AVERTISSEMENT

LIENS

Code la Propriété Intellectuelle – Articles L. 122-4 et L. 335-1 à L. 335-10
Loi n° 92-597 du 1er juillet 1992, publiée au *Journal Officiel* du 2 juillet 1992

http://www.cfcopies.com/V2/leg/leg-droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse Capitole

École doctorale : **Mathématiques, Informatique, Télécommunications de Toulouse**

Présentée et soutenue par

COUSTIÉ Oihana

le 11 mai 2021

Detecting anomalies in modern IT systems through the inference of structure and the detection of novelties in system logs

---

Discipline : Informatique

Spécialité : Intelligence Artificielle

Unité de recherche : **Institut de Recherche en Informatique de Toulouse (UMR 5505)**

Directeur de thèse : Monsieur Olivier Teste, Professeur, Université Toulouse Jean Jaurès
Madame Josiane Mothe, Professeur, Université Toulouse Jean Jaurès

JURY

Rapporteurs	Monsieur Eric Gaussier, Professeur, Université Grenoble Alpes Monsieur Pascal Poncelet, Professeur, Université de Montpellier
Suffragant	Madame Anne Sabourin, Maître de conférence, Télécom Paris
Directeur(s) de thèse	Madame Josiane Mothe, Professeur, Université Toulouse Jean Jaurès Monsieur Olivier Teste, Professeur, Université Toulouse Jean Jaurès

Mémoire composé avec LaTeX 2ε.
Version du 3 août 2021.

Oihana COUSTIÉ

Détection d'anomalies dans les systèmes d'information modernes grâce à des méthodes d'inférence de structure et de détection de nouveautés dans les logs

Encadrement de thèse : Xavier BARIL, Josiane MOTHE, Olivier TESTE

Résumé

Les anomalies dans les logs des systèmes d'information sont souvent le signe de failles ou de vulnérabilités. Leur détection automatique est difficile à cause du manque de structure dans les logs, et de la complexité des anomalies. Les méthodes d'inférence de structure existantes sont peu flexibles : elles ne sont pas paramétriques, ou reposent sur des hypothèses syntaxiques fortes, qui s'avèrent parfois inadéquates. Les méthodes de détection d'anomalies adoptent quant à elles une représentation des données qui néglige le temps écoulé entre les logs, et sont donc inadaptées à la détection d'anomalies temporelles.

La contribution de cette thèse est double. Nous proposons d'abord METI*N*G, une méthode d'inférence de structure paramétrique et modulable. METI*N*G ne repose sur aucune hypothèse syntaxique forte, mais se base sur l'exploration de motifs fréquents, en étudiant les n-grammes des logs. Nous montrons expérimentalement que METI*N*G surpasse les méthodes existantes, avec d'importantes améliorations sur certains jeux de données. Nous montrons également que la sensibilité de notre méthode à ses hyper-paramètres lui permet de s'adapter à l'hétérogénéité des jeux de données. Enfin, nous proposons une extension de METI*N*G au contexte de la racinisation en traitement automatique du texte, et montrons que notre approche fournit une racinisation multilingue, sans règle, et plus efficace que la méthode de Porter, référence de l'état de l'art.

Nous présentons également NoTIL, une méthode de détection de nouveautés par apprentissage profond. NoTIL utilise une représentation des données capable de détecter les irrégularités temporelles dans les logs. Notre méthode repose sur l'apprentissage d'une tâche de prédiction intermédiaire pour modéliser le comportement nominal des logs. Nous comparons notre méthode à celles de l'état de l'art et concluons que NoTIL est la méthode capable de traiter la plus grande variété d'anomalies, grâce aux choix de sa représentation des données.

Institut de Recherche en Informatique de Toulouse – UMR 5505 CNRS
Université Toulouse 1 – Capitole, 2 Rue du Doyen Gabriel Marty, 31000 Toulouse
Airbus SAS Operations
316, route de Bayonne, 31060 Toulouse

Oihana COUSTIE

Detecting anomalies in modern IT systems through the inference of structure and the detection of novelties in system logs

Supervisors: Xavier BARIL, Josiane MOTHE, Olivier TESTE

Abstract

The anomalies in the logs of information system are often the sign of faults and vulnerabilities. Their detection is challenging due to the lack of structure in logs and the complexity of the anomalies. Existing methods to infer the structure are poorly flexible : they are not parametric, or rely on strong syntactic assumptions, which sometimes prove to be inadequate. Anomaly detection methods adopt a data representation that neglects the time elapsed between the logs, and are therefore unsuitable for the detection of temporal anomalies.

The contribution of this thesis is twofold. We first propose METI*N*G, a parametric and modular structure inference method. METI*N*G does not rely on any strong syntactic assumption, but is based on the mining of frequent patterns, through the study of n-grams. We experimentally show that METI*N*G surpasses the existing methods, with important improvements on some datasets. We also show the important sensitivity of our method to its hyper-parameters, which allows the exploration of many configurations, and the adaptation to the heterogeneity of datasets. Finally, we propose an extension of METI*N*G to the context of stemming in text mining, and show that our approach provides a stemming solution that is multilingual, rule-free, and more efficient than that of Porter, the state-of-the-art reference.

We also present NoTIL, a novelty detection method based on deep learning. NoTIL uses a data representation capable of detecting temporal irregularities in the logs. Our method is based on the learning of an intermediate prediction task to model the nominal behavior of logs. We compare our method to the most up-to-date references and conclude that NoTIL is the method capable of dealing with the greatest variety of anomalies, thanks to its data representation.

Institut de Recherche en Informatique de Toulouse – UMR 5505 CNRS
Université Toulouse 1 – Capitole, 2 Rue du Doyen Gabriel Marty, 31000 Toulouse
Airbus SAS Operations
316, route de Bayonne, 31060 Toulouse

Remerciements

À l'issue de ces trois années de thèse, je profite de ce manuscrit pour adresser mes remerciements sincères aux personnes qui ont contribué, participé à la réalisation de mes travaux, par leur investissement professionnel ou leur accompagnement personnel.

Je remercie d'abord l'ensemble des membres du jury d'avoir accepté d'évaluer mes travaux de recherche et d'avoir participé à ma soutenance de thèse. Merci à mes rapporteurs, le Professeur Pascal Poncelet et le Professeur Eric Gaussier, qui a également accepté de présider mon jury de thèse. Merci à Anne Sabourin et à Emmanuelle Escorihuela d'avoir évalué mes travaux en tant que respectivement examinatrice et représentante industrielle. Les échanges lors de la soutenance ont été riches et fort intéressants. Je suis reconnaissante de la pertinence de vos remarques, et l'exhaustivité de votre évaluation.

Je tiens à remercier chaleureusement mon équipe encadrante. D'abord mes directeurs de thèse, les Professeurs Olivier Teste et Josiane Mothe, que je remercie pour tout ce qu'ils m'ont appris : sur le plan technique en Machine Learning, mais aussi et surtout sur la méthodologie académique (conduire un état de l'art, adopter une rigueur scientifique dans les validations expérimentales, construire une introduction pertinente ...). Merci de m'avoir guidée et d'avoir accompagné cet apprentissage.

Je remercie tout autant mes encadrants industriels : le Docteur et expert Airbus Xavier Baril, mon tuteur industriel, qui a porté cette thèse avec moi, partageant la fierté des succès et les leçons des échecs. Je le remercie pour son investissement technique, son aide perpétuelle pour définir les orientations d'un sujet si large, passionnant, et riche de perspectives. Merci à Laurent Pochon, le chef du service IYNRI dans lequel j'ai été intégrée pendant ces trois années en tant que membre à part entière. Grâce à ses efforts et sa considération pour mes travaux, j'ai bénéficié d'un cadre privilégié pour développer et promouvoir des méthodes adaptées aux besoins de l'équipe. Je lui suis à titre personnelle reconnaissante de tout ce qu'il m'a appris humainement pour ma première expérience professionnelle.

J'ajoute de sincères remerciements et salutations à l'ensemble de mes collègues Airbus, plus particulièrement le département IYNR de Jean Christophe Albouy, qui m'a chaleureusement accueillie, mais aussi au chef du domaine IYN, Jean François Alonso, qui a relu mes articles, visionné mes vidéos de conférences pour validation avant publication, et a montré un intérêt très valorisant pour mes travaux.

J'adresse par ailleurs beaucoup de reconnaissance et de gratitudes à mes collègues de l'IRIT. En particulier, à tous mes amis doctorants, qui ont rendu ces trois années bien plus amusantes. Merci à Mohamad, Elio, Florent. Merci à mes collègues de bureau et ami.e.s depuis longtemps

et pour longtemps Inès et Elliot. Merci à ma chère Wafa pour toutes les pauses café sur la terrasse, ô combien précieuses pour les moments de doute, pour les conseils, le réconfort, mais aussi pour célébrer les victoires. A vous tous, je souhaite continuer à partager ces moments, quand bien même ce ne serait pas physiquement, autour d'un café sur la terrasse de l'IRIT.

Merci à mes amies qui m'ont soutenue, ont fait preuve de patience, d'écoute à mon égard. Merci à Sandy et Yasmina, qui par leur bonne humeur m'ont redonné le sourire dans les moments difficiles. Merci à Ambre, toujours rationnelle, réconfortante et à l'écoute, intéressée. Et merci à Yousra, pour sa présence systématique, dans les moments de déception, de joie, pour pleurer, rire ensemble, pour faire la fête ou des vacances farniente à Ibiza (hors saison...), pour parler de sujets sérieux et refaire le monde ou au contraire regarder et débriefer les Marseillais. Merci à toutes pour ces moments de bonheur, de paix. J'attends avec impatience de pouvoir célébrer mon diplôme autour d'un verre au Black Lion.

Je remercie du fond du coeur ma famille pour leur soutien indéfectible, pour ces trois années de thèse et tout au long de mon parcours dans le supérieur. Merci d'abord à mes parents Carole et Christophe, de m'avoir épaulée, écoutée, de s'être intéressés au sujet de ma thèse, à mes études. Merci pour leur accueil plus que réconfortant lors des deux premiers confinements. Cet appui familial a sans nul doute été crucial puisqu'il est intervenu à des moments charniers de ma thèse : l'écriture des deux articles de conférences internationales, puis du manuscrit de thèse. Je remercie mes deux soeurs adorées, Amélia et Maylis, pour leur amour, leur soutien, leur écoute, les moments de rire et de partage. Merci à ma nièce, Mia, de me faire rire, à chaque fois que je la vois, de me donner beaucoup de force et un amour inconditionnel et réciproque.

Enfin, merci à mon conjoint, Aurélien, à qui je dédie cette thèse, pour tous les efforts qu'il a fait lors de ces trois années pour moi : les aller-retours pendant deux ans de relation à distance, presque tous les week-ends, la quasi-totalité de la charge mentale et du travail domestique une fois réunis. Je lui suis infiniment reconnaissante de m'avoir rejointe à Toulouse, ce qui a rendu la dernière année de thèse bien plus vivable, plus simple, plus joyeuse. Merci de m'accompagner au quotidien, d'être patient et tolérant malgré mon (fort ?) caractère. Merci d'apporter bonne humeur, humour et bienveillance à mon quotidien.

Contents

General Introduction

Context

With the fast growing of their size and complexity, modern information technology systems (or IT systems) contain numerous sources of potential faults and vulnerabilities (He et al., 2016a; Du et al., 2017; Borghesi et al., 2019). The failure of their services or applications can have significant consequences, ranging from degraded user's experience (Meng et al., 2019) to important financial losses (Zhang et al., 2016). For this reason, the ability to detect anomalies and provide users with failure explanation has become a key issue (Zhang et al., 2016).

The execution logs of services and applications are recognized to be systematically-available resources and to contain valuable run-time information; Yuan et al. (2012) report that 30% of coding lines are dedicated to logging, which is clear evidence that software and system developers have widely adopted logging practices. Indeed, logs are intensively used, both in production and in-service stages, for failure analysis (Mi et al., 2013), anomaly detection (Brown et al., 2018; Meng et al., 2019) or root cause analysis (Du et al., 2017). Yuan et al. (2012) evaluate that log mining diagnosis can be twice as fast as classical manual analysis.

A *log* can be defined as a semi-structured message that traces the system execution. It is automatically generated during the execution through a `print`-like command : the source code contains a line that generates a log line. This log line contains valuable monitoring information, such as the *timestamp* (time when the log occurred), the *level* (the severity of the logging event), or the *content*, an unstructured free-text part. The value of the content is given by an associated code statement. In this code statement, some parts are fix, and form the *event template*, while others are variable, and form the *parameters*. The values of the latter are determined dynamically while executing. The event template can be seen as a generic textual description of the type of information being logged. While the source code provides a clear separation between fix and variable parts, this distinction is not directly available in the log message. The *event type* of a log is an identifier that gathers logs with the same event template.

A *log anomaly* is an unexpected behaviour of the log data, and can be paired to a system anomaly. For instance, a late appearance of logs in a sequence may indicate a *performance anomaly* corresponding to an abnormal temporal irregularity in a service response (Fu et al., 2009a; Du et al., 2017). Through this example, it is clear that detecting log anomalies is an efficient mean to perform system anomaly detection (Borghesi et al., 2019). Moreover, manually analyzing large and complex log datasets represents a cumbersome and error-prone task (He et al., 2016a;

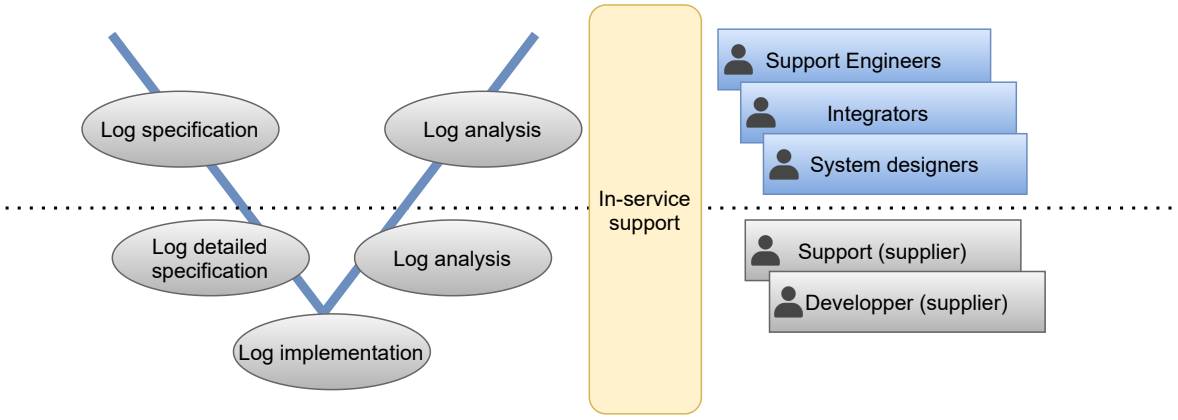


Figure 1 – The usage of FSA-NG logs throughout the V-cycle development and in-service life

Borghesi et al., 2019), justifying the need for automated data-driven solutions.

In the industrial context of this PhD, we take interest in the logs of the On-Board Information System (OBIS) of the A350 aircrafts. The OBIS is a platform that executes services and hosts applications. While executing, these services and applications generate logs, that are used both in development and in the in-service life of the system, on-boarded in airlines' aircrafts. Figure 1 shows that logs are used throughout all the development phases of the V-cycle (Oberkampf and Trucano, 2008). Systems designers and integrators provide high level directives for logs specification while the suppliers detail the definition of the types and formats of logs. Logs are used for test both by the suppliers, with unitary tests, and the module integrators, for integration. The module integrators are interested in logs to monitor the behaviour of the whole system while integrating all the applications and services from multiple providers. They often perform local look-ups in the logs and check for novelties in the relations between applications when a new standard is released. Logs are also used in production phase : the support engineers both (i) monitor the logs in a proactive way, (ii) examine the logs on-demand, to provide the customer airlines with root cause analysis of in-service issues of the system.

Challenges

The automated detection of system anomalies through the analysis of logs raises some challenging research questions. Firstly, the unstructured nature of the logs, along with the huge size of the datasets constitute important challenges for structuring the logs. Secondly, the state-of-the-art methods for detecting anomalies in logs only focus on sequential anomalies, while temporal anomalies are discarded.

Structure inference in log datasets. Most anomaly detection techniques represent logs with their event types (He et al., 2016b; Brown et al., 2018). Unfortunately, the event type field is seldom present in the log message. Moreover, the content part of the raw log messages do not contain any structural information, preventing a straightforward event type inference. Therefore, there is a substantial need of solutions to structure the content part of log messages and retrieve their event types. Due to both the huge size and fast changing character of log datasets, it is unfeasible to

manually maintain a database of event templates (Zhu et al., 2019). The solution therefore needs to be automated and data-driven.

Automated data driven methods to infer event types are called *log parsers* and aim at forming groups of logs so as to retrieve the group templates (Mi et al., 2013). Among the state-of-the-art log parsers, parametric ones have proven to be more efficient and robust, thanks to more flexible behaviour towards the heterogenity of datasets (Zhu et al., 2019). However, the most robust solutions (Makanju et al., 2012; Du and Li, 2016; He et al., 2017) still leave some state-of-the-art datasets behind : they rely on strong assumptions (e.g., logs with the same event types have the same number of words) that turn out to be inadequate for some datasets. Among these challenging datasets, OpenStack [1] is a reference for the assessment of anomaly detection methods (Du et al., 2017). There is therefore still room for improvement in the field of log parsing.

The anomaly detection task. Once the event type is retrieved, it is still not trivial to detect log anomalies. *Supervised methods*, that are popularly used in machine learning tasks for classification suffer from the scarcity of abnormal samples, and fail to learn the behaviour of the abnormal class (Chen et al., 2004; Liang et al., 2007). The *unsupervised methods* tend to simply detect samples that are both rare and significantly different from the majority regarding selected features (He et al., 2016a).

As an alternative, novelty detection methods learn the nominal behaviour on the normal class and detect anomalies as violations to the trained behaviour. While these methods have proven effective (Du et al., 2017; Meng et al., 2019), most of them adopt the same data representation : a temporally-ordered sequence of the event type that neglects the quantitative value of the elapsed time between logs. This representation is efficient for the detection of sequential anomalies, but cannot be applied for the detection of temporal ones, such as performance anomalies (Tan et al., 2012; Du et al., 2017). The literature lacks a global method that would be able to deal with both sequential and temporal anomalies.

Solution overview and contributions

To answer the aforementioned issues, we propose an end-to-end solution to detect anomalies in logs, as presented in Figure 2. Our solution is divided in two main parts : (i) the structure inference part takes as input a raw database and applies a set of operations so as to transform the logs into manipulable events (ii) the anomaly detection part takes the generated events as input and outputs the anomalies within the log events.

The structure inference part consists of two main steps. First, the log parsing step gathers logs into groups that share common templates, then the pattern extraction phase retrieves the underlying pattern of each log group. We propose a new log parsing method, METI*N*G, standing for Modular Event Type Inference based on *N*-Grams (**?**). METI*N*G has a strong modulation power thanks to (a) a flexible data representation based on frequent n-gram mining (instead of relying on strong syntactic assumptions) (b) an important sensitivity to its hyper-parameters, which enhances the modulation power of the algorithm, and enables the adaptation to the high hetero-

1. https://www.openstack.org

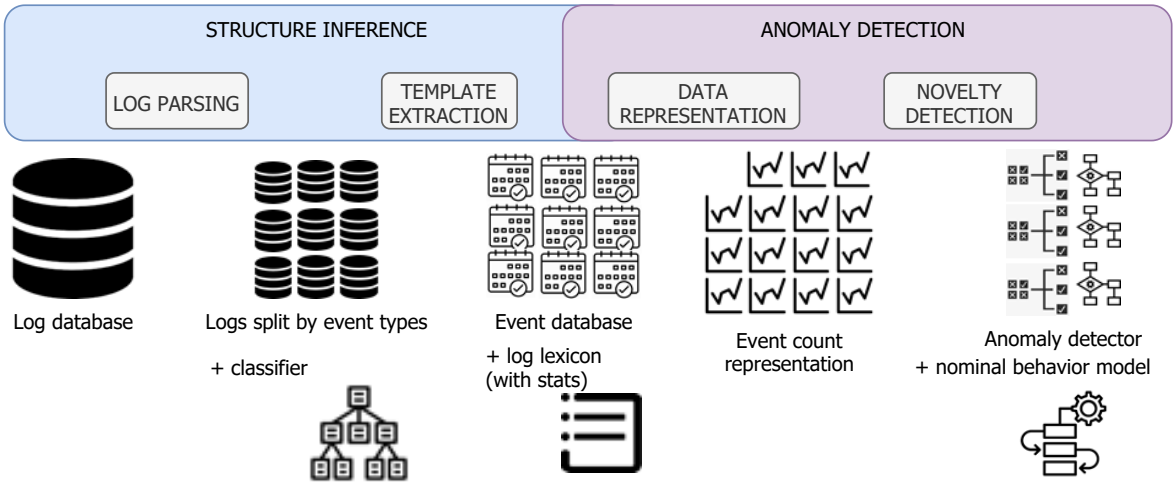


Figure 2 – An overview of our end-to-end solution for detecting anomalies in log databases

geneity of real-world datasets.

