not compromise the learning, validating the hypothesis that for similar data input, the same

programs activate. This is true for both fully supervised intrusion detection and MNIST image classification. The semi-supervised TPG framework results for a RL task of pole stabilization seems to be degraded by the modification. This is explained by the lack of the expected correlation between a class and its label in classification environments.

Preliminary results show that the TPG-based semi-supervised learning framework is able to learn with up to 70% of missing labels (see Figure A.5).

Although no comparison is yet available with the legacy TPG, research on the specialization of the semi-supervised TPG agent is ongoing as we believe that continual learning on live network is promising for the intrusion detection domain. An additional experimental work is ongoing in order to determine suiting parameter value for t and λ.

A parameterization of the thresholds is required, and the measurement of the impact of the proportion of the missing labels is ongoing.

A.9 Conclusion

This Appendix sums up the prototyping work conducted during this thesis. In particular, it helped to provide a parallel and deterministic TPG framework to enhance the performances of the TPG learning algorithms. The TPG inference can be sped up an average of 50 times using efficient code generation. Fast external parametizing is accessible through JSON file parsing and graphs can be saved and restored using .dot files that allow convenient graph visualization. Programs can be associated with constants in order to perform novel operations such as filters and convolutions. Finally, a prototype of a semi-supervised TPG framework has been developed in order to enhance the performances of intrusion detection on live IP networks. Although the prototyping work described in this chapter does not constitute a major contribution of this manuscript, it is at the ground of the other contributions presented in this manuscript and extend the future work around TPG and its capabilities in intrusion detection. Most of this prototyping work ensure the reproducibility of the obtained results, convenient in the use of the GEGELATI library and performance improvement.

List of Figures

List of Tables

Listings

Acronyms

AI Artificial Intelligence

AIDS Anomaly-based Intrusion Detection System

ALE Arcade Learning Environment

AUC Area Under Curve

CNN Convolutional Neural Network

DMZ Demilitarized Zone

DNN Deep Neural Network

DoS Denial of Service

DQN Deep Q-Networks

FPR False Positive Rate

FSM Finished State Machine

GDRP General Data Protection Regulation

GEGELATI Generic Evolvable Graphs for Efficient Learning of Artificial Tangled Intelligence

GP Genetic Programming

IDS Intrusion Detection System

IOM Imbalance Order of Magnitude

IP Internet Protocol

IPS Intrusion Protection System

IS Information System

LA Learning Agent

LAN Local Area Network

MARL Multi-Agent Reinforcement Learning

MCC Matthew's Correlation Coefficient

ML Machine Learning

MSE Mean Square Error

NaN Not a Number

NIDS Network Intrusion Detection Systems

NN Neural Network

PE Processing Element

PRNG Pseudo Random Number Generator

RF Random Forest

RL Reinforcement Learning

SIDS Signature-based Intrusion Detection Systems

SMOTE Synthetic Minority Oversampling Technique

SVD Singular Value Decomposition

SVM Support Vector Machine

TNR True Negative Rate

TPG Tangled Program Graph

TPR True Positive Rate

WAN Wide Area Network

Bibliography

[Abb+14] Ali Abbasi et al., "On emulation-based network intrusion detection systems", *in*: *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2014 (cit. on p. 47).

[AI19] Mohammed Hamid Abdulraheem and Najla Badie Ibraheem, "A detailed analysis of new intrusion detection dataset", *in*: *Journal of Theoretical and Applied Information Technology* (2019) (cit. on pp. 69, 70).

[AL15] Michael J Assante and Robert M Lee, "The industrial control system cyber kill chain", *in*: *SANS Institute InfoSec Reading Room* (2015) (cit. on p. 40).

[AMH16] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu, "A survey of network anomaly detection techniques", *in*: *Journal of Network and Computer Applications* (2016) (cit. on p. 46).

[APN18] Tamer Aldwairi, Dilina Perera, and Mark A Novotny, "An evaluation of the performance of Restricted Boltzmann Machines as a model for anomaly network intrusion detection", *in*: *Computer Networks* (2018) (cit. on pp. 116, 118, 137).

[ASS19] Simon D. Duque Anton, Sapna Sinha, and Hans Dieter Schotten, "Anomaly-based Intrusion Detection in Industrial Data with SVM and Random Forests", *in*: *arXiv:1907.10374 [cs]* (2019) (cit. on pp. 46, 47).

[ASS20] Hadeel Alazzam, Ahmad Sharieh, and Khair Eddin Sabri, “A feature selection algorithm for intrusion detection system based on pigeon inspired optimizer”, *in*: *Expert systems with applications* (2020) (cit. on p. 71).

[Aur+19] Yuri Sousa Aurelio et al., “Learning from imbalanced data sets with weighted cross-entropy function”, *in*: *Neural processing letters* (2019) (cit. on p. 91).

[Bar+18] Matthew P Barrett et al., “Framework for improving critical infrastructure cybersecurity”, *in*: *National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep* (2018) (cit. on pp. 14, 20, 48).