METI*N*G globally outperforms the existing log parsers on most of the reference datasets, including challenging ones. Once the logs are grouped, we propose a comprehensive algorithm to extract the template of the underlying groups. The structure inference phase outputs the following elements, as depicted in Figure 2 :

— a classifier, based on the log parsing strategy. This classifier is used for new logs : instead of running again the parsing task, logs are directly assigned to the existing groups;
— a log lexicon, containing the list of groups, and for each of them, some valuable statistics (e.g. periodicity, frequency, value of parameters. . .);
— an event database, which is the input of the next phase, the anomaly detection.

Once the logs are structured, they can be represented as a database of events. A *log event* can be described as a couple of timestamp and event type. Log events are the most commonly used data representation for anomaly detection (Du et al., 2017; Meng et al., 2019). Most of the existing methods define anomalies in logs as anomalies in the sequences of event types. Hence, they neglect the detection of temporal anomalies. We propose a more comprehensive censusing of the types of anomalies and present a new anomaly detection method, NoTIL, that stand for Novelty detection based on Temporal Irregularities in Logs (**?**), that can catch all the censused types of anomalies. The originality in the method stands in (a) its novelty detection approach, that outperforms traditional supervised and unsupervised techniques (Du et al., 2017) and (b) its feature extraction method : NoTIL uses a temporal event count representation of the log events, instead of the classic sequential representation (Du et al., 2017; Brown et al., 2018) and is therefore able to catch both sequential and temporal anomalies. As presented is Figure 2, NoTIL outputs:

— the list of anomalies within a dataset — which enables the evaluation of effectiveness, if ground truth is available;
— a nominal behaviour model, learnt on the training phase on nominal samples.

Along with the outputs that help the global functioning and the final output (the anomalies), our process also renders informative outputs — namely the log lexicon and the nominal behaviour model — that can provide capitalization insights of the log system. In a context where log databases are heterogeneous, fast changing and of large size, this capitalization can help users

better understand how the system generates its logs, and even how the system works.

Manuscript organization

This manuscript is organized in two parts. The Part I details our structure inference solution for log datasets. Chapter 1 presents the challenges of manipulating log datasets, due to the size of the data, the heterogeneity of logs formats and their semi-structured nature. In this introductory chapter, we also provide an insight on the limits of the existing solutions. In Chapter 2, we census the state-of-the-art work on both log parsing and template extraction. Chapter 3 details our contributions for log parsing, METI*N*G, along with its online extension, and for template extraction on parsed groups. In Chapter 4, we evaluate our log parser thanks to labeled data, and present the results of template extraction. We also take interest in the efficiency of METI*N*G, as it guarantees (i) its scalability to huge datasets, (ii) its applicability to online anomaly detection, (iii) the feasibility of the hyperparameters optimization. Finally, Chapter 5 illustrates the generalization power of our approach by extending it to the task of stemming, in the context text mining.

The Part II focuses on the anomaly detection task on logs, based on the event type representation, retrievable thanks to the works of the previous part. In Chapter 1, we present the challenges of applying anomaly detection methods on the log event database, as well as the limitations of existing propositions. Chapter 2 details the state-of-the-art propositions for the anomaly detection task on log datasets. In Chapter 3, we detail NoTIL, our novelty detection method : we present the originality of the feature selection based on a temporal event count representation, define different deep learning architectures to fulfil the learning of the nominal behaviour, and explain how to detect anomalies based on violations of this nominal model. Eventually, Chapter 4 evaluates the detection performance of NoTIL compared to the reference methods. To assess our method, we generate simulated data based on a new typology of anomalies. We also evaluate the methods on real-world datasets.

Part

Structure inference in log datasets

1 Context and challenges

In this chapter, we first detail the features describing log dataset and provide key figures that illustrate the difficulties of inferring the structure of log datasets. We then describe the main steps and explain their individual challenges. We also give an insight on existing methods' limitations before proposing a new solution.

1.1 The pervasiveness of Information System logs

With the increase of size and complexity of modern IT systems (Lin et al., 2016), it is now challenging to get a comprehensive understanding of their global behaviour. However, this comprehension is mandatory for both end-users and developers in order to detect system anomalies in the context of development and maintenance (Zhu et al., 2019). Yet, developers only have a partial view of the systems : since open-source systems (e.g. Hadoop, Spark) can be implemented by hundred of developers, most of them have full understanding only of the small components they are responsible for (He et al., 2016a), which makes it difficult to debug the system during the development phase. They also face the challenge of helping system maintenance during the in-service life of the system (also called production). Production failures are hard to reproduce for analysis because (i) end-users are often reluctant to provide the full context of execution for confidentiality reasons (ii) this context can be extremely expensive to recreate (e.g. same libraries configurations, same hardware...) (Yuan et al., 2012). End-users have little available elements to work on : the source code of the system is seldom available, especially in the case of third-party usage. Finally, classical software debugging methods (e.g. debuggers) are inconvenient to be attached to complex systems like online platforms with multiple clusters (Lin et al., 2016).

Generating execution logs is therefore regarded as an efficient way to get insights of the system behaviour. Easily generated through a `print`-like statement, logs record valuable run-time information (Yuan et al., 2012; Zhu et al., 2019). They contain high level information of the execution of the systems (Chen and Jiang, 2017); their generation can be triggered by (i) an error, being reported or detailed, (ii) a step that is achieved in execution, or (iii) a periodic monitoring of system values.

The generated logs can be monitored during both the development phase, — to understand the system, detect anomalies, or observe changes when systems are updated, — and during the in-service life of the system, mainly for maintenance purposes. During developments, logs can be used by developers to test and analyse their code (Jiang et al., 2008, 2009; Chen et al., 2018) or to evaluate the performance (Nagaraj et al., 2012; Chow et al., 2014). They are also valuable

monitoring resources to detect security threats on clients' applications (Montanari et al., 2012; Oprea et al., 2015). In production, recent work have focused on analysing the system usage through the execution logs : it includes user behaviour analysis (Yu et al., 2012; Lee et al., 2012a), business process mining (Poggi et al., 2013) and statistics analysis (Lee et al., 2012b). In both development and production phases, another important usage of logs consists in detecting abnormal system behaviours (Xu et al., 2010, 2009a; Lou et al., 2010; Du et al., 2017; Meng et al., 2019; **?**). Detecting anomalies is a powerful mean to provide insights for root cause analysis and error diagnosis for developers and support engineers (Yuan et al., 2010; Xu et al., 2014); Yuan et al. (2012) evaluate that troubleshooting is more than twice faster when log messages are available.

In front of the great potential of log analysis to answer the system analysis issues, generating logs is now recognized as a systematic practice for developers (Ding et al., 2014; Zhao et al., 2017). Both Fu et al. (2014) and Chen and Jiang (2017) study the pervasiveness of log statements in source codes, on different systems. Fu et al. (2014) reports that 1 line of source code out 30 (3.3%) is dedicated to logging — Chen and Jiang (2017) studies different systems and evaluates this rate at 1/51 (2.0%). In both cases, these figures demonstrate the pervasiveness of logging during software development. The industrial adoption of logging practices is another proof of this pervasiveness : Gartner (2014) estimate the market of tools for log management to 1.5 billion dollars, with a spectacular growth of 10% each year.

Despite the numerous sources of generation and potentially tremendous amount of data generated, it is still hard to get hands on production datasets. Indeed, private companies are often reluctant to release their logs for research purpose, since they might contain confidential information. While service providers manage to get back up to 50% of the in-service logs of their clients (Yuan et al., 2012), the research area suffers from the scarcity of publicly available production logs (He et al., 2016a; Zhu et al., 2019). It is therefore challenging to design, fine-tune, and evaluate generic anomaly detection methods. In this context, the LogPAI (Zhu et al., 2016) project provides a resourceful research framework for the log analysis. Its repository Loghub (He et al., 2020b) contains 16 execution logs datasets, that are recognized as the state-of-the-art reference (Oliner and Stearley, 2007; Xu et al., 2010; Du et al., 2017; Meng et al., 2019), and are studied and referred to throughout the entire manuscript.

We conclude that logs can be easily generated and contain rich and comprehensive information on the system behaviour. Hence, log datasets constitute the main resource for system analysis and especially system anomaly detection.

1.2 Execution logs : description, characteristics and key figures

This section aims at describing the log datasets to provide a deep understanding of the data structure, format and specificities. We aim at highlighting the difficulty withheld by the task of inferring the structure of such data, while showing the informative potential they represent. As depicted in Figure 1.1, a log contains:

— a timestamp, here "2015-10-18 18:05:29,570", which is the generation time of the log;
— a level, that gives the criticality of the log, here "INFO";
— a component, a free-text field describing the information logged, here "Received block blk_-562…".

Source code

```
/* Extract of code from : hadoop/hdfs/server/datanode/BlockReceiver.java */
LOG.info("Received block " + block + " of size " + block.getNumBytes() + " from " + inAddr);
```

Generated log

```
2015-10-18 18:05:29,570 INFO dfs.DataNode$PacketResponder: Recevied block blk_-562725280853087685 of size 67108864 from /10.251.91.84
```

Log structure

Field name	Available in generated log	Value
TIMESTAMP	Yes	2015-10-18 18:05:29,570
LEVEL	Yes	INFO
COMPONENT	Yes	dfs.DataNode$PacketResponder
CONTENT	Yes	Recevied block blk_-562725280853087685 of size 67108864 from /10.251.91.84
EVENT TEMPLATE	No	Recevied block <*> of size <*> from /<*>
PARAMETERS LIST	No	['blk_-562725280853087685', '67108864', '10.251.91.84']

Figure 1.1 – A log message with its source code and structured version, inspired from Zhu et al. (2019)

However, other features of the logs can sometimes be indirectly extracted and present an important interest for the data representation :

— the event type, an identifier of the type of log;
— the event template, a generic string shared by all the logs with the same event type. It contains the fix parts of the log. In the example, the event template can be represented by the string `Received block <*> of size <*> from /<*>`, where the wild cards **<*>** designate the place of the variable parts;
— the parameters, the value of the variable parts of the log (for instance `10.251.91.84`);
— a sequence identifier which regroups logs from the same sequence. The sequence notion is related to domain knowledge, and in the example is `blk_-562725280853087685`.

With the numerous information available in all these fields, we provide an insight on some key features of the logs in the following paragraphs.

1.2.1 Source and generation

Logs are a pervasive way to perform in-service anomaly detection, as well as debugging during production. Hence, logs are retrievable in a multitude of different sources. The following paragraphs present the different types of sources and acquisition methods of the logs.

Operating systems. The reference state-of-the-art datasets include the logs of the three most popular operating systems (OS) in the market, namely Windows, Mac and Linux. These logs are directly retrievable in a personal computer (e.g. at path /var/log/system.log for Mac OS). The Windows logs were generated in a lab environment from different sources that aim at studying the

behaviour of the CBS (Component Based Servicing) on Windows 7 (He et al., 2020b). The Linux logs were generated as part of the Public Security Log Sharing Site project (Chuvakin, 2019). Finally, the Mac logs trace user's behaviour on a personal computer (He et al., 2020b).

Distributed systems. Distributed systems have complex behaviours and logs are often used as a mean to trace the exchange between entities. HDFS (Hadoop Distributed File System) is a distributed file system used for the distributed storage of the Hadoop applications. HDFS logs are easily retrievable at the node level and for the resource manager and are gathered is the dataset HDFS. Hadoop is an open-source framework that enhances the usage of distributed applications for massive datasets thanks to cluster computing. The Hadoop dataset contains the execution logs of two applications : WordCount and PageRank. The Spark logs gather logs from the execution of an Apache Spark [1] system in a lab environement (He et al., 2020b). This system is designed for the processing and analysis of big data, thanks to a set of built-in functions for machine learning, graph processing... The Zookeeper dataset contains lab logs from a Zookeeper service (He et al., 2020b). Zookeeper [2] is a coordination service that supports, among other things, distributed configuration management, synchronization and naming. Finally, OpenStack [3] is an open cloud platform that manages processing pools, storage and networking resources. The OpenStack dataset was generated on CloudLab [4] and traces the creation and use of virtual machines (VM).

Supercomputers. The logs of supercomputers are acknowledged to be essential sources to better understand the behaviour of such powerful machines : Oliner and Stearley (2007) claims that reliability and performance challenges need a deeper understanding of these computers, while logs are the first source of information for supercomputer administrators. The BGL dataset is collected from a BlueGene/L supercomputer at Lawrence Livermore National Labs (LLNL). HPC is a dataset generated by a high performance computing cluster at Los Alamos National Laboratories. Finally, Thunderbird logs were collected from the execution of a Thunderbird supercomputer system at Sandia National Labs (SNL).

Mobile applications. Mobile operating systems also generate execution logs : Android logs gather the framework logs of an Android operating system [5]. These logs are rarely available since they are seldom released for public usage (He et al., 2020b). The mobile application hosted on the Android operating system also generate logs, such as the application HealthApp.

Server applications. Logs can also be a valuable resource for server applications to trace clients' interactions with the server. The Apache logs contain access and error logs from an Apache Web server [6], running on a Linux system, in the context of the Public Security Log Sharing Site project (Chuvakin, 2019). The OpenSSH dataset were collected in a lab environment with an OpenSSH server, that provides remote logins based on the SSH protocol (He et al., 2020b).

1. https://spark.apache.org
2. https://zookeeper.apache.org
3. https://www.openstack.org
4. https://cloudlab.us/
5. https://www.android.com
6. Server. https://httpd.apache.org

Standalone software. The Proxifier[7] software provides SOCKS or HTTPS proxy and chains for network applications. The software logs were collected in an lab environment (He et al., 2020b).

Aircraft on-board information system. The design of the understudied on-board information system (OBIS) is composed of 2 components dedicated to ACD (Aircraft Control Domain) and AISD (Airline Information Services Domain) domains defined in the aeronautic industrial standard ARINC 664P5 (2005). Both components host applications and execute internal basic services. During their execution, the basic services of the platform and the hosted applications generate execution logs. These logs are generated on-board and transmitted to the ground through archives containing logs from all the OBIS applications and services .

Systems have different purposes and functioning, they log different types of information, with important heterogeneity in, among other things, the format of the content part. This heterogeneity makes the application of a unique process difficult. The following section presents some general features of the log datasets that illustrate this heterogeneity.

1.2.2 General features : the challenges of manipulating log datasets

This section presents some important features of the log datasets concerning the volume, format and organization of the logs. We aim at highlighting the characteristics that make the processing of logs challenging. We show that logs constitute massive datasets, are heterogeneous and change fast. Table 1.1 presents the studied features detailed in this subsection. .

Storage design. The considerations about the storage of logs is a dimensioning feature for the choice of the logging strategy. In addition to their naturally important size, due to the complexity and multiplicity of sources, some systems impose storage constraints (e.g. long term storage for historical log consultation, duplicate storage for resilience (Liu et al., 2019b)), that can lead to important storage pressure. The storage capacity allocated for the logs also has a direct impact on the logging capacities : operating and mobile system logs are stored locally, yet, a personal computer generally has much more storage space available than a cellphone. These two criteria are dimensioning for the set up of the log verbosity and frequency. When the system logs volume reaches the allocated storage resources, logs can be lost (re-writing over the oldest logs or not writing the new logs). The OBIS follows heterogeneous dynamics to manage log storage, which broadly relies on the deletion of the oldest logs.

Besides, the storage architecture varies from a system to another. In Table 1.1, most systems write their logs linearly in a unique log file (e.g. BGL, Apache). Some other systems organize their logs in accordance with the generation method, respecting the architecture of the entities that generated the logs. In Spark and Hadoop, as well as for the OBIS, the logs are organized in repositories that reflect the applications of the system. Finally, the OpenStack dataset that is available for research purpose is actually divided in several files. These different storage architectures are yet another element that makes the processing of log datasets heterogeneous.

7. https://www.proxifier.com

Dataset	Number of logs	File size	Recording duration	Storage architecture	Frequence of logging (log/hour)
Operating System					
Mac	117,283	16.09MB	7.0 days	Unique file	699
Windows	114,608,388	26.09GB	226.7 days	Unique file	21,064
Linux	25,567	2.25MB	263.9 days	Unique file	4
Distributed System					
OpenStack	207,820	58.61MB	29.5 hours	Unique file	4,649
HDFS	11,175,629	1.47GB	38.7 hours	Unique file	288,882
Zookeeper	74,380	9.95MB	26.7 days	Unique file	642
Spark	33,236,604	2.75GB	5.6 days	Application	263,047
Hadoop	394,308	48.61MB	18.2 hours	Application	21,670
Supercomputer					
Thunderbird	211,212,192	29.60GB	244.7 days	Unique file	35,971
BGL	4,747,963	708.76MB	214.7 days	Unique file	922
HPC	433,489	32.00MB	27.9 hours	Unique file	15,515
Mobile system					
Android	1,555,055	183.37MB	24.4 hours	Unique file	87,733
Healthapp	253,395	22.44MB	10.5 days	Unique file	1,007
Server application					
OpenSSH	655,146	70.02MB	28.4 days	Unique file	960
Apache	56,481	4.90MB	263.9 days	Unique file	9
Standalone Software					
Proxifier	21,329	2.42MB	25.6 hours	Unique file	934

Table 1.1 – General features of the reference datasets (Zhu et al., 2019; He et al., 2020b)

Data volume metrics. Due to the scale, complexity and parallel functioning of modern systems, the logs they generate become extensively huge datasets (Lin et al., 2016; Liu et al., 2019b; Zhu et al., 2019). These datasets are generally estimated to produce 50 GB of data per hours, corresponding to 200 million log messages per hour (Mi et al., 2013; He et al., 2016a; Zhu et al., 2019). In some extreme cases, like cloud architecture, Lin et al. (2016) and Liu et al. (2019b) estimate that such system can log up to several PB of data a year. There is however some heterogeneity, especially between production logs and logs generated in a lab environment, with simplified configurations (Lin et al., 2016). In their comprehensive study, Fu et al. (2014) focus on systems that produce 2GB per machine per day, which is actually the most common situation. Regardless, this high generation rates lead to huge log datasets to be studied.

It is the case of most of the understudied reference datasets. In table 1.1, we present (i) the number of logs in the datasets (column Number of logs), (ii) the size of the corresponding files (column File size), (iii) the total recording time (column Recording duration) and (iv) the logging frequency (number of logs per hour, in column Frequence of logging). Some datasets contain hundreds of million logs (e.g. Windows, Thunderbird), representing several GB of storage, while others only contain tens of thousands logs (e.g. Linux, Zookeeper, Proxifier), with light storage impacts (i.e. a few MB). Each archive of the OBIS logs contains in average more than 15 million logs, gen-

erated by the applications and services. Our collection gathers 850 million logs from 56 archives, obtained by selecting 14 out of the 103 repositories created by the applications and services. Moreover, the applications and services present an important heterogeneity in size, even on the same cabinet.

The size of the understudied datasets does not only depend on the verbosity or the complexity of the system, but also on the recording duration. Windows logs were recorded during 227 days while the HDFS dataset corresponds to a very short duration of 39 hours. Proportionally to the recording duration, the HDFS logs are much denser then the ones of Windows. While the size of the available data has a direct impact on the log mining strategy, the logging frequency is a great indicator of the logging potential of a system. While HDFS and Spark generate more than 200,000 logs per hour, some systems are far less talkative : Linux and Apache record less than 10 logs per hour. This heterogeneity can be linked to storage constraints, system criticality or system complexity. For the OBIS, all the application logs are generated during the exact same period of time. However, the complex storage management policy leads to an important heterogeneity in the temporal availability of logs.

Log datasets represent high volume of data. The available datasets are not necessarily representative of the logging potential power, since it depends on the recording time, itself depending on storage constraints (Fu et al., 2014). The actual logging power of systems is measurable by the important logging frequency. This tremendous amount of data available, during a specific time, considerably complicates the task of performing a local diagnosis manually (Lin et al., 2016). Both He et al. (2016a) and Zhu et al. (2019) claim that even tools like grep and search are inefficient with such big data. Hence, any manipulation of log data shall be designed to be automated and computationally efficient.

Log format. The state-of-the-art execution log datasets contain an important heterogeneity in their format. Figure 1.2 presents some examples of logs from the different datasets that highlight the diversity of formats. For each dataset, the full log line is generally partially structured, with a fix set of fields, mostly logged with the same format, and separated by a unique separator character (commas, blank spaces...). Yet, the different datasets do not necessarily have the same fields recorded, mainly because they have different domain information to log. In Figure 1.2, all the studied datasets record the timestamps, but all with different formats. Contrary to the two others, HealthApp (Figure 1.2a) does not provide any criticality level information, but instead a component (`Step_SPUtils`) and a PID (`30001212`). Another formatting distinction lies in the separation character : the separator character of Apache (Figure 1.2c) and Spark (Figure 1.2b) logs is a blank space, while HPC logs are separated by the string "-", and HealthApp logs have a very compact format, using a pipe ("|") as separator.