[BB17] Solon Barocas and Danah Boyd, “Engaging the ethics of data science in practice”, *in*: *Communications of the ACM* (2017) (cit. on p. 69).

[BD12] Leyla Bilge and Tudor Dumitraş, “Before we knew it: an empirical study of zero-day attacks in the real world”, *in*: *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012 (cit. on p. 41).

[Bel+13] Marc G Bellemare et al., “The arcade learning environment: An evaluation platform for general agents”, *in*: *Journal of Artificial Intelligence Research* (2013) (cit. on pp. 62, 156).

[BMS13] Ismail Butun, Salvatore D Morgera, and Ravi Sankar, “A survey of intrusion detection systems in wireless sensor networks”, *in*: *IEEE communications surveys & tutorials* (2013) (cit. on p. 47).

[Bon+18] Justine Bonnot et al., “CASSIS: Characterization with adaptive sample-size inferential statistics applied to inexact circuits”, *in*: *2018 26th European Signal Processing Conference (EUSIPCO)*, IEEE, 2018 (cit. on p. 100).

[BOU+21] Thomas BOURGOIN et al., “Génération de code pour une bibliotheque d’apprentissage par renforcement”, *in*: (2021) (cit. on p. 153).

[Bre01] Leo Breiman, “Random forests”, *in*: *Machine learning* (2001) (cit. on p. 52).

[Bri+18] Pierre-Olivier Brissaud et al., “Passive monitoring of https service use”, *in*: *2018 14th International Conference on Network and Service Management (CNSM)*, IEEE, 2018 (cit. on p. 23).

[BRK21] Samuel Budd, Emma C Robinson, and Bernhard Kainz, “A survey on active learning and human-in-the-loop deep learning for medical image analysis”, *in*: *Medical Image Analysis* (2021) (cit. on p. 144).

[CA16] M Emre Celebi and Kemal Aydin, *Unsupervised learning algorithms*, Springer, 2016 (cit. on pp. 22, 53).

[CB05] Ramkumar Chinchani and Eric van den Berg, "A fast static analysis approach to detect exploit code inside network flows", *in*: *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2005 (cit. on p. 34).

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar, "Anomaly detection: A survey", *in*: *ACM computing surveys (CSUR)* (2009) (cit. on p. 89).

[CC] Communications Security Establishment (CSE) and Canadian Institute for Cybersecurity (CIC), *Intrusion Detection Evaluation Dataset (CSE-CIC-IDS2018)*, URL: https://www.unb.ca/cic/datasets/ids-2018.html (cit. on p. 114).

[CDP14] Dan Craigen, Nadia Diakun-Thibault, and Randy Purse, "Defining cybersecurity", *in*: *Technology Innovation Management Review* (2014) (cit. on p. 37).

[CDS19] Vinod Kumar Chauhan, Kalpana Dahiya, and Anuj Sharma, "Problem formulations and solvers in linear SVM: a review", *in*: *Artificial Intelligence Review* (2019) (cit. on p. 50).

[Cha+02] Nitesh V Chawla et al., "SMOTE: synthetic minority over-sampling technique", *in*: *Journal of artificial intelligence research* (2002) (cit. on p. 91).

[Che+21] Jinfu Chen et al., "An Efficient Network Intrusion Detection Model Based on Temporal Convolutional Networks", *in*: *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, IEEE, 2021 (cit. on p. 70).

[Con+19] Christos Constantinides et al., "A novel online incremental learning intrusion prevention system", *in*: *2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, IEEE, 2019 (cit. on pp. 116–118, 137).

[CP77] Robin LP Chang and Theodosios Pavlidis, "Fuzzy decision tree algorithms", *in*: *IEEE Transactions on systems, Man, and cybernetics* (1977) (cit. on p. 51).

[CV95] Corinna Cortes and Vladimir Vapnik, "Support-vector networks", *in*: *Machine learning* (1995) (cit. on p. 50).

[Cyb] Canadian Institute of Cyber Security, *Intrusion Detection Evaluation Dataset (CIC-IDS2017)*, URL: https://www.unb.ca/cic/datasets/ids-2017.html (cit. on pp. 75, 114).

[DA20] Padmanabhan Deepak and Savitha Sam Abraham, "Fair Outlier Detection.", *in*: *WISE (2)*, 2020 (cit. on p. 71).

[DDW00] Hervé Debar, Marc Dacier, and Andreas Wespi, "A revised taxonomy for intrusion-detection systems", *in*: *Annales Des Télécommunications* (2000) (cit. on p. 41).

[Den12] Li Deng, "The mnist database of handwritten digit images for machine learning research", *in*: *IEEE Signal Processing Magazine* (2012) (cit. on p. 102).

[Den87] Dorothy E Denning, "An Intrusion-Detection Model", *in*: *IEEE Transactions on Software Engineering* (1987) (cit. on pp. 14, 21, 31).

[Des+21] Karol Desnos et al., "Gegelati: Lightweight artificial intelligence through generic and evolvable tangled program graphs", *in*: *Workshop on Design and Architectures for Signal and Image Processing (14th edition)*, 2021 (cit. on pp. 150, 154).

[Des+22] Karol Desnos et al., "Ultra-fast Reinforcement Learning through C Code Generation for tpg Inference", *in*: *IEEE Workshop on Signal Processing Systems*, IEEE, 2022 (cit. on pp. 146, 153).

[DH08] John Doucette and Malcolm I Heywood, "GP classification under imbalanced data sets: Active sub-sampling and AUC approximation", *in*: *European Conference on Genetic Programming*, Springer, 2008 (cit. on p. 91).

[DI18] Cynthia Dwork and Christina Ilvento, "Fairness under composition", *in*: *arXiv preprint arXiv:1806.06122* (2018) (cit. on p. 70).

[dOL17] Brian d'Alessandro, Cathy O'Neil, and Tom LaGatta, "Conscientious classification: A data scientist's guide to discrimination-aware classification", *in*: *Big data* (2017) (cit. on p. 69).

[Don+19] Gangsong Dong et al., "DB-Kmeans:An Intrusion Detection Algorithm Based on DBSCAN and K-means", *in*: *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2019) (cit. on p. 46).

[DR20] Ian Davidson and Selvan Suntiha Ravi, "A framework for determining the fairness of outlier detection", *in*: *ECAI 2020*, IOS Press, 2020 (cit. on p. 71).

[ERJ21] Gints Engelen, Vera Rimmer, and Wouter Joosen, "Troubleshooting an intrusion detection dataset: the CICIDS2017 case study", *in*: *2021 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2021 (cit. on pp. 16, 24, 58, 69, 72, 73, 76, 83, 124, 136).

[Esc98] Terry Escamilla, *Intrusion detection: network security beyond the firewall*, John Wiley, 1998 (cit. on p. 47).

[Fer+18] Alberto Fernández et al., "SMOTE for learning from imbalanced data: progress and challenges, marking the 15-year anniversary", *in*: *Journal of artificial intelligence research* (2018) (cit. on p. 91).

[FN96] Batya Friedman and Helen Nissenbaum, "Bias in computer systems", *in*: *ACM Transactions on Information Systems (TOIS)* (1996) (cit. on p. 69).

[Gar+09] Pedro Garcia-Teodoro et al., "Anomaly-based network intrusion detection: Techniques, systems and challenges", *in*: *computers & security* (2009) (cit. on pp. 13, 19, 46).

[GD95] Diana F Gordon and Marie Desjardins, "Evaluation and selection of biases in machine learning", *in*: *Machine learning* (1995) (cit. on pp. 16, 23, 67).

[GHF22] Benjamin van Giffen, Dennis Herhausen, and Tobias Fahse, "Overcoming the pitfalls and perils of algorithms: A classification of machine learning biases and mitigation methods", *in*: *Journal of Business Research* (2022) (cit. on pp. 23, 68).

[GJM14] TJ Grant, RHP Janssen, and Herman Monsuur, *Network Topology in Command and Control: Organization, Operation, and Evolution*, Information Science Reference, 2014 (cit. on p. 32).

[GMS00] Anup K Ghosh, Christoph Michael, and Michael Schatz, "A real-time intrusion detection system based on learning program behavior", *in*: *International Workshop on Recent Advances in Intrusion Detection*, Springer, 2000 (cit. on pp. 116, 118, 137).

[Gon+19] Sergio González et al., "Chain based sampling for monotonic imbalanced classification", *in*: *Information Sciences* (2019) (cit. on p. 92).

[GS08] Meera Gandhi and SK Srivasta, *Detecting and preventing attacks using network intrusion detection systems*, 2008 (cit. on pp. 41, 42).

[Gu+06] Guofei Gu et al., “Measuring intrusion detection capability: an information-theoretic approach.”, *in*: 2006 (cit. on pp. 126, 127).

[Har07] Anne-Wil Harzing, *Publish or Perish*, https://harzing.com/resources/publish-or-perish, 2007 (cit. on p. 21).

[HCA+11] Eric M Hutchins, Michael J Cloppert, Rohan M Amin, et al., “Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains”, *in*: *Leading Issues in Information Warfare & Security Research* (2011) (cit. on p. 39).

[Hea+98] Marti A. Hearst et al., “Support vector machines”, *in*: *IEEE Intelligent Systems and their applications* (1998) (cit. on p. 50).

[HG09] Haibo He and Edwardo A Garcia, “Learning from imbalanced data”, *in*: *IEEE Transactions on knowledge and data engineering* (2009) (cit. on pp. 15, 21, 24, 58).

[HH05] Simon Hansman and Ray Hunt, “A taxonomy of network and computer attacks”, *in*: *Computers & Security* (2005) (cit. on pp. 13, 19, 38).