Some important heterogeneity is also observable within the same dataset. Even though it is rare, some elements of the structure might vary from one log to the other, as in Figure 1.2c, where the last Apache log has a `client` attribute that the two previous logs do not have. However, the main source of intra-dataset heterogeneity concerns the content part, since it is a free-form text. While most of the content are short messages, we observe that one of the Spark logs, presented in Figure 1.2b, is composed of a full error report, that extends over several lines. The second log of HPC (Figure 1.2d) contains, as the content part, another complete log message. A similar heterogeneity

```
20171223-22:15:32:145|Step_SPUtils|30002312| getTodayTotalDetailSteps=1514038440000##7011##...
```

(a) Healthapp : An extract of log, with pipe (|) separator

```
16/04/07 11:40:11 INFO yarn.ExecutorRunnable: Setting up ContainerLaunchContext
16/04/07 11:40:11 INFO yarn.ExecutorRunnable:
===============================================================================
YARN executor launch context:
  env:
    CLASSPATH -> PWD<CPS>PWD/__spark__.jar<CPS>...
    SPARK_LOG_URL_STDERR -> http://mesos-slave-14:8042/node/containerlogs..
    SPARK_DIST_CLASSPATH -> /usr/local/hadoop/etc/hadoop:/usr/local/hadoop/share/hadoop/...
    SPARK_YARN_STAGING_DIR -> .sparkStaging/application_1448006111297_0138
    SPARK_YARN_CACHE_FILES_FILE_SIZES -> 109525492,355358,44846
    SPARK_USER -> curi
    SPARK_YARN_CACHE_FILES_VISIBILITIES -> PRIVATE,PRIVATE,PRIVATE
    SPARK_YARN_MODE -> true
    SPARK_YARN_CACHE_FILES_TIME_STAMPS -> 1460000445562,1460000445715,1460000445754
    PYTHONPATH -> PWD/pyspark.zip<CPS>PWD/py4j-0.9-src.zip
    SPARK_LOG_URL_STDOUT -> http://mesos-slave-14:8042/node/containerlogs...
    SPARK_YARN_CACHE_FILES -> hdfs://10.10.34.11:9000/user/curi/.sparkStaging/...
  command:
    JAVA_HOME/bin/java -server -XX:OnOutOfMemoryError='kill %p' -Xms38912m -Xmx38912m...
===============================================================================
```

(b) Spark: A classic log and an extract of a log containing a full error report

```
[Sun Jul 31 04:08:59 2005] [notice] workerEnv.init() ok /etc/httpd/conf/workers2.properties
[Sun Jul 31 04:08:59 2005] [error] mod_jk child init 1 -2
[Mon Aug 01 05:28:21 2005] [error] [client 210.125.126.191] Directory index forbidden by rule...
```

(c) Apache : 2 logs without the "client" field and 1 with the client field

```
(1) 2015-07-29 17:41:41,649 - INFO  [main:QuorumPeer@959] - initLimit set to 10
(2) 2015-07-29 17:42:28,550 - INFO  [...] - Server 2015-07-29 17:42:59,195 - INFO  [WorkerReceiver
[myid=1]:FastLeaderElection@542] - Notification: 3 (n.leader), 0x0 %(n.zxid), 0x1 (n.round), LOOKING
(n.state), 3 (n.sid), 0x0 (n.peerEPoch), FOLLOWING (my state)
```

(d) HPC : A classic log and a log which content is another full log

Figure 1.2 – Example of the diversity of log format on the reference datasets. Specific fields are colorized : criticality level, application/service, event type and content.

is observed on the OBIS logs.

The flexibility offered to the developers in terms of content to log (Zhu et al., 2019) results in important quality issues and a great diversity among both the datasets and among the logs of a unique dataset. The diversity of information logged (with optional fields) and of the free-text content part constitute a challenge for the automation of knowledge extraction in log datasets. In addition, the heterogeneity of log format requires important generalization efforts to propose a universal method for the structure inference task.

Data quality and log modification. It is recognized that logs delivery comes in second place, as the core functions and main interest lie in the applications and services. Despite the potential of information buried in logs, their specification and implementation is often neglected in favour of greater effort on the core applications (Yuan et al., 2012).

Moreover, knowing what information is relevant to log, how it should be logged, and when, are still open questions (Fu et al., 2014; Zhao et al., 2017) : multiple anomalies can have different

ways to be detected, and it is difficult to know, a priori, which logs would be helpful to detect and understand such anomalies. The logging strategy is often subjective, linked to the developers' perception of the system (Liu et al., 2019b), which is necessarily partial. Besides, the logging system is generally submitted to important verbosity constraints, since over-logging might (i) cut back on the resources (CPU consumption, disk storage, I/O operations...), (ii) makes it difficult to extract the logs of interest in the huge amount trivial logs (Fu et al., 2014; Ding et al., 2014).

These challenges and constraints lead to a globally poor quality of log generation. Hence, logs are significantly modified after-thoughts. Yuan et al. (2012) and Chen and Jiang (2017) found out that the *churn rate* — the proportion of lines of the source code that are modified during a revision — of logging statements is twice as high as the churn rate of the overall source code. In addition, 18% of the revisions contain modifications of the logging statements, which is spectacularly high compared to the relative minority of the logging statements in the source code (1/30). Finally, we learn that most of the modifications of the log statements consists in updates and insertions . These modifications often happen when the core functions are modified, for consistency. Yet, 33% of the modifications are simply performed after-thoughts, without core functions updates.

These studies confirm that (i) the initial log quality is poor and needs updates, (ii) the developers are interested in improving the log quality. Besides, these frequent updates of the logs prohibit any manual or key-word-oriented treatment of the database. It especially prevents the maintenance of a lexicon of log templates, to infer the event types based on regular expressions. As a result, most of the log manipulation studies concern automated solutions.

To conclude, log datasets have huge sizes, important heterogeneity, poor data quality, and change fast. Hence, all the manipulation performed on log data should be automated, including the structure inference and the anomaly detection. In the next subsection, we present the information available to perform anomaly detection, and deduce the necessary structure inference steps.

1.2.3 Information available for anomaly detection

This subsection presents the information concepts that are available within the logs datasets and that are commonly used to perform anomaly detection. We describe their potential for the anomaly detection task, and study their availability in the logs, or the effort required to retrieve them.

Event type. The event type of a log is a crucial information : since the content parts are free texts, with highly heterogeneous formats, they are difficult to manipulate as-is. Instead, the event type knowledge can summarize the information being logged. Indeed, each event type is supposed to correspond to a single statement in the source code. Since the number of such statements is finite and limited, event types constitute a much more concise representation than the — potentially infinite — set of generated logs (Fu et al., 2009a). As a consequence, the event type is often used as the data representation of logs for log mining tasks, especially for the anomaly detection (Xu et al., 2010; Du et al., 2017).

Unfortunately, this information is seldom available as a field in the raw logs. In the OBIS, the logs of only one of the two components contain event types, while none of the reference dataset provides the event types of its logs as a field. Yet, for research purpose, the LogPAI (Zhu et al., 2016)

Dataset	Criticality level						Mean size of content	#event types (2k)	seq. id
Mac	NA						98	341	NA
Windows	Info			Error			192	50	NA|
	100.00%			0.00% (150 logs)					
Linux	NA						40	118	NA
OpenStack	Info	Warning		Error	Critical		100	43	VM
	98.41%	1.50%		0.09%	0.00% (2 logs)				
HDFS	Info			Warning			68	14	Block
	96.75%			3.25%					
Zookeeper	Info		Warning		Error		47	50	NA
	99.92%		0.04%		0.04%				
Spark	Info		Warning		Error		49	36	NA
	33.96%		65.22%		0.83%				
Hadoop	Info	Warning		Error	Fatal		149	114	NA
	93.38%	6.32%		0.29%	0.01%				
Thunderbird	NA						42	149	NA
BGL	I	W	E	F	S	Fl	48	120	NA
	78.68%	0.49%	2.37%	18.02%	0.40%	0.04%			
HPC	Info		Warning		Error		26	46	NA
	96.55%		2.72%		0.73%				
Android	I	W	E	F	D	V	49	166	NA
	50.57%	5.34%	10.42%	0.01%	33.37%	0.29%			
Healthapp	NA						47	75	NA
OpenSSH	NA						43	37	NA
Apache	Notice		Warning		Error		28	6	NA
	26.45%		0.32%		73.23%				
Proxifier	NA						84	8	NA

Table 1.2 – Main figures on field values in the reference datasets. The NA value indicates that the filed is not defined in the dataset.

project manually labelled subsets (2000 logs) of the log datasets and attributed an event type to the logs, considered as ground truth.

In Table 1.2, the column #event types (2k) represents the number of event types in the 2000-log subsets. In average, 2000-log subset contains 85 different event types, and some datasets exceed a hundred of event types (Mac, Android, Thunderbird). These figures are particularly high considering the relatively small samples considered; He et al. (2016a) points out that the number of event types increases with the number of log messages. However, some datasets only count a few types of logs (HDFS, Apache, Proxifier), which highlights the heterogeneity in terms of variety of information logged. This heterogeneity is also visible within the OBIS logs, among the applications and services.

Besides, the event type labelling is generally arguable. For instance, according to LogPAI labels, the following logs have two different event types, although their content parts are syntactically very similar :

```
[07.27 10:22:39] chrome.exe *64 - t12.baidu.com:80 open through proxy proxy.cse.cuhk.edu.hk:5070 HTTPS
[10.30 17:02:17] putty.exe - 183.62.156.108:22 open through proxy socks.cse.cuhk.edu.hk:5070 SOCKS5
```

Syntactically, these two logs could also be gathered within the same event types, forming a less homogeneous group with more logs. The hierarchical nature of the textual content is indeed a source of non-determinism. It is even more appreciable in the OBIS logs, where the different applications and services are implemented by different suppliers, with different labelling strategies. Messaoudi et al. (2018) structure this difficult question by defining the trade-off between frequency — an event type shall gather as many logs as possible — and specificity — an event type should describe precisely the logs, with as few variable parts as possible. Generally, the final decision does not only rely on syntactic considerations, but rather on semantic ones, especially based on domain understanding of the logged information.

The event type presents an interesting power of data summarization, which is why it is the most commonly used data representation in the log mining research area. Since it is seldom available, it shall be deduced from the data. This task is however complicated by (i) the important number of event types in a dataset (ii) the high diversity in event types distribution between the datasets (iii) the non-determinism of the log grouping task.

Criticality level. The *criticality level* is a field designed by the developers, associated to an event type, that indicates the seriousness of a log. It often only takes a small amount of possible values (e.g. INFO, WARNING, ERROR, FATAL), which are often common between the systems (Yuan et al., 2012; Fu et al., 2014). This simple and uniform information facilitates both the implementation and the exploitation of this field. Anomaly detection can easily be performed by filtering the erroneous logs, to detect unexpected situations. As a result, the use of the criticality level is popular for direct failure detection (Burns et al., 2001; Lim et al., 2014; Lin et al., 2016); developers dedicate 60% of their logging statements to anticipate the monitoring of unexpected situations [8] (Fu et al., 2014).

8. We however argue that this result does not only depend on the logging strategy, but also on the development maturity. Indeed, logging statements, especially the unexpected ones, are often added as the error occurs. Hence the number of logging statements is likely to increase during the project development.

In Table 1.2, the Criticality level column depicts the proportion of each criticality level on the 10 log datasets (out of 16) that define it. Most of the them (8/10 datasets) contain a majority of nominal logs (Info, I, D, V and Notice), such as Windows, HDFS, HPC... with marginal proportion of exception logs (Warning, Error, Fatal, Critical ...). There is however a distinction between the datasets that only exceptionally captures unexpected behaviours, like HDFS or Zookeeper, and the ones where unexpected logs are minority but not rare : in BGL, Error and Fatal levels are represented in more than 20% of the logs, and Android contains more than 15% of abnormal logs (W, E, F). Finally, Apache and Spark adopt another strategy by logging mainly warning or error logs. The high logging frequency of Spark makes us think that this strategy is implemented for the sake of storage constraints : logging more informative logs in a systematic manner would probably exceed the logging storage capacity. The OBIS logs also contain criticality level, and show an important heterogeneity from one application/service to another.

Nonetheless, there are two main limitations to the anomaly detection based on the criticality level. First, it is not systematically available in log data (e.g. HealthApp, Mac, HPC). Second, the logs marked as erroneous only concerned anomalies that the developers anticipated. Filtering logs by criticality level is inefficient for the detection of new anomalies, and for the root cause analysis of existing anomalies. Since, unexpected logs are not self-explanatory, informative logs are also vital to improve system understanding and modeling; at least 81% of the developers estimate that these logs are of critical importance (Fu et al., 2014).

Categories of event types. The informative logs can contain important information on the execution state of the system, which can help to detect finer anomalies, or troubleshooting discovered ones(Vaarandi, 2003). In their study, Fu et al. (2014) divide the informative logs in two types : (i) the logic-branch logs, that trace the code execution path when entering if/else conditions, and represent 16% of the logging statements (ii) the observing points, that gather main steps of code execution (excluding the logic-branches) and regular value monitoring of system values, which account for 24% of the logging statements. Being able to automatically distinguishing these logs enables to apply different treatments for the anomaly detection.

Nonetheless, the study only relies on the source code, which is often available, making the classification inapplicable to most of the log systems. Instead, in the following, we distinguish two types of informative event types :

— The on-event logs describe the path execution of the system. The presence of such a log is the main information : the event type representation of the log is self-explanatory. In the anomaly detection, the logs are represented by their event types;

— The periodic logs, that happen at regular temporal intervals to provide the current monitored values of the system. The values of the rendered parameters are resourceful. In the anomaly detection, the logs are represented by the values that are monitored. The event type can also be used for anomaly detection, to detect an absence or an irregularity in the periodicity.

This distinction can therefore help to improve the representation of logs passed to the anomaly detection method. For each event type, we can determine if it matches a regular monitoring of information by studying the regularity of the time separating two consecutive logs.

The sequence_id. Some log datasets define an optional *sequence_id*, an identifier of a *sequence* (e.g. the name of a file on which several operations are performed, traced in several logs). The definition of a sequence depends on the domain context, and does not systematically make sense. When it is defined, the sequence_id often needs to be retrieved within the content part of the logs. This identifier is of great importance for some anomaly detection methods (Du et al., 2017; Meng et al., 2019) that perform the anomaly detection on each sequence individually, which generally corroborates domain knowledge. It also enables to trace the exchange of some elements from one entity to another in complex, parallel systems. However, only 2 datasets out of the 16 understudied datasets define an identifier (column seq. id in Table 1.2). Moreover, the definition of sequence_id is often partial and does not concern all the logs of the datasets, because some logs are contextual and do not paired to a specific sequence. In **?**, we evaluated that 60% of the OpenStack logs were contextual, i.e. did not have a sequence_id. Relying on the presence of the sequence_id to split the logs into sequences would lead to the removal of 60% of the data. In the OBIS logs, there is no global sequence identifier, yet, locally, some applications present an identification mechanism for some event types. Contrary to the event type, which can always be defined and inferred, the sequence_id absence often means that its definition would not have any domain sense. Hence, we estimate that relying on the sequence_id for a log mining task is not recommendable.

From this subsection, we conclude that the event type is the most reliable information available to represent logs. The criticality level is used to extract erroneous logs, which enables a fast diagnosis of predefined anomalies. Yet, it cannot detect new anomalies, and is insufficient for troubleshooting, which requires the investigation of non-erroneous logs. These latter are divided in two types : the on-event logs, that trace the execution steps, and the periodic logs, that regularly monitor some system variables. For both types, the event type representation is commonly used for log anomaly detection. Yet, detecting the periodic logs and extracting the values of the monitored variable is another source of information that can be used for anomaly detection. Finally, the sequence_id can divide the log dataset in meaningful subsets, but is seldom available.

1.2.4 Data analysis conclusions

Table 1.3 summarises the different challenges from the dataset of logs. Firstly, log data generally constitute massive datasets, with high generation frequencies. 7 of the 16 datasets contain more than 1 million logs, and 6 systems generate logs faster than 10 thousand logs an hour. Since log datasets change fast, any log processing treatment should be automated. Yet, the automation of log processing is challenging because logs are heterogeneous (in their formats, sources, available fields...) and have low data quality.

To automatically detect anomalies, the content part contains the richer source of information, and is systematically available, compared to the criticality and sequence_id (available in respectively 11 and 2 datasets out of 16, in Table 1.3). The event type is often used as a concise representation of the content but it is not directly available in logs (see Table 1.3). More knowledge is buried in the content, especially in the variable parts of the regular logs. The extraction of both the event types and the variable parts of the logs require the inference of structure in logs. The automation of such process is especially challenging when log datasets contains a wide variety of different event types : 6 datasets out of 16 have more than 100 event types in their 2000-log subset (Table 1.3).

In the following section, we therefore describe the traditional steps to infer the structure within

Dataset	Huge size	High gener. rate	High #event types	Event type unavailable	Sequence undefined	Criticality undefined
Operating System						
Mac	✘	✘	✓	✓	✓	✓
Windows	✓	✓	✘	✓	✓	✘
Linux	✘	✘	✓	✓	✓	✓
Distributed System						
OpenStack	✘	✘	✘	✓	✘	✘
HDFS	✓	✓	✘	✓	✘	✘
Zookeeper	✘	✘	✘	✓	✓	✘
Spark	✓	✓	✘	✓	✓	✘
Hadoop	✘	✓	✓	✓	✓	✘
Supercomputer						
Thunderbird	✓	✓	✓	✓	✓	✓
BGL	✓	✘	✓	✓	✓	✘
HPC	✘	✓	✘	✓	✓	✘
Mobile system						
Android	✓	✓	✓	✓	✓	✘
Healthapp	✘	✘	✘	✓	✓	✓
Server application						
OpenSSH	✘	✘	✘	✓	✓	✓
Apache	✘	✘	✘	✓	✓	✘
Standalone Software						
Proxifier	✘	✘	✘	✓	✓	✘
Total						
	6	7	6	16	14	5

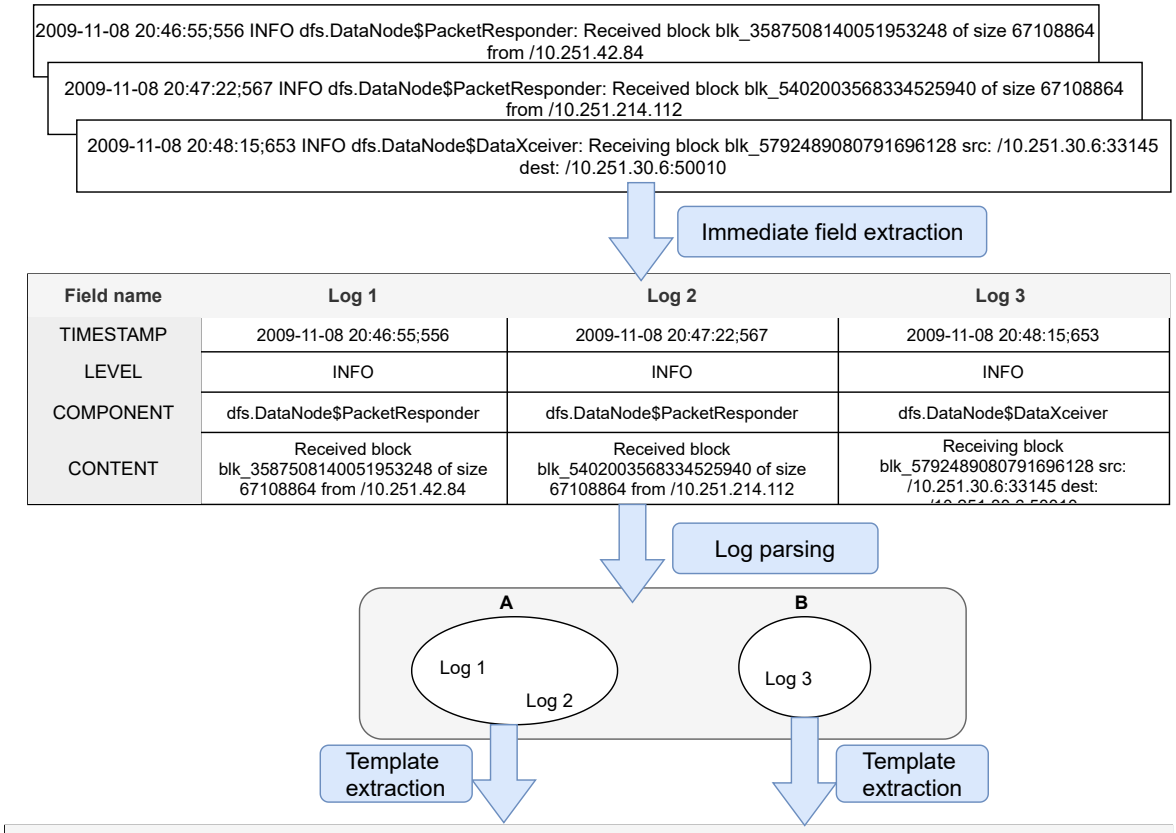
Table 1.3 – Summary of the challenges observable in the reference datasets. Huge size : the dataset has more than 1M logs. High gener. rate : the generation rate of the dataset is greater than 10k logs/hour. High #event types : the 2000-log subset contains more than 100 event types.

log datasets. This section also presents the limits of the main state-of-the-art techniques before introducing our contribution in this part.