[Hol14] Hannes Holm, “Signature based intrusion detection for zero-day attacks:(not) a closed chapter?”, *in*: *2014 47th Hawaii international conference on system sciences*, IEEE, 2014 (cit. on p. 42).

[Hol92] John H Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, MIT press, 1992 (cit. on p. 55).

[HS14] Neminath Hubballi and Vinoth Suryanarayanan, “False alarm minimization techniques in signature-based intrusion detection systems: A survey”, *in*: *Computer Communications* (2014) (cit. on pp. 13, 19, 41, 42).

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, “Overview of supervised learning”, *in*: *The elements of statistical learning*, Springer, 2009 (cit. on pp. 22, 50).

[JCD13] László A Jeni, Jeffrey F Cohn, and Fernando De La Torre, “Facing imbalanced data–recommendations for the use of performance metrics”, *in*: *2013 Humaine association conference on affective computing and intelligent interaction*, IEEE, 2013 (cit. on pp. 17, 25, 91).

[Jos+16] Matthew Joseph et al., “Fairness in learning: Classic and contextual bandits”, *in*: *Advances in neural information processing systems* (2016) (cit. on p. 70).

[JPP11] VVRPV Jyothsna, Rama Prasad, and K Munivara Prasad, “A review of anomaly based intrusion detection systems”, *in*: *International Journal of Computer Applications* (2011) (cit. on pp. 13, 19, 46).

[KA17] Kwangjo Kim and Muhamad Erza Aminanto, “Deep learning in intrusion detection perspective: Overview and further challenges”, *in*: *2017 International Workshop on Big Data and Information Security (IWBIS)*, IEEE, 2017, pp. 5–10 (cit. on pp. 13, 19).

[Kel18] Stephen Kelly, “Scaling genetic programming to challenging reinforcement tasks through emergent modularity”, *in*: (2018) (cit. on pp. 62, 63, 76, 102, 126).

[KH17a] Stephen Kelly and Malcolm I Heywood, “Emergent tangled graph representations for Atari game playing agents”, *in*: *EuroGP*, Springer, 2017 (cit. on p. 62).

[KH17b] Stephen Kelly and Malcolm I Heywood, “Multi-task learning in atari video games with emergent tangled program graphs”, *in*: *Proceedings of the Genetic and Evolutionary Computation Conference*, 2017 (cit. on pp. 14, 17, 19, 25, 62, 89, 115, 136).

[Khr+19] Ansam Khraisat et al., “Survey of intrusion detection systems: techniques, datasets and challenges”, *in*: *Cybersecurity* (2019) (cit. on pp. 41, 47).

[KLM96] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore, “Reinforcement learning: A survey”, *in*: *Journal of artificial intelligence research* (1996) (cit. on pp. 22, 53).

[KSH] Stephen Kelly, Robert J Smith, and Malcolm I Heywood, “Emergent policy discovery for visual reinforcement learning through tangled program graphs: A tutorial”, *in*: *Genetic programming theory and practice XVI* () (cit. on pp. 60, 62, 75, 114, 152, 153).

[KT03] Christopher Kruegel and Thomas Toth, “Using Decision Trees to Improve Signature-Based Intrusion Detection”, *in*: Lecture Notes in Computer Science (2003), ed. by Giovanni Vigna, Christopher Kruegel, and Erland Jonsson (cit. on pp. 13, 19, 42).

[KV02] Richard A Kemmerer and Giovanni Vigna, "Intrusion detection: a brief history and overview", *in*: *Computer* 35.*4* (2002), supl27–supl30 (cit. on pp. 13, 19).

[KZH05] H Günes Kayacik, A Nur Zincir-Heywood, and Malcolm I Heywood, "Selecting features for intrusion detection: A feature relevance analysis on KDD 99 intrusion detection datasets", *in*: *Proceedings of the third annual conference on privacy, security and trust*, Citeseer, 2005 (cit. on p. 71).

[Lab15] Kaspersky Lab, *Damage control: the cost of security breaches*, `https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf`, 2015 (cit. on p. 41).

[Lan+10] Jyhshyan Lan et al., "An investigation of neural network classifiers with unequal misclassification costs and group sizes", *in*: *Decision Support Systems* (2010) (cit. on p. 69).

[LAR13] ROGER LARSEN, *BRO-an Intrusion Detection System*, 2013 (cit. on p. 35).

[Las+17] Arash Habibi Lashkari et al., *CICFlowMeter*, 2017 (cit. on p. 35).

[LB+95] Yann LeCun, Yoshua Bengio, et al., "Convolutional networks for images, speech, and time series", *in*: *The handbook of brain theory and neural networks* (1995) (cit. on p. 52).

[LBX20] Yi Liu, Will N. Browne, and Bing Xue, "Absumption and subsumption based learning classifier systems", *in*: *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, July 8-12, 2020*, ACM, 2020 (cit. on p. 92).

[LCS20] Manuel Lopez-Martin, Belen Carro, and Antonio Sanchez-Esguevillas, "Application of deep reinforcement learning to intrusion detection for supervised problems", *in*: *Expert Systems with Applications* (2020) (cit. on pp. 116, 118, 137).

[Le+21] Hoang Lam Le et al., "EUSC: A clustering-based surrogate model to accelerate evolutionary undersampling in imbalanced classification", *in*: *Applied Soft Computing* (2021) (cit. on p. 92).

[Lee+18] Joffrey L Leevy et al., "A survey on addressing high-class imbalance in big data", *in*: *Journal of Big Data* (2018) (cit. on p. 90).

[Lee06] Edward A Lee, "The problem with threads", *in*: *Computer* (2006) (cit. on p. 150).

[Lev16] Jake Lever, "Classification evaluation: It is important to understand both what a classification metric expresses and what it hides", *in*: *Nature methods* (2016) (cit. on pp. 15, 21, 56).

[Lia+13] Hung-Jen Liao et al., "Intrusion detection system: A comprehensive review", *in*: *Journal of Network and Computer Applications* (2013) (cit. on p. 41).

[Liu+19] Jinping Liu et al., "ANID-SEoKELM: Adaptive network intrusion detection based on selective ensemble of kernel ELMs with random features", *in*: *Knowledge-based systems* (2019) (cit. on pp. 116, 118, 137).

[Liu+20] Jinping Liu et al., "Adaptive intrusion detection via GA-GOGMM-based pattern learning with fuzzy rough set-based attribute selection", *in*: *Expert Systems with Applications* (2020) (cit. on pp. 116, 118, 135, 137).

[Lóp+13] Victoria López et al., "A hierarchical genetic fuzzy system based on genetic programming for addressing classification with highly imbalanced and borderline data-sets", *in*: *Knowledge-Based Systems* (2013) (cit. on p. 91).

[LP18] Camelia Lemnaru and Rodica Potolea, "Evolutionary cost-sensitive balancing: A generic method for imbalanced classification problems", *in*: *EVOLVE-A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation VI*, Springer, 2018 (cit. on p. 93).

[Mac+67] James MacQueen et al., "Some methods for classification and analysis of multivariate observations", *in*: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Oakland, CA, USA, 1967 (cit. on p. 53).

[McH00] John McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory", *in*: *ACM Transactions on Information and System Security (TISSEC)* (2000) (cit. on p. 69).

[Meh+21] Ninareh Mehrabi et al., "A survey on bias and fairness in machine learning", *in*: *ACM Computing Surveys (CSUR)* (2021) (cit. on pp. 23, 68, 69).

[MG09] Luis Mena and Jesus A Gonzalez, "Symbolic one-class learning from imbalanced datasets: application in medical diagnosis", *in*: *International Journal on Artificial Intelligence Tools* (2009) (cit. on p. 69).

[MIT99] Massachusetts Institute of Technology MIT Lincoln Laboratory, *1999 DARPA Intrusion Detection Evaluation Dataset*, https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset, 1999 (cit. on p. 57).

[MK18] Soosan Naderi Mighan and Mohsen Kahani, "Deep learning based latent feature extraction for intrusion detection", *in*: *Electrical Engineering (ICEE), Iranian Conference on*, IEEE, 2018 (cit. on p. 71).

[MLK14] Weizhi Meng, Wenjuan Li, and Lam-For Kwok, "EFM: enhancing the performance of signature-based network intrusion detection systems using enhanced filter mechanism", *in*: *computers & security* (2014) (cit. on pp. 68, 70, 71).

[Mni+15] Volodymyr Mnih et al., "Human-level control through deep reinforcement learning", *in*: *nature* (2015) (cit. on p. 54).

[MNN21] Behzad Mirzaei, Bahareh Nikpour, and Hossein Nezamabadi-pour, "CDBH: A clustering and density-based hybrid approach for imbalanced data classification", *in*: *Expert Systems with Applications* (2021) (cit. on p. 92).

[Moh+19] Marwa R Mohamed et al., "Exploiting Incremental Classifiers for the Training of an Adaptive Intrusion Detection Model.", *in*: *Int. J. Netw. Secur.* (2019) (cit. on pp. 116–118, 137).

[Mor11] Benoit Morel, "Artificial intelligence and the future of cybersecurity", *in*: *Proceedings of the ACM Conference on Computer and Communications Security* (2011) (cit. on pp. 17, 25, 31, 113).

[MP17] Savita Mohurle and Manisha Patil, "A brief study of wannacry threat: Ransomware attack 2017", *in*: *International Journal of Advanced Research in Computer Science* (2017) (cit. on p. 41).

[MP43] Warren S McCulloch and Walter Pitts, "A logical calculus of the ideas immanent in nervous activity", *in*: *The bulletin of mathematical biophysics* (1943) (cit. on p. 52).

[MPB14] Ioan-Cosmin Mihai, Stefan Pruna, and Ionut-Daniel Barbu, "Cyber kill chain analysis", *in*: *Int'l J. Info. Sec. & Cybercrime* (2014) (cit. on p. 40).