1.3 Structure inference in logs

Retrieving the structure of logs is a mandatory task in order to exploit them for anomaly detection. Figure 1.3 presents the classical steps of structure inference in logs. While some features can be immediately extracted (e.g. timestamp, criticality level, content...), the structural information of the content part is buried in the log datasets. The most common way to retrieve this structure is to use a *log parser*, that aims at attributing to each log an identifier of its group, the event type. In the Figure 1.3, the two first logs are attributed to the event type A, while LOG 3 is attributed to the event type B. Each event type is a concise identifier of the group of logs.

Once the logs are regrouped, a finer structure can be extracted from each group of logs. It



Field name	Log 1	Log 2	Log 3
TIMESTAMP	2009-11-08 20:46:55;556	2009-11-08 20:47:22;567	2009-11-08 20:48:15;653
LEVEL	INFO	INFO	INFO
COMPONENT	dfs.DataNode$PacketResponder	dfs.DataNode$PacketResponder	dfs.DataNode$DataXceiver
CONTENT	Received block blk_3587508140051953248 of size 67108864 from /10.251.42.84	Received block blk_5402003568334525940 of size 67108864 from /10.251.214.112	Receiving block blk_5792489080791696128 src: /10.251.30.6:33145 dest:

Field name	Log 1	Log 2	Log 3
TIMESTAMP	2009-11-08 20:46:55;556	2009-11-08 20:47:22;567	2009-11-08 20:48:15;653
EVENT TYPE	A	A	B
TEMPLATE	Recevied block <*> of size <*> from /<*>	Recevied block <*> of size <*> from /<*>	Receiving block <*> src: /<*> dest: /<*>
PARAMETERS LIST	['blk_3587508140051953248', '67108864', '10.251.42.84']	['blk_5402003568334525940', '67108864', '10.251.214.112']	['blk_5792489080791696128', '10.251.30.6:33145', '10.251.30.6:50010']

Figure 1.3 – Overview of the structure inference steps. Some fields can be immediately extracted (e.g. timestamp, criticality level, content...). To extract the event type, the log parsing task regroups the logs of similar contents, and associates an event type. For each group, the template extraction phase finds the fix and variable parts to deduce (i) the template of the group, (ii) the parameters of each logs.

generally consists in distinguishing which parts of the logs constitute the template of the groups, and which parts are variable among the logs of the groups. In Figure 1.3, the log group of event type A is associated to the template `Received block <*> of size <*> from /<*>`, where the `<*>` characters represent the places of the variable components of the log content. While this template is defined for the whole group, each log has a specific set of parameters, the value taken by the variable parts in the log. In Figure 1.3, for the first log, the three variable parts take the values `blk_3587508140051953248`, `67108864` and `10.251.42.84`.

In this section, we present in detail the two consecutive tasks, give an insight of the limits of the existing state-of-the-art methods, and introduce our proposition for the structure inference task.

1.3.1 Log parsing

Historically, event type inference used to rely on manually maintained sets of regular expressions (*regex*), matching event templates (He et al., 2017). Even though this approach promises an interesting integration of domain knowledge, it has now become an incredibly time-consuming and error-prone task (Zhu et al., 2019), mainly due to the aforementioned huge size of logs (Mi et al., 2013). Moreover, the fast changing character of log datasets discourages the maintenance of a fix list of event types with associated templates (Yuan et al., 2012). Some work has been proposed to automatically retrieve the event type from the source code (Nagappan et al., 2009; Xu et al., 2010). Yet, the latter is seldom available, especially in the case of third-party system usage. Therefore, the event type extraction needs to be automated and data-driven.

Log parsers are automated data-driven solutions to infer the event types of logs (Mi et al., 2013). They create groups of similar logs, that share a common underlying structure. They are recognized to be an unavoidable step of data processing for log mining (He et al., 2017; Du et al., 2017; Zhu et al., 2019). Not only is log parsing mandatory to obtain the input representation of log mining algorithm, but it is also crucial to perform this task as accurately as possible : He et al. (2016a) show that the accuracy of log parsing has important impacts on the results of the log mining task. For this reason, the log parsing question has raised both academic and industrial interest, and is still an open topic, with very recent reviews and new propositions (He et al., 2018a; Zhu et al., 2019; Dai et al., 2020).

The existence of a common underlying structure and the high volume of log datasets motivate log parsers to exclusively rely on syntactic analysis, discarding any semantic aspect. Yet, the meaning of log statements influences the labelling of log groups, introducing important syntactic heterogeneity among the groups' profiles : as shown before, very syntactically similar logs can be associated to different event types (Proxifier example), while logs from a unique group can have important syntactic diversities (expression of time, plural forms...). The ability of a log parser to modulate its behaviour to different grouping profiles is therefore key to insure its robustness. To achieve this modulation, most of the existing log parsers (Du and Li, 2016; He et al., 2017) are parametric : they rely on hyper-parameters which can be tuned to propose different versions of parsing groups. The results of the benchmark of Zhu et al. (2019) show the superiority both in accuracy and robustness of these parametric methods over non-parametric ones (Messaoudi et al., 2018; Dai et al., 2020). Hence, the robustness of a log parser across different types of log data depends on its ability to modulate, which is enhanced by the use of hyper-parameters.

The literature contains only few examples of such adaptive log parsers (Makanju et al., 2012; Du and Li, 2016; He et al., 2017). These modern techniques often propose an online functioning, which makes them eligible for online log mining applications (Du et al., 2017). These parsers rely on strong syntactic assumptions; e.g. a group can only contain logs of the same *length* (number of words) (Makanju et al., 2012; He et al., 2017). Yet, Zhu et al. (2019)' study points out that some reference datasets are challenging to parse because they do not comply with the strong syntactic assumptions of the various state-of-the-art methods. As a result, these datasets do not benefit from any adapted robust log parsing solution, while they include important log data, such as OpenStack, a reference for the evaluation of anomaly detection (Du et al., 2017).

Finally, frequent pattern mining methods (Vaarandi, 2003; Nagappan and Vouk, 2010a; Dai et al., 2020) offer a more text-driven representation. These methods are based on the soft and

popular assumption that frequent patterns are likely to be fix parts of the template and offer interesting perspective to enhance flexibility around the syntax. Nonetheless, the existing frequent pattern mining techniques are not robust and show very heterogeneous results (Zhu et al., 2019) either because they are parameter-free (Dai et al., 2020) — and lack of modulation power — or because their functioning around the frequent patterns still rely on strong assumptions; e.g. frequent tokens always appear at the same position in the logs (Vaarandi, 2003).

In conclusion, log parsing is a major research issue since it is a key step for log mining. It has been deeply studied on the academic side, and adopted on the industrial one. Among the existing log parsers, the parametric ones proved to be superior to non-parametric ones, which raises the question of the hyper-paramaters optimization. The parametric log parsers rely on strong assumptions that can turn out to be violated in some important log datasets. A frequent pattern mining approach can offer a more data-driven representation, yet none of the existing methods present satisfactory results. We also mentioned that an online functioning of the log parser makes it eligible for the recent issue of detecting anomalies in online systems.

1.3.2 Template extraction

Each group created by the log parsing task is represented by an event type, an abstract identifier, which enables the direct application of most log mining tasks (Fu et al., 2009a; Du et al., 2017; Meng et al., 2019). In Figure 1.3, the logs LOG 1 and LOG 2 are represented by the identifier A. However, this concise representation does not give any insight on the meaning of the messages logged with the event type. Especially, they do not give access to the parameter values of logs. Moreover, since most of the log parsers are generated offline, and based on the data, the event type information is not sufficient to help the parsing of new arriving logs. It is therefore common to associate a log parser to a template extraction method (Fu et al., 2009a; Makanju et al., 2012; Du and Li, 2016). This task consists in retrieving, in a group of logs, the parts that are fix, and constitute the event template. These templates are therefore in bijection with the event types (Makanju et al., 2012). In Figure 1.3, the logs LOG 1 and LOG 2 can be represented by their template `Recevied block <*> of size <*> from /<*>`. The event template can be used as a readable representation of the information being logged. It can also be considered as the representative of the logs : when new logs arrive, they can be checked for template matching, and accordingly associated to the existing groups (Fu et al., 2009a). For instance, in the context of Figure 1.3, if a new log `Recevied block blk_-7451501814005195324 8 of size 54875632 from /10.251.78.10` arrives, a simple regular expression check with the formally stated template would be sufficient to parse this log in group A.

The template extraction also enables the deduction of the variable parts of logs. These variable parts contain valuable run-time information that can be used for system monitoring (Fu et al., 2009a; Xu et al., 2010). In accordance with our definition of logging profiles (Section 1.2.3), we argue that the event type of regular logs is a poorly-informative representation : such logs are expected to happen at regular temporal intervals, hence, their appearance do not bring any additional information — contrary to their absence. Yet, such logs are often implemented in order to record important system values, that constitute the variable parts of the log content. In other words, the variable parts are actually the only informative data when it comes to regular event types.

The pervasiveness of the template extraction task is not only supported by its presence in most of the log parsing articles, but also by its importance in the evaluation part of these studies : both Messaoudi et al. (2018) and Dai et al. (2020) evaluate the accuracy of the extracted template rather than the traditional grouping evaluation (He et al., 2017; Zhu et al., 2019). It highlights that these works pay a great attention to the template extraction, and judge its accuracy as a requirement for a structure inference proposition.

We mainly distinguish two types of template extraction approaches : the ones embedded in the log parsing process, and the ones performed as a post-processing. Most of the online methods (Du and Li, 2016; He et al., 2017) propose an embedded template discovery : since the groups are dynamically created — as though each log was a new coming one — the algorithms need a systematically available representative of the existing groups. Even though this approach offers the advantage of dealing with the two tasks simultaneously, they are often computationally inefficient, since the templates are systematically challenged and reevaluated (He et al., 2018a). Moreover, the proposed methods are not exportable to external usage. For instance, it could not be applied to an already-parsed set of log groups. The existing post-processing template extraction methods (Fu et al., 2009a; Tang et al., 2011; Makanju et al., 2012) are performed afterwards. Yet, they often rely on the same definitions and calculations than the log parser, and are therefore not independent of the parsing method, which limits their potential to be exported to other situations (e.g. if another log parser is used, if more accurate or efficient; or if logs are already parsed).

In addition to this lack of practicality, Dai et al. (2020) report that the existing methods show mitigated results in the accuracy of template extraction. The literature therefore lacks an accurate and efficient method for template extraction that would be independent of any log parsing method, and could therefore be used as the post-processing step of the log parsing.

1.4 Contributions and part organization

Inferring the structure of log datasets is a necessary task to both gain insights on the behaviour of the logs, and be able to retrieve the event types, which are used to represent the logs in log mining tasks. The inference of structure often consists in (i) parsing the logs, to retrieve the event types, (ii) extracting the template of the inferred groups of logs. This inference is however challenging, since it should be robust to the heterogeneity of the logs.

To answer this challenge, we propose a two-fold contribution. We first introduce a new parametric log parser, named METI*N*G that stands for Modular Event Type Inference with *N*-Grams. Firstly introduced in (**?**), METI*N*G is parametric and has an important modulation power, thanks to its high sensitivity to its two hyper-parameters. Our method uses the frequent pattern mining approach in a new flexible way, by extracting frequent n-grams, without any additional syntactic assumptions. METI*N*G is eligible for online parsing : the groups are organized in a tree-like structure, which can be run through in an online fashion to parse new arriving logs. We then propose a template extraction method for log data that is independent of the log parser and computationally efficient. To exhibit the generalization power of our approach, we study its adaptation to the task of stemming, which can be considered as inferring the structure of textual data. We introduce RFreeStem (**??**), a multi-language stemmer that provides a rule-free and language-independent stemming solution that compete with Porter (1980), the state-of-the-art reference.

In this part, Chapter 2 presents the main state-of-the-art work concerning both the log parsing task and the associated template extraction, and highlights the limits of the existing methods. We then detail the principle and the implementation of our solutions in Chapter 3. This chapter formalizes the problem of structure inference, and presents our solution, led by some concrete examples on the understudied log datasets. We also propose an online version of the log parser. Chapter 4 assess our methods on the reference datasets. With a comprehensive evaluation on 16 datasets, we show that METI*N*G is robust and globally outperforms the state-of-the-art methods. METI*N*G is also able to tackle challenging datasets, on which existing methods fail, with some impressive accuracy improvements. Finally, Chapter 5 details the application of our algorithm to the stemming of textual data in text mining.

2 Related work

This chapter presents the related work concerning the inference of structure for log datasets. The section 2.1 details the reference methods for log parsing techniques, while section 2.2 presents the propositions to extract the templates of the formed groups. In section 2.1, we review the different methods that assess the structure inference algorithms of the logs.

2.1 Log parsers

Log parsers are regarded as the best manner to retrieve event types, in an automated and data-driven way. Since the late 2000s, many articles proposed new techniques to infer log event types (Fu et al., 2009a; Tang et al., 2011; Messaoudi et al., 2018). We present here the different types of existing log parsers.

2.1.1 Optimization methods for log parsing

Some log parsing methods (Fu et al., 2009a; Tang et al., 2011; Hamooni et al., 2016; Messaoudi et al., 2018) search for the best group partition by optimizing a partition criteria. Among them, most are inspired from the classical clustering techniques. With a quantitative definition of distance, these algorithms calculate a pairwise square distance matrix to split the logs into groups. LKE (Fu et al., 2009a) computes the pairwise weighted edit distances and clusters the logs accordingly, before splitting the clusters again, based on a Longest Common Sequence (LCS) threshold. This matrix calculation demands high computational resources ($\mathcal{O}(n^2)$, where n is the number of logs), and thus cannot scale to large log databases. In their evaluation, He et al. (2016a) estimate the limits of LKE at 4 million logs for the BGL dataset, and 10 million logs for the HDFS dataset. Moreover, He et al. (2016a) argue that the LKE clustering process is aggressive : clusters are grouped based on the smallest distance of their logs. Hence, if only two logs from two clusters are considered as similar, the full clusters are merged.

To overcome the complexity issue, other distance calculations can be applied : LogSig (Tang et al., 2011) optimizes a cluster homogeneity indicator, based on a LCS distance, while LogMine (Hamooni et al., 2016) calculates the distance between a log and a group representative, namely the group template. Instead of a $\mathcal{O}(n^2)$ complexity, where n is the number of logs, these methods rather perform a $\mathcal{O}(n \cdot m)$ complexity, where m is the number of created groups. He et al. (2016a) remark that it can still lead to important execution times. Moreover, both He et al. (2016a) and Zhu et al. (2019) agree that LogSig is inaccurate, especially for datasets containing important number

of groups. LogMine also presents mitigated accuracy results according to Zhu et al. (2019).

We conclude that, despite a slight improvement in computational capacity, these solutions still show mitigated efficiencies and poor accuracy results (Zhu et al., 2019). Generally speaking, He et al. (2016a) shows that clustering methods do not scale well on large datasets, which has several problematic consequences : (i) they are not applicable on large datasets, (ii) they are not efficient on intermediate-size datasets, and (iii) the optimization of their parameters is time-and-resource-consuming, which might partially explain their limited accuracy results.

Alternatively, MoLFI (Messaoudi et al., 2018) models the log parsing task as an optimization problem. MoLFI tries to optimize two conflicting objective functions : the created log groups should maximize the sizes of the groups, while maximizing their homogeneity. Besides, MoLFI is a parameter-free method, which avoids the computationally expensive task of optimizing parameters. Nonetheless, the optimization functioning still presents an important computational complexity, making the method inefficient. In addition, MoLFI shows very heterogeneous and globally inaccurate results (Zhu et al., 2019), which we suspect to be caused by the lack of flexibility, due to the absence of hyper-parameters.

2.1.2 Heuristics based on strong syntactic assumptions

To cope with the expensive computational costs, most methods rely on strong syntactic assumptions. AEL (Nagappan and Vouk, 2010b) relies on a set of regular expressions to determine the format of variable parts. This method is therefore dependant on the quality of the maintenance of such a list of regular expressions. Similarly to the maintenance of a log template lexicon, this maintenance is time-consuming and error-prone. IPLoM (Makanju et al., 2012) assumes that logs of different lengths cannot be gathered, and achieves a first split based on log lengths. On the one hand, this assumption is highly convenient, since it allows to align the words of the logs, and form columns. These columns are then used to further partition the logs : (i) columns are analyzed to find fix words, that have a unique value, (ii) relationships between columns are analyzed (one-to-one, one-to-many, many-to-many) in order to extract other fix parts. On the other hand, the strong assumption that logs of the same group must have the same length is sometimes violated (see Proxifier dataset). IPLoM however exhibits an overall high accuracy, for the datasets that validate this hypothesis, and an interestingly low algorithmic complexity (linear) (He et al., 2016a; Zhu et al., 2019; Dai et al., 2020). Hence, IPLoM has long been acknowledged as the state-of-the-art reference, before being challenged by more modern, online methods.

The most modern methods focus on online solutions, where the groups are generated on-the-fly. The online functioning of these methods makes them eligible for online log mining tasks, such as online anomaly detection (Du et al., 2017), or online performance analysis (Chow et al., 2014). The online aspect is also especially suitable for fast-changing datasets : the generated groups are automatically adapted to new types of logs, while offline methods would need a whole recalculation. Spell (Du and Li, 2016) progressively generates groups of logs using LCS (Longest Common Sequence) as a distance between a new arriving log and the representative of an existing group. While Spell presents a flexible representation of the logs, which fits the textual aspect of logs, it exhibits a mitigated time complexity of $\mathcal{O}(n \cdot m)$. We also found examples of log datasets where considering the LCS as an indicator of similarity can lead to judgement errors (e.g. OpenStack). SHISO (Mizutani, 2013) is another online method that proposes an iterative version of LEARN-

PADS (Fisher et al., 2008), an automated framework that learns rules to describe the logs. This method however presents limited accuracy (Zhu et al., 2019). Both Drain (He et al., 2017) and LenMa (Shima, 2016) steer the research of existing groups with strong assumptions : logs are firstly parsed according to their lengths. For LenMa, another assumption is that the variable words, situated at the same position in the template, should have a similar number of letters. LenMa has a rather satisfying overall accuracy, but reaches extremely low results when the previous assumption is violated (Zhu et al., 2019). Moreover, LenMa presents a limited computational efficiency.

Finally, Drain is the current state-of-the-art reference. Along with the unique length assumption, Drain assumes that the fix parts of the templates are localized at the beginning of the logs. Based on these two assumptions, Drain builds a fixed-length tree : (i) at the first level, nodes contain all the logs from the same length, (ii) these nodes are split according to their first word, then according to their second ... this splitting iteration is controlled by a parameter d, (iii) if the final groups obtained are homogeneous enough — based on a distance calculation to a representative — they are validated as final groups, otherwise, all the logs are split into singletons. This very straightforward process enables Drain to be one of the most computationally efficient method (He et al., 2016a; Zhu et al., 2019). Despite, its overall high accuracy, its strong assumptions are however violated for some of the reference datasets (e.g. Proxifier, OpenStack).

Heuristic-based log parsers generally offer an important improvement of algorithmic complexity. Some of the methods combine this efficiency with high accuracy scores. Moreover, online methods offer the possibility to be used as a part of online log mining methods. Nevertheless, these heuristics are based on strong syntactic assumptions, that do not comply with all the state-of-the-art datasets (Zhu et al., 2019), leading to some gaps in the treatment of the reference datasets.

2.1.3 Methods based on the frequent pattern mining assumption

Alternatively, the frequent pattern mining approach offers a more flexible representation of log syntax, and might avoid the introduction of strong syntactic assumptions. SLCT (Vaarandi, 2003) and its extension LogCluster (Vaarandi and Pihelgas, 2015), as well as LFA (Nagappan and Vouk, 2010a) and Logram (Dai et al., 2020) support the popular idea that frequent patterns (words, tokens, token pairs) are likely to be markers of event templates, whereas rare patterns are generally log parameters (i.e. variable parts). These algorithms run several passes through the logs, building a frequent pattern dictionary. With this linear functioning, these methods appear to be computationally efficient (Zhu et al., 2019). SLCT was the very first log parser proposition, and did not actually have the objective to generate a full partition of the logs, but instead, to detect the frequent patterns within the logs. As a result, SLCT and its extension LogCluster are the only methods that do not cover the whole datasets, and propose a partial log parsing. LogCluster extends SLCT in order to enable length variations of the variable parts.

LFA and Logram are parameter-free. LFA studies the frequency distribution of words and is also able to parse rare events. Logram is based on frequent n-gram mining, and generates a dictionary of 2-grams and 3-grams in order to deduce the dynamic words, focusing on the overlapping of different n-grams. To do so, Logram attempts to automatically determine a frequency threshold for n-grams, instead of specifying it as a hyper-parameter.

Despite their computational efficiency and the promising flexibility of frequent pattern rep-

resentation, these methods show disappointing accuracy results : they either integrate strong assumptions around the frequent patterns (e.g. fix tokens must have fix positions in logs), or are parameter-free. Moreover, they generally evaluate the frequent patterns once, for the global dataset. This often prevents the proper treatment of small groups of logs, since their fix parts are unlikely to be frequent at the dataset level. In addition, all of these methods are offline.

2.2 Template extraction

Most of the previously described log parsers also provide a method to extract the template from the created groups of logs. Indeed, this extraction provides a intelligible representation of the groups, and can also enable the separation between fix and variable parts of the log messages. We mainly distinguish two types of template extractions, in the aforementioned work : the ones that are created while the log parser is running, and the ones that are created by post-processing operations.

2.2.1 Template extraction during log parsing

For some of the understudied log parsers, the construction of the groups of logs relies on the creation and maintenance of group representatives. When a group representative is complete — meaning that all the fix parts are identified — it represents the template of the group. Most of the online methods create and update the template of the group during the process of building these groups. Spell (Du and Li, 2016) uses a group representative to build the groups in an online fashion, adding one log to a group at a time. This representative is therefore updated each time a log is added to the group, so that the current template is systematically available, at any time of the execution. The same observation can be noted for the online methods SHISO (Mizutani, 2013) and LenMa (Shima, 2016).

Moreover, some other log parsers are directly based on the retrieval of fix and variable parts. It is the case of all the log parsers based on the frequent pattern mining assumption, which aim at identifying the fix parts. Therefore, the template of each group is directly retrievable after the log parsing task is achieved. Logram (Dai et al., 2020) is also concerned, since its functioning consists in extracting the variable parts. Finally, MoLFI (Messaoudi et al., 2018) directly analyzes group template propositions, that are directly evaluated for the optimization task.

To conclude, these log parsing methods present the advantage of including the template extraction inside their main functioning, without the necessity of a extra post-processing step. Nevertheless, for the online methods, the systematic updates of the group representatives might degrade the computational efficiency of the methods — in (He et al., 2018a), an extension of Drain, a whole graph needs to be recalculated at each iteration. Moreover, the template extraction part of these log parsers is not exportable for external usage. For instance, it could not be applied to an already-parsed set of log groups.

2.2.2 Template extraction as a post-processing step

The other log parsers clearly identify the template extraction as an independent post-processing step. For the methods that adopt the assumption that log from the same group should have the same length, namely LKE (Fu et al., 2009a), IPLoM (Makanju et al., 2012) and Drain (He et al., 2017), the template extraction is straightforward : the logs can be aligned word-by-word, forming columns that correspond to the position of the words in the logs. Hence, the columns that have a unique word value are considered as fix tokens, and the rest of the words form the variable parts. Alternatively, LogSig (Tang et al., 2011) relies on the distance metric used during the parsing process in order to estimate a score for the words and determine the fix parts. In conclusion, these methods are highly dependent on the parsing task, since they use the same syntactic assumption and distance definitions. Therefore, they are also difficult to export (e.g. to compare with another log parser, or to use on already-parsed logs).

Generally speaking, Dai et al. (2020) report that the existing methods show mitigated results in the accuracy of template extraction. The literature lacks a template extraction method that (i) is independent of the log parsing task, implying that (a) it is a post-processing step (b) it does not rely on the same strong syntactic assumption, (ii) provides accurate templates, compared to a ground truth labelling.

2.3 Reviews and assessment of log structure inference

This section presents the related work around the evaluation of structure inference methods for log datasets. We first present the reviews and most common evaluation framework. We then detail the different metrics used to asses the structure inference. We finally describe the limitations of the existing methods.

2.3.1 Reviews and evaluation framework

As a core part of the log mining process, log parsing techniques have raised research interest, both in industrial and academic domains. Industrial tools (like Splunk [1]) provide specific solutions to log parsing, yet require extensive domain knowledge and present a poor generalization power. In turn, academic studies face the difficulty of getting hold of log datasets. Despite the scarcity of labelled logs, Zhu et al. (2019) managed to gather 16 different log datasets from different sources such as distributed system logs or mobile system logs. These datasets constitute the state-of-the-art reference (Fu et al., 2009a; He et al., 2017; Dai et al., 2020). Their tool named LogPAI (Zhu et al., 2016) also provides manual labels for extracts of 2000 logs. The availability of the labels (both the partition and the template of groups) enables the automated application of external measures to evaluate and compare the log parsers. The authors also implemented and released 13 of the previously mentioned state-of-the-art log parsers. They compare the accuracies of the log parsers, assess their robustness and study their computational efficiencies.

As another important work, He et al. (2016a) review the literature of log parsing and evaluate its impact on log mining. This former study compares 4 log parsers applied on 5 reference datasets.

1. https://www.splunk.com/

The authors compare the accuracy, robustness and computational efficiencies of the log parsers. They also evaluate the impact of the parsing on the anomaly detection task, by comparing the detection results obtained with the different structure inference versions, induced by each log parser. Apart from these two major comparative studies, most of the articles that propose a new log parser evaluate their proposition and compare it to the former log parsers, based on similar criteria. The next subsection censuses the criteria that are evaluated and the metrics used to evaluate them.

2.3.2 Evaluation metrics

We describe here the different indicators that are monitored to evaluate the performance of a log parser. Thanks to the previously described framework, we often have the ground truth of both the event type — corresponding to the partition of logs — and the template of each group, and each log.

Log parsing accuracy. The first criteria used to assess a log parser is its accuracy. Generally speaking, it consists in evaluating whether the partition formed by the parsing is equivalent to the partition of the ground truth. In other words, the accuracy evaluates whether the logs with the same event types are correctly parsed together. To measure this criteria, traditional precision, recall and F1-measure are often used (Nagappan and Vouk, 2010b; Tang et al., 2011; He et al., 2016a, 2017; Messaoudi et al., 2018). For a log of event type e, that was parsed to the group g, we can calculate :

— the *true positives* (TP) : the logs of event type e that are also parsed to group g;
— the *false negatives* (FN) : the logs of event type e that are not parsed to group g;
— the *false positives* (FP) : the logs which event type is not e but that are parsed to group g.

Accumulating the TP, FN and FP of all the logs, the *precision* is defined as the ratio $\frac{TP}{TP+FP}$ while the *recall* is the ratio $\frac{TP}{TP+FN}$. Finally, the F1-measure is the harmonic mean of the precision and recall : $\frac{2 \cdot precision \cdot recall}{precision+recall}$. Alternatively, Hamooni et al. (2016) studies the ratio $\frac{TP}{n}$, where n is the total number of logs. Zhu et al. (2019) however argue that these pairwise metrics tend to provide very high and smooth scores for most of the evaluated methods. Instead, the authors use the more rigorous metric of *parsing accuracy*, which calculates the ratio of well-parsed logs. A log is *well-parsed* if it exactly matches the group formed by its label : all the logs of event type e are parsed to g, and no log with a different event type is parsed to g. For instance, if a dataset contains 6 logs of event types $[E1, E1, E2, E2, E3, E3]$ that are parsed to the groups $[G1, G1, G2, G2, G2, G3]$, the accuracy score is $2/6$ since only the logs of event type $E1$ are well-parsed (the logs of event type $E2$ are parsed with a log of event type $E3$, while the last log of event type $E3$ is not parsed with all the logs of $E3$). This measure was first used in Du and Li (2016) and has been adopted to log parsing evaluation ever since (Zhu et al., 2019). We also used this metric in our work (**?**), and use it in the evaluation Chapter 4.

More rarely, some internal indicators can be used to measure the quality of the created groups. While *external* metrics, as the aforementioned ones, measure the similarity between the ground truth partition and the partition provided by the log parser, *internal* metrics rather provides a measure of the quality of the generated groups, and do not require labels. For instance, Nagappan and Vouk (2010b) measure the Shanon entropy of the created clusters. These metrics present the advantage to be applicable to any partition, without the need of labels. We noticed however that

such measures rely on a specific definition of similarity between logs, whose determination is a challenging question.

Extracted template accuracy. Alternatively, some articles focus on the accuracy of the template extracted, instead of the partition itself (Makanju et al., 2012; Hamooni et al., 2016; Dai et al., 2020). A first validation of the template extracted from a log parser can be to manually exhibit the results of some extracted templates, like in the article of Tang et al. (2011). Yet, the labelling provided by LogPAI (Zhu et al., 2016) also includes a template label for each log. This enables the automated calculation of indicators. Some articles opt for a strict comparison and count the number of logs which extracted template exactly match the label (Makanju et al., 2012; Dai et al., 2020), while Hamooni et al. (2016) have a more tolerant evaluation, where the accuracy is measured field by field inside the template.

Robustness of log parsers. Some articles evaluate the robustness of the two previous features. Zhu et al. (2019) evaluate and compare the robustness of the parsing accuracy for the understudied log parsers over several log datasets. The authors calculate the interquartile range to measure the steadiness of the results over the datasets. They also assess the robustness on change of volume by studying the influence of the size of data on the log parsing accuracy. Finally, Logram (Dai et al., 2020), that measures the accuracy of the group templates proposes to study the *agreement ratio*. For Logram, a dictionary of n-grams is built to detect the variable parts. This dictionary can be built on a smaller set of data, and used on the full dataset. The agreement ratio is defined as $\frac{acc(sub-sample)}{acc(full-data)}$, where $acc(sub-sample)$ is the accuracy obtained when the dictionary is learned on a smaller sample and applied on the full dataset, whereas $acc(full-dataset)$ is the accuracy obtained when the dictionary is learned and applied on the full dataset. This indicator actually evaluates the ability of the model to be learned on a smaller set of logs, and Dai et al. (2020) study the robustness of this indicator according to different sizes of the learning dataset.

Computational efficiency. The computational efficiency measures gather all the indicators that evaluate the algorithmic performance of the log parsers. Most of the articles that propose a log parser measure its efficiency, since it represents a guarantee not only of the rapidity of the method, but more importantly of the ability to be applied on huge datasets. Indeed, in their comparative study, Zhu et al. (2019) not only plot the time elpased for the algorithms to parse the datasets, but they also emphasis the limits of some algorithms which could not even parse some reference datasets. The most common indicator for algorithmic efficiency is the time elapsed for a log parser to complete the task (Tang et al., 2011; Hamooni et al., 2016; He et al., 2017; Messaoudi et al., 2018; Dai et al., 2020). This indicator is of course highly dependent on the execution environment, and size of data. It should therefore always be a relative measure, and should be used to compare one algorithm to another in the same environment. Alternatively, some articles evaluate the memory consumption (Nagappan and Vouk, 2010a; Hamooni et al., 2016). In our validation Chapter 4, we also evaluate the computational efficiency of the studied methods by measuring the execution time.

Impact on a log mining task. Since log parsing is a pre-processing task for log mining algorithms, it is common to assess log parsers by evaluating their impact on log mining tasks. Fu et al. (2009a) build a Finite State Automaton (FSA) based on the extracted event types and use it to detect execution errors and performance anomalies. The article of Drain (He et al., 2017) as well as the compartive study of He et al. (2016a) perform an anomaly detection task and census (i) the number of reported anomalies, (ii) the number of correctly detected anomalies, and (iii) the number of false alarms. These indicators are compared for datasets pre-processed with different log parsers, coming from the same dataset, with the same anomaly detection method applied. As a matter of facts, He et al. (2016a) conclude that the accuracy of the log parsers have a direct impact on the ability to detect anomalies : 4 % errors in parsing could be sufficient to degrade the anomaly detection performance.

Sensitivity to parameters. For parametric methods, the impact of the values of their hyper-parameters on the accuracy results is more rarely studied. However, the sensitivity of a parametric method to its hyper-parameter is of critical importance. On the one hand, if a method shows limited sensitivity to its parameters in terms of accuracy results, then these parameters have little impact on the method's behaviour and their optimization is not necessary. Yet, the parameters are therefore unable to help the modulation of the log parser, which might struggle to adapt to all the reference datasets. On the other hand, if the method is very sensitive to its hyper-parameters, then the optimization of these parameters is a crucial step, that needs to be discussed for the method to be used on new datasets. Despite the great majority of parametric methods, the only study of parameters' influence was performed by Hamooni et al. (2016). The authors however only study the sensitivity of their method to its hyper-parameters in terms of computational efficiency. In our evaluation, we study this sensitivity in terms of impact on the accuracy. We plot heatmaps to visualise it, and study the number of different partitions the method is able to make, by varying its hyper-parameters.

2.3.3 Main conclusions of the related work on structure inference assessment

We summarize the existing methods in Table 2.1 and present, in the following subsections, the desired properties for both the log parsing and the template extraction tasks.

Parametric vs parameter-free methods. While most of the log parsing methods rely on parameters to modulate their behaviour, recent techniques have argued that parameter optimization is an issue, and proposed parameter-free techniques. It is the case of MoLFI, Logram and LFA. However, parameter-free methods tend to provide very heterogeneous and globally inaccurate results (Zhu et al., 2019), which feeds our intuition that parameter setting is key to modulate the behaviour of the log parsers. We therefore consider the parametric character as a desirable feature.

Syntactic assumptions. All of the existing techniques rely on syntactic assumptions, presented in the column Assumption of Table 2.1. Even for the optimization methods that do not implement heuristics behaviours, the distance definition is systematically designed with assumptions on the syntactic similarity of logs. We census the different assumptions in the studied articles in the following paragraphs.

	Log parsing				Template extraction	
Method	Parametric	Assumption	C. efficient	Online	Post-proc.	Indep.
Optimization methods						
LKE	✓	Length, LCS	✘	✘	✓	✘
LogSig	✓	LCS	✘	✘	✓	✘
LogMine	✓	Alignment	✘	✘	✘	✘
MoLFI	✘	Length Distribution	✘	✘	✘	✘
Heuristic-based methods						
AEL	✓	Format	✓	✘	✘	✘
IPLoM	✓	Length Alignemnent	✓	✘	✓	✘
Spell	✓	LCS	✓	✓	✘	✘
SHISO	✓	Length Alignment Format	✓	✓	✘	✘
Drain	✓	Length Position	✓	✓	✓	✘
LenMa	✓	Length Alignment Format	✘	✓	✘	✘
Methods based on frequent pattern mining						
SLCT Distribution	✓	Frequent pattern Distribution Alignment	✓	✘	✘	✘
LogCluster	✓	Frequent pattern Distribution	✓	✘	✘	✘
LFA	✘	Frequent pattern Distribution Alignment	✓	✘	✘	✘
Logram	✘	Frequent pattern Distribution	✓	✘	✘	✘
METI*N*G	✓	Frequent pattern	✓	✓	✓	✓

Table 2.1 – Summary of the state-of-the-art log parsers and their associated template extraction methods, regarding desirable criteria. C. Efficient : computationally efficient. Post-proc : the template extraction is a post-processing step. Indep. : the template extraction method is independent of the assumptions and calculations of the log parsing.

The length and alignment assumptions. The length assumption considers that logs from the same group necessarily have the same number of words. It is a widely used assumption (LKE, MoLFI, IPLoM, SHISO, Drain, LenMa), which presents the convenience of aligning the words in the logs. However, this assumption is unverified for multiple groups in the reference datasets. The alignment assumption consists in considering that the fix parts of the logs are at fix positions in the logs. This constraints is linked to the previous one, even when they are not explicitly applied together. As a result, the alignment assumption alone (in LogMine, SLCT and LFA) offers very limited representation power compared to the length assumption : the fix parts must remain at the same position, hence, only the end of the messages can have variable lengths.

The position assumption. The position assumption, only visible in Drain, consists in assuming a particular position in the logs for the fix words (at the beginning for Drain). The reference datasets contain counter examples of this hypothesis, which is thus too restrictive.

The format assumption. The format assumption is based on the idea that the variable parts of the logs with a common event types have similar formats. AEL directly retrieves the variable parts by defining their formats with regular expressions. SHISO defines some features that can be counted (uppercase characters, numeric characters) and used to create a numerical vectorized representation of words. Finally, LenMa assumes that aligned variable tokens have similar number of characters. These versions of the format assumptions are often contradicted by the reference datasets. In addition, they often strongly rely on domain knowledge, whenever format definitions are involved.

The LCS assumption. A significant amount of the state-of-the-art methods (LKE, LogSig, Spell) implement the LCS (Longest Common Sequence) to evaluate the similarity between logs (or between logs and log representatives). Considering the LCS as a similarity measure consists in assuming that fix parts are likely to be gathered, and form long sequences in the logs. This assumption is unverfiered for some of the groups of the reference datasets, which show strong alternations of fix and variable parts.

Frequent pattern mining and distribution assumptions. The frequent pattern mining assumption generally represents the idea that frequent patterns (word, tokens, pair of words...) are likely to be the fix words of the messages. This assumption seems to be undisputed in the reference dataset, and is therefore the only assumption we consider as recommendable. Yet, most of the state-of-the-art methods (SLCT, LogCluster and LFA) evaluate the frequency of the words in the whole datasets. Hence, implicitly, the groups of logs generated are necessarily the ones that gather the most logs. These methods, that do not recalculate dynamically the frequency of the words, implicitly rely on a distribution assumption : the generated clusters should be as big as possible. Note that this is also the case of MoLFI, which explicitly states this requirement. Nonetheless, the reference datasets contain many examples of small groups of logs, which these methods fail in extracting. We conclude that the frequent pattern mining assumption should be used in a dynamic way, so as to adapt to the presence of rare event types.

Algorithmic efficiency. The algorithmic efficiency, as per evaluated by Zhu et al. (2019), is presented in column Comp. efficient of Table 2.1. Generally speaking, the methods that require an optimization do not scale well on huge datasets, and are therefore not applicable, especially if they require a parameter optimization phase. On the contrary, heuristic-based methods alleviate the computational efficiency by applying efficient heuristics instead of searching for the best solution in an optimization way. Similarly, the frequent pattern mining methods generally perform a one-pass analysis of the frequent pattern, removing the numerous rare ones, and are therefore computationally efficient.

Online functioning. The online functioning of the methods is censused in column Online of Table 2.1. This feature makes the methods eligible for online log mining algorithms. Hence, an online log mining method that requires a structure inference preprocessing can only consider the log parsers that can be applied in an online manner. Only a few modern methods are eligible, namely Spell, SHISO, Drain and LenMa.

Independence of template extraction method. Finally, the columns Post-processing and Independent of Table 2.1 present the template extraction methods which can be used alone, without log parsing. Naturally, the template extraction that are realized during the log parsing are not eligible, which is the case of 10 out of the 14 state-of-the-art methods. The remaining 4 are however not independent of the log parsing, since they rely on previous calculations, or at least, on the same strong syntactic assumptions. Proposing an independent template extraction method is however necessary for extracting the templates of already grouped logs, or to compare the template extraction results alone (independently of the parsing method used).

We conclude that none of the existing structure inference validates all of the desired features. Especially, none of the log parsing method is generic enough to tackle simultaneously the reference datasets, since they contain strong syntactic assumption or are parameter-free. In the next section, we present METI*N*G, a new log parser that relies on a dynamic version of the frequent pattern mining assumption. METI*N*G is parametric, can scale to huge datasets, and can be adapted to an online functioning. We associate to our log parser an independent method that works as a post-processing step.

3 Structure inference with METI*N*G

This chapter details our contribution for the structure inference of log data. Figure 3.1 details the steps that are generally performed in the log parsing articles. The raw logs are first preprocessed by a regex matching step : a list of regular expressions (regex) is provided and the words that match a specific regex are replaced by a unique identifier (letters X, N, P in the second part of the figure). The processed logs are then grouped thanks to the log parser. Each log is associated to a group and therefore to a unique event type. Finally, each generated group is analyzed so as to extract the fix and variables parts. In the figure, the fix words are displayed whereas the variable parts are replaced by a generic token <*>, as in (Zhu et al., 2019).