[MPV21] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining, *Introduction to linear regression analysis*, John Wiley & Sons, 2021 (cit. on p. 50).

[NN98] James R Norris and James Robert Norris, *Markov chains*, Cambridge university press, 1998 (cit. on p. 54).

[OC20] Luca Oneto and Silvia Chiappa, "Fairness in machine learning", *in*: *Recent Trends in Learning From Data*, Springer, 2020 (cit. on p. 70).

[OHT20] Yassine Ouali, Céline Hudelot, and Myriam Tami, "An overview of deep semi-supervised learning", *in*: *arXiv preprint arXiv:2006.05278* (2020) (cit. on p. 53).

[Olt+19] Alexandra Olteanu et al., "Social data: Biases, methodological pitfalls, and ethical boundaries", *in*: *Frontiers in Big Data* (2019) (cit. on pp. 68, 69).

[OM05] James A O'brien and George M Marakas, *Introduction to information systems*, McGraw-Hill/Irwin New York City, USA, 2005 (cit. on p. 32).

[PB17] Paul Pols and Jan van den Berg, "The Unified Kill Chain", *in*: *CSA Thesis, Hague* (2017) (cit. on p. 39).

[PB18] Ranjit Panigrahi and Samarjeet Borah, "A detailed analysis of CICIDS2017 dataset for designing Intrusion Detection Systems", *in*: *International Journal of Engineering & Technology* (2018) (cit. on p. 74).

[PBC15] Todd Perry, Mohamed Bader-El-Den, and Steven Cooper, "Imbalanced classification using genetically optimized cost sensitive classifiers", *in*: *2015 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2015 (cit. on pp. 91, 93).

[PD07] Larry L Peterson and Bruce S Davie, *Computer networks: a systems approach*, Elsevier, 2007 (cit. on p. 33).

[Pei+20] Wenbin Pei et al., "Genetic programming for high-dimensional imbalanced classification with a new fitness function and program reuse mechanism", *in*: *Soft Computing* (2020) (cit. on p. 91).

[Pel21] LeeAnne M. Pelzer, *The true cost of cyber security incidents*, https://www.paloaltonetworks.com/blog/2021/06/the-cost-of-cybersecurity-incidents-the-problem/, 2021 (cit. on p. 41).

[PKP21] Mirjam Pot, Nathalie Kieusseyan, and Barbara Prainsack, "Not all biases are bad: equitable and inequitable biases in machine learning and radiology", *in*: *Insights into imaging* (2021) (cit. on pp. 16, 23, 67).

[Poz+21] Muhammad Syafiq Mohd Pozi et al., "SVGPM: evolving SVM decision function by using genetic programming to solve imbalanced classification problem", *in*: *Progress in Artificial Intelligence* (2021) (cit. on p. 92).

[PS22] Dana Pessach and Erez Shmueli, "A Review on Fairness in Machine Learning", *in*: *ACM Computing Surveys (CSUR)* (2022) (cit. on p. 70).

[Rai12] Costin Raiu, "Cyber-threat evolution: the past year", *in*: *Computer Fraud & Security* (2012) (cit. on pp. 17, 25, 31, 113).

[Rau+20] Pierre-Yves Raumer et al., "Reinforcement Learning Library based on Tangled Program Graphs: Development of New Learning Environments and Library Features", *in*: (2020) (cit. on p. 161).

[Ren+08] Rafal Renk et al., "Intrusion detection system based on matching pursuit", *in*: *2008 First International Conference on Intelligent Networks and Intelligent Systems*, IEEE, 2008 (cit. on p. 21).

[Rin+19] Markus Ring et al., "A survey of network-based intrusion detection data sets", *in*: *Computers & Security* (2019) (cit. on pp. 16, 23, 57, 124).

[Sá+20] Alex GC de Sá et al., "A robust experimental evaluation of automated multi-label classification methods", *in*: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 2020 (cit. on p. 92).

[SA13] Santanu Santra and Pinaki Pratim Acharjya, "A Study And Analysis on Computer Network Topology For Data Communication", *in*: *International Journal of Emerging Technology and Advanced Engineering* (2013) (cit. on p. 33).

[SAH21] Robert J Smith, Ryan Amaral, and Malcolm I Heywood, "Evolving simple solutions to the CIFAR-10 benchmark using tangled program graphs", *in*: *2021 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2021 (cit. on pp. 89, 92, 93, 99).

[Sar+20] T Saranya et al., “Performance analysis of machine learning algorithms in intrusion detection system: A review”, *in*: *Procedia Computer Science* (2020) (cit. on p. 47).

[SB18] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018 (cit. on p. 55).

[SG21] Harini Suresh and John Guttag, “A framework for understanding sources of harm throughout the machine learning life cycle”, *in*: *Equity and Access in Algorithms, Mechanisms, and Optimization*, 2021 (cit. on p. 69).

[SGA20] S Sandosh, V Govindasamy, and G Akila, “Enhanced intrusion detection system via agent clustering and classification based on outlier detection”, *in*: *Peer-to-Peer networking and Applications* (2020) (cit. on p. 46).

[SLG18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization.”, *in*: *ICISSP*, 2018 (cit. on pp. 15, 21, 58, 69, 71, 74, 75, 81, 107, 114, 126).

[SLP22] Mohanad Sarhan, Siamak Layeghy, and Marius Portmann, “Towards a standard feature set for network intrusion detection system datasets”, *in*: *Mobile Networks and Applications* (2022) (cit. on pp. 71, 116–118, 137, 139, 143, 144).

[Smi03] Clifton L Smith, “Understanding concepts in the defence in depth strategy”, *in*: (2003) (cit. on p. 47).

[Sou+22a] Nicolas Sourbier et al., “Imbalanced Classification with TPG Genetic Programming: Impact of Problem Imbalance and Selection Mechanisms”, *in*: *The Genetic and Evolutionary Computation Conference*, acm, 2022 (cit. on pp. 89, 146).

[Sou+22b] Nicolas Sourbier et al., “SECURE-GEGELATI Always-On Intrusion Detection through GEGELATI Lightweight Tangled Program Graphs”, *in*: *Journal of Signal Processing Systems* (2022) (cit. on pp. 113, 137, 146).

[SSG18] Rohini Sharma, RK Singla, and Ajay Guleria, “A new labeled flow-based DNS dataset for anomaly detection: PUF dataset”, *in*: *Procedia computer science* (2018) (cit. on pp. 15, 21).

[Sta87] William Stallings, *Handbook of computer-communications standards; Vol. 1: the open systems interconnection (OSI) model and OSI-related standards*, Macmillan Publishing Co., Inc., 1987 (cit. on p. 36).

[SV17] Rafath Samrin and D Vasumathi, "Review on anomaly based network intrusion detection system", *in*: *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, IEEE, 2017 (cit. on p. 46).

[Sym17] Symantec, *ISTR Volume 22| Symantec*, 2017 (cit. on pp. 17, 25, 31, 46, 113).

[Tav+09] Mahbod Tavallaee et al., "A detailed analysis of the KDD CUP 99 data set", *in*: *2009 IEEE symposium on computational intelligence for security and defense applications*, Ieee, 2009 (cit. on p. 70).

[Tec] Institut d'Electronique et des Technologies du numéRique (IETR), *Generic Evolvable Graphs for Efficient Learning of Artificial Tangled Intelligence*, URL: https://github.com/gegelati/gegelati (cit. on p. 149).

[Van20] Paul C Van Oorshot, *Computer Security and the Internet: tools and jewels*. Springer Nature, 2020 (cit. on pp. 13, 19, 20, 31).

[Vet21] Antonio Vetrò, "Imbalanced data as risk factor of discriminating automated decisions", *in*: (2021) (cit. on p. 69).

[Vie+18] Felipe Viegas et al., "A genetic programming approach for feature selection in highly dimensional skewed data", *in*: *Neurocomputing* (2018) (cit. on p. 91).

[VKS10] Susana M Vieira, Uzay Kaymak, and João MC Sousa, "Cohen's kappa coefficient as a performance measure for feature selection", *in*: *International Conference on Fuzzy Systems*, IEEE, 2010 (cit. on p. 92).

[Wan+15] CY Wang et al., "imDC: an ensemble learning method for imbalanced classification with miRNA data", *in*: *Genetics and Molecular Research* (2015) (cit. on p. 91).

[War+19] Ni Wayan Surya Wardhani et al., "Cross-validation metrics for evaluating classification performance on imbalanced data", *in*: *2019 international conference on computer, control, informatics and its applications (ic3ina)*, IEEE, 2019 (cit. on p. 98).

[WD92] Christopher JCH Watkins and Peter Dayan, "Q-learning", *in*: *Machine learning* (1992) (cit. on p. 54).

[Wei05] Sanford Weisberg, *Applied linear regression*, John Wiley & Sons, 2005 (cit. on p. 50).

[WRM03] Jianxin Wu, James Matthew Rehg, and Matthew D Mullin, *Learning a rare event detection cascade by direct feature selection*, tech. rep., Georgia Institute of Technology, 2003 (cit. on pp. 89, 116).

[WRR03] Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha, "Singular value decomposition and principal component analysis", *in*: *A practical approach to microarray data analysis*, Springer, 2003 (cit. on p. 53).

[Wu+21] Zhijun Wu et al., "An incremental learning method based on dynamic ensemble RVM for intrusion detection", *in*: *IEEE Transactions on Network and Service Management* (2021) (cit. on pp. 116, 118, 137).

[WZ15] Marvin N Wright and Andreas Ziegler, "ranger: A fast implementation of random forests for high dimensional data in C++ and R", *in*: *arXiv preprint arXiv:1508.04409* (2015) (cit. on pp. 127, 135).