We first present our log parser in section 3.1. METI*N*G is a parametric method that relies on frequent pattern mining through the search of fix parts among the frequent n-grams. We provide an overview of the method before formally defining its functioning and its hyper-parameters. We also provide an insight on its online extension. We then present our proposition for the template extraction in section 3.2, that is independent of the log parser, and generic enough to treat most of the log groups.

3.1 METI*N*G : a log parser based on frequent n-grams mining

We first present our log parser, published in (**?**). METI*N*G is a parametric method that relies on frequent pattern mining through the search of fix parts among frequent n-grams. We first provide an overview of the method before formally defining its functioning and its hyper-parameters. We also provide an insight on its online extension.

3.1.1 Method overview

Log parsers aim at grouping the logs of a dataset so as to retrieve the original groups of logs, induced by the labels. In Figure 3.1, the log L_1 is associated to the group E_8 and shall be gathered with the log L_4. We summarize the desirable properties for a log parser and explain how METI*N*G implements them.

Adapted syntactic assumption. The frequent pattern mining assumption is considered as a flexible syntactic assumption. It states that the frequent pattern (e.g. words, tokens, pair of words...)

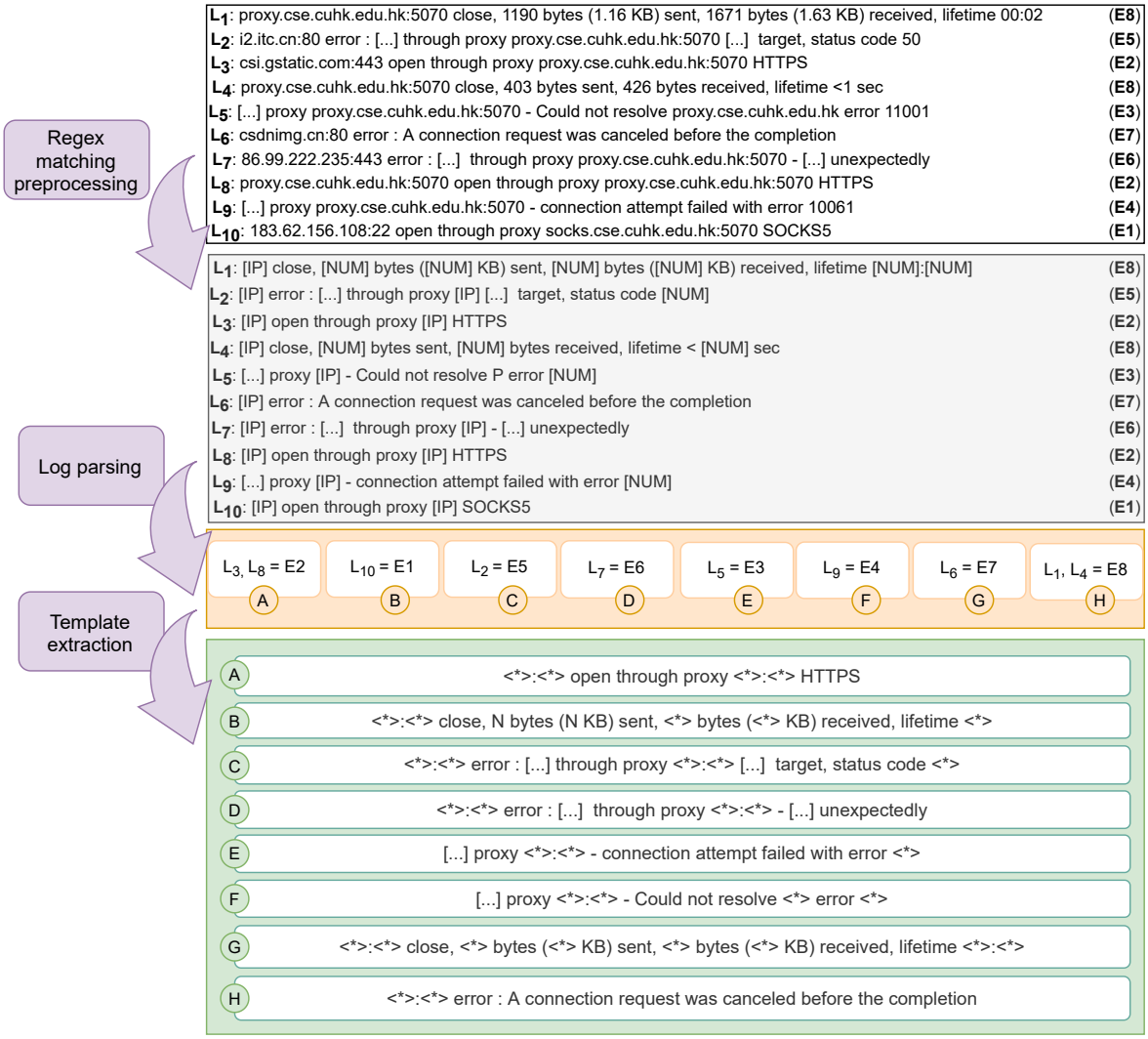


Figure 3.1 – Details of the structure inference steps. The raw logs are preprocessed with a simple regex matching in order to identify some tokens (number, IP addresses...). The processed logs are then parsed, creating groups of logs with the same retrieved event type (A, B, ...). Eventually, each group of logs is treated to extract the fix and variable parts (<*>), creating the template of the group.

are more likely to be fix parts, which is often verified in real datasets. METI*N*G relies on this assumption. Especially, it focuses on frequent n-grams — sequence of n consecutive words. We assume that logs generated by the same command are likely to have common word sequences, and that frequent n-grams might be the fix parts of group templates (Cancedda et al., 2003).

Modulation power enhanced by hyper-parameters. Parametric log parsers offer reliable results, and their modulation power of is enhanced by the usage of hyper-parameters that can change the behaviour of the algorithm. METI*N*G contains two hyper-parameters : n, the length of the n-grams that enables the adaptation of the method to the length of logs, and h, a hyper-parameter that decides how divisive the log parser is — it can help gather logs that are rather heterogeneous or split logs that are similar.

Flexible representation. The state-of-the art methods widely use hierarchical representations (tree construction, hierarchical clustering...). Indeed, these representations fit the nature of textual data (Gaussier et al., 2002). For instance, in Figure 3.1, the log L_3 shares some common structure with the log L_2 — they share the terms `through proxy [IP]` — and even more with the log L_{10} — they share the terms `[IP] open through proxy [IP]`. A hierarchical representation might be able to (i) report on this hierarchical nature (ii) enhance the modulation power of the parsing, by enabling different cuts in the hierarchy. In METI*N*G, we propose to create a *dendrogram*, a tree-like structure, inspired from hierarchical clustering techniques (Johnson, 1967).

Computational efficiency. Due to the huge size of log data, log parsers must be computationally efficient. Clustering techniques that calculate pairwise distances, building a quadratic matrix, are prohibited. Instead, METI*N*G builds its hierarchical representation with a heuristic function that performs a unique pass through the n-grams : at each step, all the words containing a given n-gram a regrouped. This efficient heuristic enables the algorithm to avoid any distance calculation.

Online usage. We mentioned the interest of online log parsers in section 2.3.3. METI*N*G is based on a mining technique (since it studies the overall frequencies of n-grams) and therefore requires an offline learning phase to build the dendrogram. Nonetheless, the rules built during the learning (i.e. the n-grams used to perform the grouping) can directly be used to place new coming logs in the dendrogram in an online fashion.

In the rest of this section, we formalize the context and the problem of log parsing before detailing the implementation of METI*N*G to answer this problem while respecting the aforementioned features.

3.1.2 Formalization

Notations. We define a sequence s as a collection of ordered objects of the same types (not necessarily unique). A collection of unique and unordered objects of the same types is defined as a set E. Table 3.1 presents all the notations associated to these two objects. We further detail some of them in this paragraph.

Sequence $s = (s_i) = s_1 s_2 \dots$	
$s[i] = s_i$	The i^{th} element of s
$s[i, j]$	The subsequence of s, from index$i$ to j (included)
$s(\alpha)[i], \alpha \in \mathcal{A}$	The i^{th} occurrence of symbol α in s
s.pop_head()	Returns and removes the first element of s
Set $E = \{e, \forall e \in E\}$	
$\bigcup_i E_i, \bigcap_i E_i$	Union, intersection of the elements of the sets E_i
sort$(E, \phi), \phi$ a function	Returns a sorted list of the elements of E, according to the order provided by ϕ
$\mathcal{P}(E)$	The set of partitions of E
$\Delta(A, B), A, B \in \mathcal{P}(E)^2$	Returns a similarity measures between two partitions of E
$\|s\|, \|E\|$	length of sequence s, set E
Log $\ell \in L$, composed of words $w_1, w_2 \dots w_{\|\ell\|}$	
(w_i^ℓ)	The sequence of words of ℓ
W_ℓ	The set of words of ℓ
$N(\ell), N(L)$	The set of n-grams of ℓ, of L
$L_g, g \in N(L)$	The subset of words of L that contain the n-gram g
Dendrogram D	
r_D	The root node of D
add_child_nodes$(A, G), A \in D, G \in \mathcal{P}(A)$	Adds children to the node A of D, according to G, a partition of A

Table 3.1 – Summary of the notation conventions used throughout the chapter.

Let $\alpha \in \mathcal{A}$ be a value of the alphabet on which the sequence s is defined. Since the sequence allows repetitions, we identify the occurrences of a value $\alpha \in \mathcal{A}$. These occurrences are ordered, we note $s(\alpha)[i]$ the i^{th} occurrence of the value $\alpha \in \mathcal{A}$ in the sequence s.

For a set E, we define a *partition* $P = \{P_i \subset E\}$ as a set of subsets P_i of E verifying :

— the union of the parts corresponds to E : $\bigcup_i P_i = E$,

— the parts are pairwise disjoints : $\forall i, j \neq i < |P|,\ P_i \cap P_j = \emptyset$.

We note $\mathcal{P}(E)$ the set of all the partitions of E.

We define the function sort, that takes in input a set of elements E and a function ϕ. The function ϕ is defined on E, and attributes a score to any $e \in E$:

$$\begin{array}{rccc} \phi & : & E & \rightarrow & \mathbb{R} \\ & & e & \mapsto & \phi(e) \end{array}$$

The sort function returns a sequence s of unique elements that corresponds to the elements of E ordered according to their scores provided by ϕ. This function is used to sort the values of a set according to a function ϕ that attributes a score to these values.

Context formalizing. Let $\mathscr{L}$ be a dataset of logs. A log $\ell \in \mathscr{L}$ of length $|\ell|$ is composed of the words $w_i^\ell, i < |\ell|$. We note (w_i^ℓ) the sequence of these words and W_ℓ the corresponding set of unique words. For $n \leq |\ell|$, an n-gram of ℓ is a n-long sub-sequence of consecutive words of ℓ : $(w_i^\ell)_{k \leq i < k+n}$, with $k < |\ell| - n$. In Figure 3.2(a), the log L_6 contains, among others, the 2-grams "connection request" and "before the". All the n-grams of ℓ constitute its set of n-grams :

$$N(\ell) = \left\{(w_1..w_{n-1})..(w_{k-n+1}..w_k)\right\}$$

Let $L \subset \mathscr{L}$ be a subset of $\mathscr{L}$, a group of logs. We call $N(L)$ the set of n-grams present in the logs of L : $N(L) = \bigcup_{\ell \in L} N_\ell$. Reciprocally, for a n-gram $g \in N(L)$ we call L_g the subset of logs of L that contain the n-gram g.

Since we opt for a hierarchical representation, we use the *dendrogram* object, a tree-like structure whose nodes are formed by groups of logs. A dendrogram D is composed of nodes, including (i) its *root* r_D, the topmost node, (ii) its *inner nodes*, which are all the nodes that have a child node, (iii) its *leaf nodes*, the final nodes, that do not have child nodes. For a node A, that corresponds to a group of logs, and G a partition of the logs of A, we make use of the function `add_child_nodes`(A, G), that adds all the groups of G as children of A in D by performing the two actions : $\forall G_i \in G$, (i) create a new node corresponding to the logs of G_i (ii) add an edge between A and G_i. This operation is sufficient to build the dendrogram containing the groups of logs. We mention that the leaf nodes correspond to the final groups.

Problem statement. For a dataset of logs $\mathscr{L}$, performing the log parsing task consists in providing a partition of $\mathscr{L}$, G, which is a set of groups of logs. In the example of Figure 3.1, the set of groups $\{A, B, ...F\}$ forms such a partition.

3.1.3 Presentation of METI*N*G

input : $\mathscr{L}$: the logs of a dataset
output: groups : A partition of $\mathscr{L}$

```
1  D ← Dendrogram(root=L);
2  inner_nodes ← { r_D };
3  groups ← ∅;
4  while |inner_nodes|>0 do
5      L ← inner_nodes.pop();
6      G ← division(L);
7      if continue(L, G) then
8          D.add_child_nodes(L, G);
9          for G_i ∈ G do
10             inner_nodes ← inner_nodes ∪ { G_i };
11         end
12     else
13         groups ← groups ∪ { L };
14     end
15 end
16 return groups;
```

METI*N*G generates the parsing partition through the creation of a dendrogram. Starting from a group with all the logs, our method recursively splits the groups into several sub-groups, as shown in Figure 3.2(b). When the dendrogram is built, the final groups are retrieved (represented by the capital letters in Figure 3.2(b) and (c)).

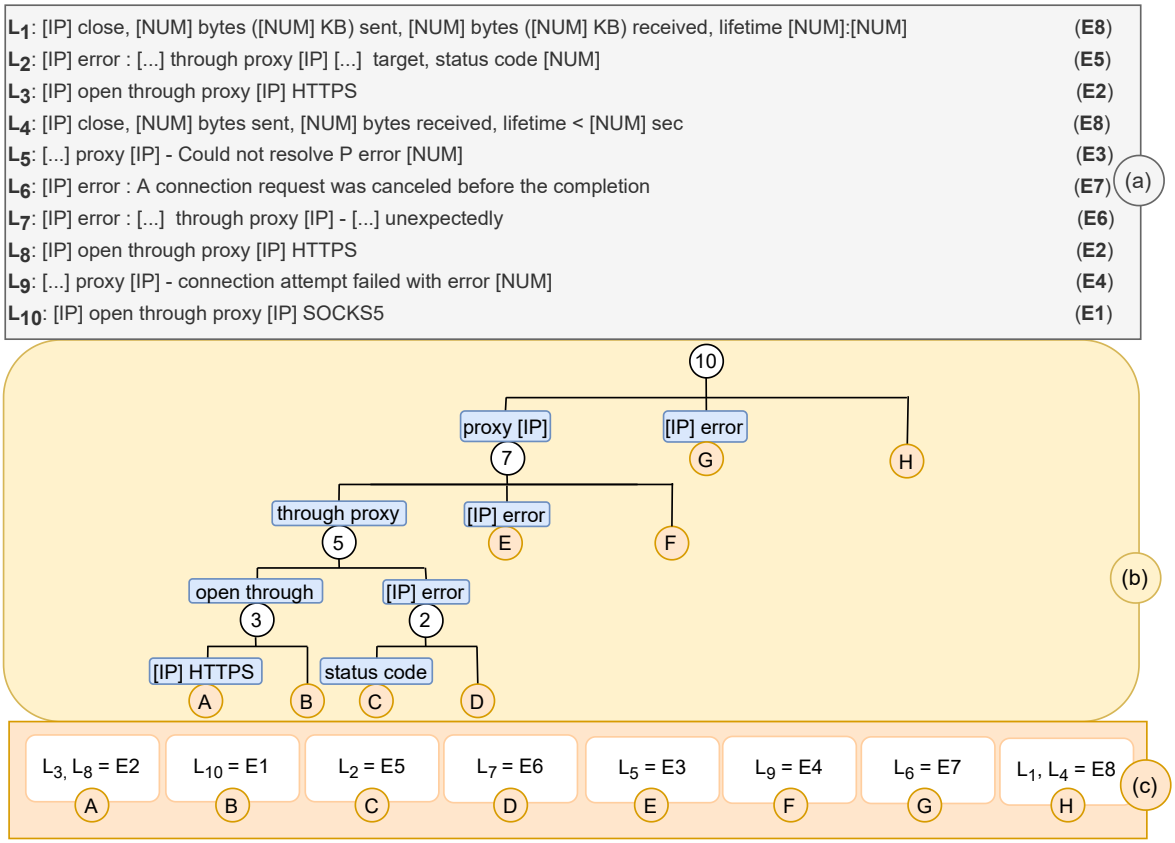


Figure 3.2 – The detailed functioning of METING. From a set of processed logs **(a)**, METING builds a dendrogram to group the logs in a hierarchical manner **(b)**. The final groups in **(c)**, corresponding to the assigned event types can be retrieved as the leaf nodes of the dendrogram (orange nodes **(b)**). In **(b)**, the logs are iteratively split in smaller groups. The blue labels correspond to the n-gram that performs the division. All the inner nodes (in white) contain the number of logs they represent. Note that the ground truth labels (E_1, E_2, ...) are generally not observed. METING does not rely on this information.

The Algorithm 16 presents the global functioning of the METING. Here, the set `inner_nodes` represents the set of nodes that need to be visited. These nodes are visited one after the other (the **while** loop, line **4**), starting from the root r_D (line **2**), that contains all the logs (line **1**). When a node is visited, it is first divided in subgroups by the function `division` (line **6**). G forms a partition of the logs of L. This partition is evaluated with a function `continue` (line **7**) that decides whether the partition is desirable. For example, in Figure 3.2(b), the division that generates the groups A and B is desirable, since it separates logs that have different labels. However, splitting the logs L_1 and L_4 of groups H would not be desirable. The `continue` function aims at managing the interruption of the division process. Depending on the results of this function, the division is either (i) validated; in this case, all the groups of G are added as children of L (line **8**) and will be visited later, (ii) rejected; in this case, the groups of G are dismissed and L is considered as a final group, since it constitute a leaf node (line **13**).

The following paragraphs detail the functions `division` and `continue` that respectively describe how to perform the partition of a group of logs and how to determine whether a partition is desirable or not.

input : L : a set of logs, n : a positive integer, ϕ : a score function for the n-grams, lim : the limit number of n-grams to visit
output: G : a set of subsets of L created by the division process

```
1  n-grams ← N(L);
2  n-grams ← sort(n-grams, φ);
3  G ← ∅;
4  while |n-grams| > |N(L)| − lim AND |L|>0 do
5  |   g ← n-grams.pop_head();
6  |   G ← G ∪ {L_g};          // L_g : the logs of L that contain the n-gram g
7  |   L ← L\L_g;                    // the logs of L without the n-gram g
8  end
9  G ← G ∪ {L};          // the remaining logs are gathered in a unique group
10 return G;
```

An iterative division. METING is composed of successive divisions, each consisting in splitting a group of logs into several smaller groups of logs. Our proposition consists in creating the partition based on the presence of n-grams, where n is a hyper-parameter. The idea is to choose n-grams that are likely to be fix parts, and regroup all the logs containing these n-grams. The Algorithm **??** details how METING performs a division step and renders a partition of a group of logs L.

We detail the functioning of our method on a complete example, corresponding to the logs of the Figure 3.2. In this example, we fix $n = 2$. The root node, r_D (line **2** of Algorithm 16) corresponds to the first node of the dendrogram, and contains all the (10) logs. When it is visited, its logs are first divided into several groups by the `division` method (called in Algorithm 16, line **6**). The Algorithm **??** describes how a group L — here, all the logs — is divided into a partition. First, all the n-grams of all the logs of L are collected (line **1**), with $N(L) = \{$`[IP] close`, `close [NUM]`, `[NUM] bytes`, ..., `[IP] error`, ..., `[IP] open`, `open through` ...$\}$ The n-grams are then sorted according to ϕ (line **2**), a function that attributes a score to each n-gram and which is described later in this section. The n-grams are visited in the order provided by the scoring function (**while** loop, line **4**). When a n-gram is visited, all the logs that contain this n-gram are parsed together (line **6**), while the remaining logs will be treated later, with another n-gram (line **7**). Without describing the scoring function, Table 3.2 censuses the n-grams according to their order, and shows the associated logs for each of them.

In the example of Figure 3.2, the chosen n-grams appear in the blue label above the nodes. In the first division, the n-gram with the best score is `proxy [IP]`, therefore, all the logs containing

n-gram	score	Associated logs
proxy [IP]	7	$L_2, L_3, L_5, L_7, L_8, L_9, L_{10}$
through proxy	5	$L_2, L_3, L_7, L_8, L_{10}$
[IP] error	4	L_2, L_5, L_6, L_7
open through	3	L_3, L_8, L_{10}
[IP] HTTPS	2	L_3, L_8

Table 3.2 – The scores and associated logs of some relevant n-grams for the group of logs $L = \mathscr{L}$ (all the logs of the example of Figure 3.2

this n-gram (i.e. $L_2, L_3, L_5, L_7, L_8, L_9, L_{10}$) are regrouped in the leftmost node. The second n-gram in the sorted list is `through proxy`, that matches the logs $L_2, L_3, L_7, L_8, L_{10}$. However, all of these logs are already regrouped in the leftmost node. This n-gram is therefore ignored. The next n-gram on the list is `[IP] error`, that matches the logs L_2, L_5, L_6, L_7. Yet, three of these logs (i.e. L_2, L_5, L_7) are already in the leftmost node. Hence, only L_6 is set to the middle node. Finally, the remaining logs L_1 and L_4 are parsed to the rightmost node. Indeed, the **while** condition (line **4**) controls the visit of the n-grams, and specifies that the process should stop either (i) when all the logs are grouped or (ii) after lim iterations, where lim is a parameter to define. This limit enables to avoid visiting n-grams with low scores, that would generate unsatisfactory grouping. Instead, the remaining logs are grouped in a final node, without n-gram label. In the example we presented, $lim = 2$.