[YH18] Binghao Yan and Guodong Han, "Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system", *in*: *IEEE Access* (2018) (cit. on p. 71).

[YL06] Show-Jane Yen and Yue-Shi Lee, "Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset", *in*: *Intelligent Control and Automation*, Springer, 2006 (cit. on pp. 17, 24, 89, 91).

[YR15] Tarun Yadav and Arvind Mallari Rao, "Technical aspects of cyber kill chain", *in*: *International Symposium on Security in Computing and Communication*, Springer, 2015 (cit. on p. 40).

[YSS19] Arif Yulianto, Parman Sukarno, and Novian Anggis Suwastika, "Improving adaboost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset", *in*: *Journal of Physics: Conference Series*, 2019 (cit. on p. 127).

[Zam01] Diego Zamboni, "Using Internal Sensors For Computer Intrusion Detection", *in*: (2001) (cit. on pp. 13, 19, 20, 31, 113).

[ZCS20] Wei Zong, Yang-Wai Chow, and Willy Susilo, "Interactive three-dimensional visualization of network intrusion detection data for machine learning", *in*: *Future Generation Computer Systems* (2020) (cit. on pp. 116, 118, 137).

[ZD21] Hongjing Zhang and Ian Davidson, "Towards fair deep anomaly detection", *in*: *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021 (cit. on p. 71).

[Zhu05] Xiaojin Jerry Zhu, "Semi-supervised learning literature survey", *in*: (2005) (cit. on p. 53).

[Zhu20] Qiuming Zhu, "On the performance of Matthews correlation coefficient (MCC) for imbalanced dataset", *in*: *Pattern Recognition Letters* (2020) (cit. on p. 91).

[ZL03] Mian Zhou and Sheau-Dong Lang, "A frequency-based approach to intrusion detection", *in*: *Proc. of the Workshop on Network Security Threats and Countermeasures*, 2003 (cit. on p. 21).

[ZL14] Huaxiang Zhang and Mingfang Li, "RWO-Sampling: A random walk oversampling approach to imbalanced data classification", *in*: *Information Fusion* (2014) (cit. on pp. 17, 24, 89, 91).

DOCTORAT BRETAGNE LOIRE / MATHSTIC

INSA RENNES

Titre : Détection d'intrusion réseaux par apprentissage machine : Un problème temporel, déséquilibré et en constante évolution

Mot clés : Détection d'intrusion, Programmation génétique, Sécurité réseau, Apprentissage déséquilibré, Apprentissage incrémentiel et TPG

Résumé : Les systèmes de détection d'intrusion réseau (NIDS) observent le trafic réseau et essayent d'en extraire les intrusions : des compromissions de l'intégrité, la disponibilité ou la confidentialité des services et des données fournies par ce réseau. Il existe deux types de NIDS. 1) Les systèmes de détection d'intrusion par signature identifient les intrusions connues en se référant à une base de connaissance existante. 2) Les systèmes de détection d'intrusion par anomalie qui qualifient les intrusions en se basant sur un modèle du trafic réseau normal, généralement appris par des techniques d'apprentissage machine.

La détection d'intrusion dans les réseaux, en évolution constante, comporte des verrous pour être déployée. Premièrement, la collecte de données réseau représentatives et correctement étiquetées est complexe et coûteuse. Ces données sont également fortement déséquilibrées, les attaques étant des événements rares. Enfin, le portage opérationnel d'un AIDS appris peut entraîner une chute des taux de détection de par la différence entre le contexte d'apprentissage et le contexte d'inférence.

Ce manuscrit explore les apports des TPGs au domaine de la détection d'intrusions par anomalies et montre que les TPG sont prometteurs pour la levée des verrous du domaine.

Title: Learning-Based Network Intrusion Detection : an Imbalanced, Constantly Evolving and Timely Problem

Keywords: Intrusion detection, Genetic Programming, Network Security, Imbalanced Learning, Incremental Learning, Tangled Program Graphs

Abstract: Network Intrusion Detection Systems (NIDS) observe a network environment and aim to identify intrusions: malicious behaviors that compromise integrity, confidentiality or availability of either the network data or the systems. NIDS can be classified into signature-based NIDS, that identify known intrusions by comparing the traffic with a knowledge base and anomaly-based NIDS (AIDS) that aim to qualify the unknown intrusion traffic, from a model of normal traffic, mostly based on Machine Learning techniques.

Performing detection of rare events such as intrusions in an ever-changing network environment using learned AIDS is a challenge bound to several big issues. Firstly, gathering representative network data with accurate label information is costly. These data are also highly imbalanced as intrusions are rare events. Finally, there is no guarantee that a learned AIDS on a network intrusion detection dataset is useful for real NIDS inference.

This thesis explores the capabilities of the TPG framework to act as an AIDS probe. TPG is a form of machine learning based on genetic programming that offers lightweight and versatile learning capabilities.