After this first iteration is completed, a partition is formed and evaluated in Algorithm 16 line **7**, with the function `continue`. If the partition is validated, the groups are created and added to the dendrogram. Otherwise, the partition is rejected. The groups are not added to the dendrogram, and the initial group of logs L is labelled as a final group (orange nodes with capital letters in Figure 3.2(b)). We mention that this iteration process is computationally efficient, since it only requires to check the presence of a n-gram in a log.

The function ϕ, that associates a score to the n-grams, is of fundamental importance : if a n-gram g has a high score, it will lead to the gathering of all the logs containing it. Otherwise, when it is visited, some of the logs that contain g might already be grouped, due to a former n-gram (it might also not be visited at all, due to the limit lim). The idea is to select n-grams that are likely to be the fix parts of the templates of the groups. The following paragraph details our proposition for the implementation of this function.

N-gram sorting function. METI*N*G relies on the frequent pattern mining assumption. Hence, at each division step, METI*N*G performs the partition according to frequent n-gram mining. We therefore want to assign a high score to the most frequent n-grams.

For each n-gram $g \in N(L)$, its frequency in L is the number of logs of L in which it appears. We define the frequency function by:

$$\begin{array}{rccl} \texttt{freq}_L & : & N(L) & \to & \mathbb{N}^* \\ & & g & \mapsto & |\{\ell \in L,\ g \in N(\ell)\}| \end{array}$$

In Table 3.2, that represents the group $L = \mathscr{L}$, containing all the logs, the 2-gram `proxy [IP]` appears in 7 logs of L, so $\texttt{freq}_L(\textit{proxy [IP]}) = 7$. Since this 2-gram is frequent, it is likely to be a fix part. Hence, we split the logs of L based on the presence of the most frequent n-grams. Note that the score function is re-calculated at each division step (since it is part of the `division` function, in Algorithm **??**). Hence, the scores of Table 3.2 are different at the next iteration : in the leftmost node, that contains all the logs matching `proxy [IP]`, the score of the n-gram `through proxy` is still 5, yet the score of `[IP] error` falls down to 3.

Stopping criteria. We finally define a condition so as to stop the splitting iteration and render the final groups of logs. This condition is checked in Algorithm 16, line **7**, and decides whether to validate or reject a proposed partition. The condition is based on the improvement of a homogeneity

score, $\texttt{homogeneity}(L)$, calculated for each group L. The `continue` function can be expressed as :

$$\texttt{continue}(G,L) \iff \texttt{homogeneity}(L) > \mu \cdot \frac{1}{|G|} \cdot \sum_{G_i \in G} \texttt{homogeneity}(G_i) \tag{3.1}$$

In the condition 3.1, μ is a coefficient of the stability of homogeneity during divisions, and G is the partition created by the division of L with the function `division`. In other words, we compare the former homogeneity score to the average of the homogeneity scores of the new groups : the ratio $\frac{\texttt{homogeneity}(L)}{\frac{1}{|G|}\cdot\sum_{G_i \in G}\texttt{homogeneity}(G_i)}$ measures the steadiness of the homogeneity during division. This ratio is 1 if the homogeneity is stable, and is between 0 and 1 if the homogeneity is improved. The division process is continued only if this ratio is smaller than its maximum value, μ, meaning that the homogeneity has sufficiently improved. μ can be seen as the required steadiness of homogeneity to generate the final groups. We now define both the homogeneity score $\texttt{homogeneity}(L)$ and μ, the coefficient of homogeneity improvement.

We define the homogeneity score as the arithmetical mean of two indicators of homogeneity :

$$\texttt{homogeneity}(L) = \frac{\texttt{fix\_word}(L) + \texttt{length\_stability}(L)}{2}$$

First, $\texttt{fix\_word}(L)$ measures the homogeneity of L as the proportion of words appearing in all the logs of L:

$$\texttt{fix\_word}(L) = \frac{\left|\left\{w_j \mid \forall \ell \in L, w_j \in (w_i^\ell)\right\}\right|}{\left|\left\{w_j \in (w_i^\ell) \mid \ell \in L\right\}\right|}$$

The numerator calculates the number of words present in all the logs $\ell \in L$ and the denominator counts the total number words in the logs. This indicator associates a high homogeneity score to groups containing numerous common words.

Secondly, the `length_stability` of a group measures the variations among the lengths of logs, $X_L = (|\ell|,\ \ell \in L)$. We denote the p-percentile of X as $Q_p(X)$. $Q_p(X)$ is the value so that p% of the values of X are inferior to $Q_p(X)$. We define:

$$\texttt{length\_stability}(L) = 1 - \frac{Q_{95}(X_L) - Q_5(X_L)}{Q_{95}(X_L)}$$

The ratio in this formula is equivalent to a relative interquartile range of the lengths distribution. This indicator attributes a high homogeneity score to the groups that have a small variability in their log lengths.

Finally, μ is the coefficient of homogeneity steadiness, introduced in 3.1. μ imposes an improvement of the homogeneity during a division step : a small value of μ imposes an important gap between the homogeneity scores of L and its successors. We define μ as the arithmetical mean of three indicators:

$$\mu = \frac{1}{3}\left(\frac{1}{h} + \frac{1}{\texttt{nb\_selection}(L)} + \frac{1}{|G|} \cdot \sum_{G_i \in G} \phi\big(g(G_i)\big)\right) \tag{3.2}$$

$\texttt{nb\_selection}(L)$ is the number of times, during its divisive creation process, that L was generated thanks to a n-gram, instead of being created as the group of remaining logs. In Figure 3.2(b), `A` is selected 4 times while `F` or `G` are only selected once. Groups resulting from a lot of selections are far more homogeneous than unselected groups, since, their logs share more common n-grams, by

construction. μ is linked to the inverse of $\texttt{nb\_selection}(L)$, which means that the requirement of improvement of homogeneity is higher for groups which have already been selected. This mechanism promotes the split of unselected groups, rather than already-selected ones. To determinate the value of $\texttt{nb\_selection}(L)$ for a group L, we propose a recursive definition : we instantiate $\texttt{nb\_selection}(\mathscr{L}) = 1$, to avoid division by zero. Thereafter, we define the following recursive relations for the subgroups of G, created from the partition of L :

$$\forall G_i \in G, \texttt{nb\_selection}(G_i) = \begin{cases} \texttt{nb\_selection}(L) + 1 \text{ if } G_i \text{ was generated by an } n\text{-gram;} \\ \texttt{nb\_selection}(L) \text{ otherwise.} \end{cases}$$

The rate μ also depends on the average of the $\phi\big(g(G_i)\big)$, where $\forall G_i \in G$, $g(G_i)$ is the n-gram used for the creation of the subgroup G_i, and ϕ is the function that attributes a score to the n-grams. We are here evaluating the score of the selected n-grams, that generated the groups of G. The assumption is that if the most frequent n-grams in a group of logs have low frequencies, then it is likely that the final group is reached. For instance, if these n-grams are (or contain) numbers, they may have low frequencies in the original group, and splitting L according to such n-grams is not desirable.

Finally, h, the *homogeneity tolerance* is an hyper-parameter, described in the following section.

3.1.4 Hyper-parameter description

In the presentation of our method, we referred to three parameters :

— n is the length of the n-grams. In the specific case, for a log ℓ, where $n > |\ell|$, ℓ does not contain any n-gram. Hence, the "short" logs of a dataset are parsed apart from the others, thanks to 1-grams. n should be carefully chosen so as to avoid splitting the logs of a common group between the two categories. On the contrary, this n-length threshold can promote a difficult separation between two similar groups of logs.

— h, the homogeneity tolerance is a rate, with $0 < h \leq 1$, that controls how divisive the algorithm is. High value of h implies small values of μ, leading to strict splitting conditions, preventing the algorithm to be divisive. h enables the tuning of the algorithm divisibility.

— lim is the limit number of n-grams visited at each division step. It enables to avoid visiting n-grams with low scores, that would lead to unsatisfactory divisions. A special case worth describing is the value $lim = 1$, where only the most frequent n-gram is visited and leads to a grouping, while all the remaining logs are grouped in rightmost node. In this case, we obtain a binary tree, and the n-grams scores are re-calculated after each single grouping.

In the context of log parsing, we fix $lim = 1$, meaning that we choose to systematically build a binary tree. This configuration presents the advantage to run a unique n-gram at each division step, which means that the order of the n-grams, induced by their scores ϕ, is updated as often as possible. This configuration guarantees the most precise scoring of the n-grams. Our method actually censuses two hyper-parameters only, n and h. Section 4.2.3 provides further guidance on parameter setting.

3.1.5 Online extension

Thanks to its tree-like representation, our method is eligible for online parsing. Indeed, once the dendrogram is built, a new log can be automatically parsed through the n-grams. At each level of the dendrogram, if the new log contains the splitting leftmost n-gram, then it is sent to the leftmost node, otherwise, if it contains the second n-gram, it is grouped with the second node... and finally, if it does not contain any n-gram of the labels : (i) if the level contains a node without label, like the first level of Figure 3.2 with group `H`, the log is assigned to this node, (ii) else, if all the nodes have a label, like in the third level of the Figure (*open through* and *[IP] error*), a new child node is created. This process is applied until the run reaches the leaf nodes of the dendrogram. Thereafter, the splitting condition 3.1 is questioned again : since the group where the log was assigned to has changed, it might be eligible for splitting. If so, the group is split and the dendrogram updated.

3.2 Template extraction

We detail in this section a complete method to extract the template of a group of logs. This process does not rely on any assumption related to METI*N*G, hence, it can be applicable as post-processing step to any log parser, or to logs that are already parsed. Our method does not only generate a template with wildcards (`<*>`) to identify the variable parts, but it also can identify the types of the variables.

3.2.1 Problem presentation

In this subsection, we first present a toy example of data that we use throughout the section. We specify the desired output for the template extraction task and we formally describe the problem of extracting the template.

Presentation of a toy example. As a minimal example, we choose a group of 3 fictive logs that census the same information, and could therefore plausibly be parsed together, despite their heterogeneity. We call L this group of logs, and the logs are noted ℓ_1, ℓ_2, ℓ_3 :

ℓ_1 =*Message "Data has been exported" sent. Success. Message "ok" received.*
ℓ_2 =*Message "Transferring data" sent. Success. Message "Permission to transfer" received.*
ℓ_3 =*Message "Message sent" sent. Success. Message "Message has been transferred" received.*

Objectives and desired outputs. We want to extract the template of this group of logs based on the data. The expected results is to be able, for each log, to distinguish fix and variable parts of the log. Not only do we want to identify fix and variable words for each log individually, but we also want to be able to represent the fix and variable parts in columns, like IPLoM or LKE do. A desired output (Table 3.3) could be to represent the group of logs L with both the template : `Message "[VARIABLE - TEXT]" sent. Success. Message "[VARIABLE - TEXT]" received.`, and the corresponding table, indicating, for each log, the values of the variables.

Template	VARIABLE - TEXT (1)	VARIABLE - TEXT (2)
ℓ_1	Data has been exported	ok
ℓ_2	Transferring data	Permission to transfer
ℓ_3	Message sent	Message has been transferred

Table 3.3 – Example of a desirable output for the template extraction of the toy example.

In this example, the fix and variable parts have been aligned so as to be able to compare the correspondence between the logs. Instead of simply identifying the variable parts with a token <*>, we can analyze the types of variable data being rendered (integers, strings, dates...). This is particularly important if we want to use these variable values to monitor the system execution (Du et al., 2017). Some statistics can be inferred when the variable is a numerical value (mean, standard deviation...), and when it is a categorical value, the distribution of the values can be statistically analyzed. Generally, this distribution can be a valuable information for the end-users to quickly detect unexpected values (an outlier numerical value, a rare or new categorical value...).

Finally, the template extraction problem consists in (i) identifying in each log the fix parts (ii) aligning the fix and variable parts between all the logs (iii) finding the types of the variable parts and (iv) optionally, providing some distribution statistics of these variables.

Problem statement. The identification of the fix parts of the logs consists in identifying the words that are common to all the logs, and providing a generic template. This template is an alternation of fix and variable blocks that must fit all the logs. We define $f(L)$ as the sequence of fix words in L, containing the words that are present in all of the logs of L, respectfully of the order of $f(L)$. To extract the template, we need to group the consecutive words into blocks, so that (i) no variable part is present between the words of a block (ii) optional variable parts can be present between two consecutive blocks. With these rules, the template τ can be described only with a sequence of blocks : $\tau = b_1 b_2 \ldots$ where $\forall j < |\tau|, b_j = f_{j_1} f_{j_2} \ldots$, with the conditions :

— τ is a partition of $f(L)$, meaning that each element of $f(L)$ appears in exactly one block of τ;

— $\forall j < |\tau|, \forall \ell \in L, b_j \subset \ell$, meaning that each block of τ must be present in each log (without interruption);

— $\forall \ell \in L$, the blocks must appear in the same order in the log. In other word, if a block b_j is found in a log ℓ, the following blocks of τ must be found later in the log.

The following formalizing resumes these requirements :

$$\forall \ell \in L,\ \forall b_i \in \tau,\ \exists (p,q) < |\ell| \textbf{ s.t. } b_j = \ell[p,q] \textbf{ AND } b_{j+1} \subset \ell[q+1, |\ell|] \tag{3.3}$$

Extracting the template of L consists in identifying the blocks of τ. The variable parts are then detected as the words between each consecutive blocks. The logs can then be parsed so as to match the template, and can be aligned in columns.

3.2.2 Identifying the fix parts

In this subsection, we illustrate the challenges of identifying the fix blocks from the fix words of a group of logs, and propose an automated solution composed of two steps : (i) identifying the fix words, (ii) selecting the occurrences of the fix words.

1	2	3	4	5	6	7	8	9	10	11
Message	Data	has	been	exported	sent	Success	Message	ok	received	
Message	Transferring	data	sent	Success	Message	Permission	to	transfer	received	
Message	Message	sent	sent	Success	Message	Message	has	been	transferred	received

Table 3.4 – Positions of words in the 3 logs. IPLoM aligns the words according to their positions. In this case, IPLoM does not identify any fix part.

	Message	data	Success	sent	transfer	received	ok
ℓ_1	2	1	1	1	0	1	1
ℓ_1	2	1	1	1	1	1	0
ℓ_1	4	0	1	2	0	1	0
min	2	0	1	1	0	1	0

Table 3.5 – An extract (not all the words are displayed) of the document term matrix obtained in the toy example.

Challenges. The fix words $f(L)$ cannot be extracted as straightforwardly as in IPLoM. IPLoM performs an alignment of words and simply identifies the columns that have a unique values as fix words — as presented in Table 3.4. In our example, the logs do not have the same length, hence, they cannot be directly aligned in columns.

Once the fix words are identified (in order), finding a bijection between the fix words of $f(L)$ and the words of each log is challenging whenever the log has more occurrences of the fix words. For instance, ℓ_3 has 2 additional occurrences of the word *Message*, and 1 additional occurrence of the word *sent*, compared to the two other logs. It is therefore challenging to identify which occurrence(s) of the words of ℓ_3 correspond to the fix parts. Answering this question is a requirement for the obtaining of the fix blocks. For instance, for log ℓ_3, if the second occurrence of *Message* is chosen as a fix part instead of the first occurrence, the general template would become `<*> Message <*> sent. Success. Message <*> received`. To obtain the desired template, we need to select, in ℓ_3, the first and third occurrences of *Message*, and the second occurrence of *sent* as fix parts.

Identification of fix words. To identify the fix words $f(L)$, we use a Document-Term matrix (introduced in (Kosala and Blockeel, 2000)), that provides a mapping of the words and the logs. We aim at counting the number of occurrences of each word in each log. Table 3.5 presents an extract (not all the words are displayed) of the Document-Term matrix in the example.

The last line *min* shows the minimum number of occurrences of the word over the logs. A value $n \geq 1$ means that the word is present at least once in all of the logs. It is therefore a fix word. If the value is $n > 1$, then n occurrences of the word might be considered as fix words. In this case, a common order can be found within the logs, giving the result of :

$$f(L) = \textit{Message}, \textit{sent}, \textit{Success}, \textit{Message}, \textit{received}$$

The logs ℓ_1 and ℓ_2 have the minimum number of occurrences of the fix words, in the right order. For these logs, the fix and variable parts can therefore be directly extracted. However, this extraction is more challenging for the log ℓ_3, which has 4 occurrences of the fix part *Message* instead of 2, and 2 occurrences of the fix word *sent* instead of 1. Respecting the order is necessary to align the fix parts among the logs. This imposes that the first fix word *Message*, $f(\textit{Message})[1]$

corresponds to either the first or second occurrences of *Message* in the log ℓ_3, $\ell_3(Message)[1]$ and $\ell_3(Message)[2]$, while the second fix word *Message* is necessarily among the third and fourth occurrences of *Message* in ℓ_3, $\ell_3(Message)[3]$ and $\ell_3(Message)[4]$.

Selection of occurrences of the fix words. For the three fix words $f(Message)[1]$, $f(Message)[2]$ and $f(sent)[1]$, we have to decide between two candidate occurrences in ℓ_3. We call the associated occurrences in ℓ_3 *ambiguous occurrences*. To obtain the desired template, we must select the occurrences $\ell_3(Message)[1]$, $\ell_3(sent)[2]$ and $\ell_3(Message)[3]$. Our proposition consists in studying the gaps (number of words) between consecutive fix words, and choosing the occurrences that promote steady gaps. This paragraph presents different simpler approaches and their limitations, before showing the behaviour of our proposition on the toy example.

Study of the mean positions. A first simple intuition to select the occurrences is to compare the ambiguous occurrences to the nominal positions of the fix words (in the logs without ambiguous occurrences, namely ℓ_1 and ℓ_2). Table 3.6 contains the mean positions of the fix words, in the nominal cases.

	$f(Message)[1]$	$f(sent)[1]$	$f(Success)[1]$	$f(Message)[2]$	$f(received)[1]$
ℓ_1	1	6	7	8	10
ℓ_2	1	4	5	6	10
Average	1	5	6	7	10

Table 3.6 – The positions of the fix words in the reference words, i.e. the words that contain the minimum number of occurrences of each fix word.

By comparing the position of the observed occurrences of the fix words in ℓ_3 to the average positions, we find that $\ell_3(Message)[1]$ best matches $f(Message)[1]$ and $\ell_3(sent)[2]$ best matches $f(sent)[1]$, which are both the right decisions. Yet, $\ell_3(Message)[4]$ best matches $f(Message)[2]$, whereas the actual occurrence is $\ell_3(Message)[3]$. The promoted solution is the triplet ($\ell_3(Message)[1]$, $\ell_3(sent)[2]$, $\ell_3(Message)[4]$). Actually, we observed that the position of the fix words are not fixed, because the variable parts have variable lengths.

Study of the mean spaces. Another proposition consists in studying the spaces between the fix words (number of words that separate the fix words). These spaces are likely to be similar from one log to another, which can help to select the occurrences of fix words in ℓ_3. Table 3.7 contains the average space separating the fix words in the nominal logs, obtained by averaging the differences of positions from Table 3.6. This table is the reference for nominal spaces between fix words. We compare it to the observed spaces in ℓ_3, under the different assumptions of selected occurrences (e.g., $f(Message)[1] = \ell_3(Message)[1]$). Table 3.8 presents the difference between the expected spaces (from Table 3.7) and the observed spaces under the different assumption. The smallest the difference is, the most likely the assumption is. Since the actual positions of the fix words $f(Message)[1]$, $f(sent)[1]$ and $f(Message)[2]$ are not determined for ℓ_3, we only study the spaces between the fix words $f(Success)[1]$ and $f(received)[1]$, and the ambiguous occurrences. We see that this method does not promote any of the *Message* occurrences. It however rectifies the previous error and correctly chooses the second occurrence of *sent*.

	$f(Message)[1]$	$f(sent)[1]$	$f(Success)[1]$	$f(Message)[2]$	$f(received)[1]$
$f(Message)[1]$	0	4	5	6	9
$f(sent)[1]$	-4	0	1	2	5
$f(Success)[1]$	-5	-1	0	1	4
$f(Message)[2]$	-6	-2	-1	0	3
$f(received)[1]$	-9	-5	-4	-3	0

Table 3.7 – The average space separating the positions of the occurrences of the fix words, in the nominal logs (ℓ_1 and ℓ_2).

	$f(Success)[1]$	$f(received)[1]$	Average difference
$f(Message)[1]$	5	9	0
$\ell_3(Message)[1]$	4	10	**1**
$\ell_3(Message)[2]$	3	9	**1**
$f(sent)[1]$	1	5	0
$\ell_3(sent)[1]$	2	8	2
$\ell_3(sent)[2]$	1	7	**1**
$f(Message)[2]$	-1	3	0
$\ell_3(Message)[3]$	-1	5	**1**
$\ell_3(Message)[4]$	-2	4	**1**

Table 3.8 – The observed space separating the positions of the occurrences of the fix words, in the log of interest ℓ_3. We only select, as reference (columns), the fix words that are not ambiguous. The column Average difference is the average of the two previous columns. We compare the actual position space to the formerly calculated ones (e.g., $f(Message)[1]$, calculated in 3.7).

This method discards a wealth of information concerning the spaces, since it discards all the relations between ambiguous occurrences. Instead, we could study all the possible triplets. To avoid redundancies, we only census the relations between consecutive fix words in Table 3.9.

In this study, two triplets are promoted : the one that was proposed in the former paragraph,

Triplets (occurrence in ℓ_3)	*Mess.*[1] - *sent*[1]	*sent*[1]- *success*	*success*- *Mess.*[2]	*Mess.*[2]- *received*	Average difference
Reference	4	1	1	3	0
Mess.[1] *sent*[1] *Mess.*[3]	2	2	1	5	1.25
Mess.[1] *sent*[1] *Mess.*[4]	2	2	2	4	1.25
Mess.[1] *sent*[2] *Mess.*[3]	3	1	1	5	**0.75**
Mess.[1] *sent*[2] *Mess.*[4]	3	1	2	4	**0.75**
Mess.[2] *sent*[1] *Mess.*[3]	1	2	1	5	1
Mess.[2] *sent*[1] *Mess.*[4]	1	2	2	4	1
Mess.[2] *sent*[2] *Mess.*[3]	2	1	1	5	1
Mess.[2] *sent*[2] *Mess.*[4]	2	1	2	4	1

Table 3.9 – Alternative study of the position spaces, by comparing only consecutive occurrences of fix words. We study, for each possible triplets of ambiguous fix words, the actual values of the space positions between the consecutive words, and compare it to the reference, calculated in Table 3.7. Each column corresponds to the space between two consecutive fix words, and the column Average difference is the average, for each previous column, of the difference between the actual space and the reference space.

Occurrence selection for $f(Message)[1]$	$f(Success)[1]$	$f(received)[1]$	Average difference	Weighted average difference
Mean space	5	9		
Interquartile range	2	4		
Weights	0.135	0.018		
$\ell_3(Message)[1]$	4	10	**1**	**1**
$\ell_3(Message)[2]$	3	9	**1**	1.762

Table 3.10 – The average difference weighted according to the interquartile ranges, to resolve which occurrence of $f(Message)[1]$ to choose (same can be applied to the two other ambiguous occurrences). We calculate the reference average space (from 3.8) and the interquartile range of the same space. The interquaritle ranges are used to calculate weights for the new weighted average difference (last column) between the reference space and the actual spaces (last two lines).

namely ($\ell_3(Message)[1]$, $\ell_3(sent)[2]$, $\ell_3(Message)[4]$), which is incorrect, and ($\ell_3(Message)[1]$, $\ell_3(sent)[2]$, $\ell_3(Message)[3]$) the correct triplet. This method is therefore insufficient. Moreover, the number of possible combinations of fix words rapidly reaches high values when the number of ambiguous occurrences increases, and the number of concerned fix words increases (exponential size). This solution is therefore abandoned, and the former one is rather investigated for improvement.

Study of the dispersion of the spaces. We observe that some of the aforementioned spaces are more reliable than others : while the space between $f(Success)[1]$ and $f(Message)[2]$ is always 1, the space between $f(Message)[1]$ and $f(sent)[1]$ varies according to the length of the first variable block (the content of the message being sent). Generally speaking, the space between fix words becomes more and more unreliable when the number of variable blocks between them augments. We want to measure this reliability as the dispersion of the series of spaces. We propose to study the interquartile range of each space and use the ranges to weight the previous average calculations. Table 3.11 presents the obtained results for the study of $f(Message)[1]$.

In this table, we calculate the weights of the space relations thanks to the interquartile ranges. For two fix words f[i] and f[j], we note $ir(f[i], f[j])$ the interquartile of the ranges between these two fix words, and $m(f[i], f[j])$ the mean space. We attribute the following score to the mean $m(f[i], f[j])$:

$$w_{i,j} = e^{-ir(f[i],f[j])}$$

In the former table, the weight of the relation $f(Message)[1]$ - $f(Success)[1]$ is ten times more than the weight of the relation $f(Message)[1]$ - $f(received)[1]$, meaning that the first space is much more reliable than the second. The final scores, for the two potential occurrences of $f(Message)[1]$ in ℓ_3 are calculated by the formula :

$$score_i = \frac{1}{\sum_j w_{i,j}} \cdot \sum_{j<f, j\neq i} w_{i,j} \cdot m(f[i], f[j])$$

This results to the selection of $\ell_3(Message)[1]$. Table 3.11 censuses the scores obtained for the ambiguous occurrences, which leads to the selection of the right triplet of occurrences, namely

$f(Message)[1]$		$f(sent)[1]$		$f(Message)[2]$	
$\ell_3(Message)[1]$	$\ell_3(Message)[2]$	$\ell_3(sent)[1]$	$\ell_3(sent)[2]$	$\ell_3(Message)[3]$	$\ell_3(Message)[4]$
1	1.762	1.238	**0.238**	**0.238**	1

Table 3.11 – The weighted average difference obtained for the ambiguous occurrences of the fix words in ℓ_3, which enhance the selection of the right occurrences.

$\big(\ell_3(Message)[1],\ \ell_3(sent)[2],\ \ell_3(Message)[3]\big)$. This selection provides the desired alignment of columns.

Template generation. Once the fix parts are identified and localized in each log, the blocks of fix words can be immediately retrieved, and the template can be generated by inserting wildcards between each block. We obtain the desired template : `Message “<*>” sent. Success. Message “<*>” received.`

3.2.3 Post-processing the variable parts

After the identification of the variable parts as wildcards, these parts can be post-processed so as to identify the types of data. Thanks to a set of regular expression, that match the domain context, the variable parts can be parsed and identified according to the correspondence to these elements. For instance, the variable parts can be identified as numbers, integers, dates, IP addresses...

If all the values of a variable part over the logs are composed of only the regular expression to be matched, this process is straightforward, since the whole string that represents the variables can be checked for matching. It becomes more complicated when the pattern actually corresponds to only a part of the variable string. For instance, for the following logs, the identification of the types of the variable parts is challenging :

Action Send message 21 to 10.25.69.71 done
Action Read text 09 on 102.27.79.01 done

In this case, we perform the search for matching term-by-term. This can lead, for the central variable part, to the identification of the types : `[VARIABLE - TEXT] [VARIABLE - INTEGER] [VARIABLE - TEXT] [VARIABLE - IPV4]`

In conclusion, we presented a very complete method to process the groups of logs so as to extract the templates. We detailed a challenging example and show how our proposition can deal with such situations, contrary to the log parsers which rely on the unique length assumption.

4 Validation

In this chapter, we assess METI*N*G on datasets from different sources and compare it to the state-of-the-art references. We study the behaviour of the methods on challenging datasets and evaluate their modulation power. We analyze in detail the behaviour of the methods in specific data configurations. We also provide an insight on the parameter setting and its use for online log parsing. We exhibit an example of template extraction with the output obtained for a group of logs. Finally, we evaluate the computational efficiency of our method and its scalability to huge datasets, and online functioning.

4.1 Parsing evaluation framework

For our evaluation, we rely on the complete framework of LogPAI (Zhu et al., 2016), that provides the evaluation of 13 methods on 16 datasets.

4.1.1 Evaluation measures

To measure the accuracy of a log parsing methods, external measures are commonly used to calculate pairwise relationship matches, i.e. evaluating the success of grouping pairs of logs together. The traditional precision, recall and F-measures indicators are often employed (Makanju et al., 2012; Messaoudi et al., 2018). However, as explained by Zhu et al. (2019), these indicators tend to provide very smooth positive results. Indeed, these measures only take interest in pairwise relationships, and do not consider the success of correctly retrieving a whole group of logs. This correct retrieving is however key to insure a correct template extraction. The authors define the more rigorous measure of parsing accuracy as the rate of logs that are exactly correctly parsed, as described in section 2.1. A log is correctly parsed if its parsing group contains all the logs with the same label, and does not contain any logs with different labels. For instance, if a dataset contains 6 logs of event types $[E1, E1, E2, E2, E3, E3]$ that are parsed to the groups $[G1, G1, G2, G2, G2, G3]$, the accuracy score is $2/6$ since only the two first logs (with label $E1$ and attributed to $G1$) are well-parsed. We use the parsing accuracy to measure the performance of METI*N*G.

4.1.2 Experimental protocol

To evaluate the methods, we perform the log parsing of the 16 datasets of LogPAI (Zhu et al., 2016). Our experiments are conducted on a PC with Windows 10 OS, with an Intel(R) Core(TM) i7-

6820HQ, 2.70GHz CPU and 16Go RAM. We first preprocess the logs (as proposed in LogPAI), apply the parsing method, retrieve the corresponding event types, and compare them to the manually-obtained labels thanks to the accuracy score calculation. We detail in the following paragraphs some important points of these steps.

Data presentation. The proposed log datasets were made available with the LogPAI tool. The framework gathers 16 datasets composed of 2000 log lines each, and manually labelled. Details of the source and distribution of the datasets can be found in Section 1.2.2. We also study logs from the OBIS. We only select the applications that contain an event type. The log parsers can obviously be applied to label-free data, yet, no numerical results would be available for comparison. Moreover, it would be impossible to perform hyper-parameter optimization.

Preprocessing of logs. LogPAI implementation provides, along with each dataset, a set of regular expressions to apply, regardless of the parsing algorithm. The OBIS dataset does not benefit from such a list. We therefore propose two experiments : (i) we first perform the log parsing with a very basic preprocessing, that simply identify the numeric values, and which does not require any specific domain knowledge, (ii) we conduct a deeper analysis on the understudied logs in order to find relevant regular expressions to be provided to preprocessing, as a second experiment. We aim at observing (i) which methods are able to parse the logs without domain knowledge, and (ii) the impact of this domain knowledge addition to the results.

By analyzing the implementation of the log parsers in LogPAI, we observed that the methods dealt differently with these regular expressions : some methods remove the parts of logs that match the regular expressions, while others replace the matching parts with a generic term, describing the regular expression (e.g. "INTEGER", "IPv4"...). For our method, we choose this last option, as it might reinforce the similarity between logs with common patterns. We kept the proposed preprocessing for the other methods. This preliminary step is considered as crucial to the improvement of log parsing results (He et al., 2016a).

Hyper-parameters optimization. For the parametric methods, including METI*N*G, the values of the hyper-parameters need to be set. The LogPAI evaluation proposed default parameter setting based on a 10-run grid-search. To obtain a better scanning of the possible configurations, we perform a denser grid-search optimization : we run the parametric methods several times in a grid-search way and retrieve the best accuracy result recorded over 120 combinations for each log parser and each dataset. To build the grid and select the values to experiment, we took the minimum and maximum of the observed values for each parameter and divide the grid-search so as to run, for each log parser, 120 combinations. Therefore, we sometimes found better parameter settings than the ones proposed in LogPAI. We census in Appendix A.1 all the divergences between the tool's default parameter settings and ours. We also census the best parametric configuration for our new method, METI*N*G.

Results retrieved. To assess the log parsing results, we first show the best accuracy results obtained by grid-search for all the methods selected over the different datasets. We then study the sensitivity of the parametric methods to the values of their hyper-parameters. Indeed, parametric

methods are recognized to be more accurate and robust than parameter-free ones, and we want to show that this robustness is directly linked to the modulation power of the algorithm, induced by its high sensitivity to its hyper-parameters. Nonetheless, if METI*N*G is sensitive to its parameters, a particular attention should be taken to provide a solution to set their values.

4.1.3 Baselines

We select the most promising methods among the state-of-the-art algorithms as baselines, according to the first results of LogPAI's evaluations (Zhu et al., 2019), namely Drain (He et al., 2017), IPLoM (Makanju et al., 2012) and Spell (Du and Li, 2016). To assess our proposition on the influence of hyper-parameters on parsing results, we also select the recent parameter-free algorithms MoLFI (Messaoudi et al., 2018) and Logram (Dai et al., 2020). Logram is also based on the study of n-grams, with however a very different process than METI*N*G (as presented in section 2.1). By selecting a fewer set of methods, we conduct a further analysis of the log parsing results, deciphering the typical errors and weaknesses of the methods.

4.2 Parsing results

Table 4.1 presents the best accuracy results obtained by a grid-search optimization for the selected algorithms on the datasets. We observe that our log parser, METI*N*G achieves better and more steady results in average than the state-of-the-art propositions, with some impressive improvements compared to the existing results, e.g. +47.3% for Pr (Proxifier), +22.3% for Lin (Linux). In fact, these datasets are difficult to parse for most of the log parsers, in opposition to some other datasets that are commonly well-parsed, regardless of the method used (like Apache or HDFS). We also observe that the two parameter-free methods MoLFI and Logram render highly disappointing and unsteady results (with very high standard deviations, up to 0.29), due to their weak modulation power, which confirms the state-of-the-art knowledge that the use of hyper-parameters is key to adapt to the greatest number of datasets, and assure robustness. For this reason, we focus our analysis on the promising parametric methods.

For the OBIS logs, the overall results are even more spread, with a large advantage for METI*N*G, when using a generic preprocessing (Table 4.2). Similarly to the set of public datasets, some datasets seem to be easy to parse, with overall good results independantly of the method used, like the Application-4 or Application-6. Conversely, some datasets are globally more challenging to parse : none of the parser manages to parse the Application-3 correctly, while only METI*N*G succeeds in parsing the Application-7, with some impressive improvement (+65.5%). Nevertheless, the globally mitigated results for some datasets, along with the important variance of accuracy (high standard deviations) encourage us to improve the preprocessing.

Table 4.3 perfectly illustrates the assumption that the preprocessing has a critical impact on the final result. Here, all of the methods have significantly better results. 3 of the 4 methods are improved of at least 12% on average, while Spell and METI*N*G have reached very low standard deviations. There is still a clear advantage for METI*N*G on average, which reaches a quite impressive mean accuracy of 94.8%. We conclude from this study that METI*N*G is the only method that is able to obtain satisfactory results without domain knowledge. It is however, like the other meth-

Dataset	Drain	IPLoM	Spell	MoLFI	Logram	METI*N*G
Operating System						
Mac	**0.859**	0.673	0.757	0.636	0.744	0.824
Windows	**0.997**	0.684	0.989	0.406	0.974	0.996
Linux	0.690	0.676	0.639	0.284	0.201	**0.922**
Distributed System						
OpenStack	0.881	0.871	0.806	0.213	0.246	**0.969**
HDFS	0.998	**1.000**	**1.000**	0.998	0.930	**1.000**
Zookeeper	**0.988**	0.984	0.964	0.839	0.955	0.965
Spark	0.920	0.920	0.919	0.418	0.916	**0.996**
Hadoop	0.948	0.955	0.778	**0.957**	0.920	0.911
Supercomputer						
ThunderBird	**0.958**	0.663	0.934	0.646	0.554	0.931
BGL	**0.973**	0.944	0.787	0.960	0.805	0.889
HPC	0.901	0.829	0.654	0.824	**0.978**	0.918
Mobile System						
Android	0.913	0.712	**0.919**	0.788	0.674	0.911
Healthapp	0.780	0.890	0.639	0.440	**0.981**	0.688
Server application						
OpenSSH	0.788	**0.871**	0.554	0.500	0.556	0.555
Apache	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Standalone software						
Proxifier	0.527	0.517	0.527	0.013	0.504	**1.000**
Global metrics						
Mean	0.883	0.824	0.804	0.620	0.746	**0.905**
Std	± 0.126	± 0.145	± 0.158	± 0.297	± 0.256	± **0.120**

Table 4.1 – Best accuracy results (in %) of the studied methods on the 16 reference datasets. Global metrics : the mean and standard deviation of accuracy among the datasets.

	Drain	IPLoM	Spell	METI*N*G
Application-1	**0.940**	0.846	0.887	0.937
Application-2	**0.736**	0.729	**0.736**	**0.736**
Application-3	**0.375**	0.351	0.359	0.362
Application-4	**1.000**	0.999	0.987	**1.000**
Application-5	0.718	0.500	0.747	**0.822**
Application-6	**1.000**	0.986	0.983	0.974
Application-7	0.258	0.258	0.319	**0.974**
Mean	0.718	0.667	0.717	**0.829**
Std	± 0.277	± 0.278	± 0.256	± **0.210**

Table 4.2 – The best accuracy results for the parametric methods on 7 datasets of the OBIS. Log parsing was executed with a generic preprocessing.

	Drain	IPLoM	Spell	METI*N*G
Application-1	**0.958**	0.904	0.725	0.937
Application-2	0.812	0.729	0.812	**0.813**
Application-3	0.989	0.989	0.677	**1.000**
Application-4	**1.000**	0.986	0.963	**1.000**
Application-5	**0.917**	0.506	0.914	0.912
Application-6	0.981	0.564	0.965	**1.000**
Application-7	0.258	0.258	0.912	**0.974**
Mean	0.845	0.705	0.852	**0.948**
Std	± 0.247	± 0.256	± 0.107	± **0.064**

Table 4.3 – The best accuracy results for the parametric methods on 7 datasets of the OBIS. Log parsing was executed after a specific and appropriate preprocessing.

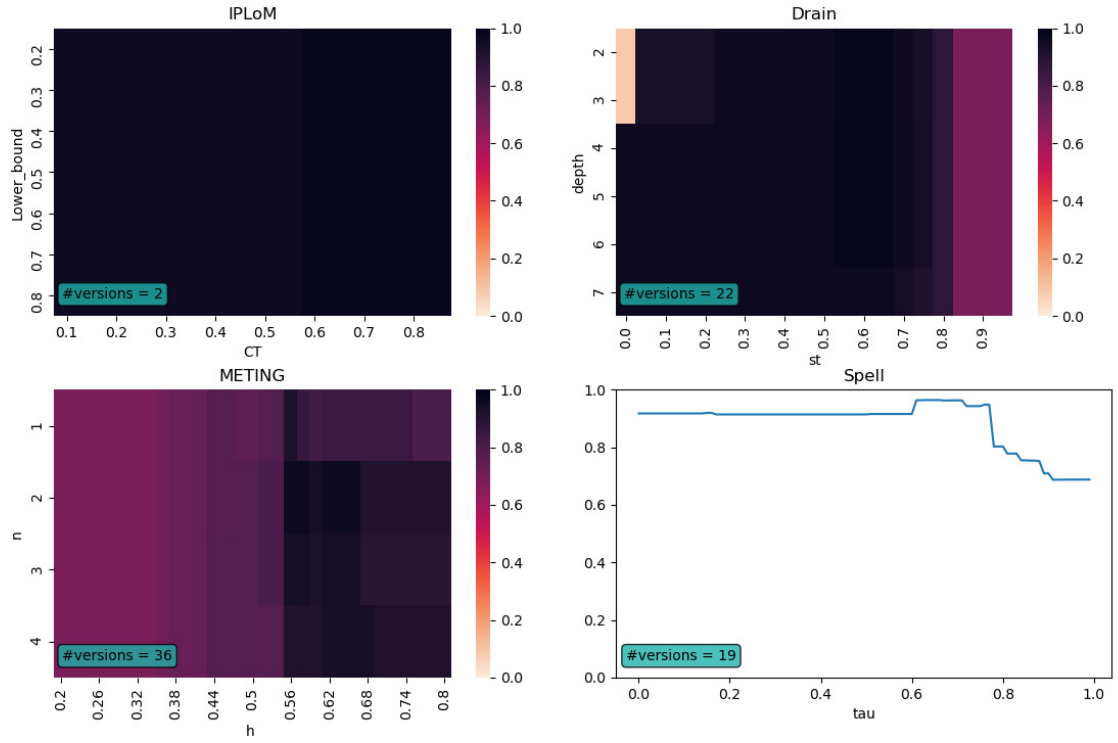
ods, positively impacted by the addition of domain knowledge through the preprocessing choices. On the OBIS, METI*N*G is undeniably the most reliable method to use.

4.2.1 Modulation power

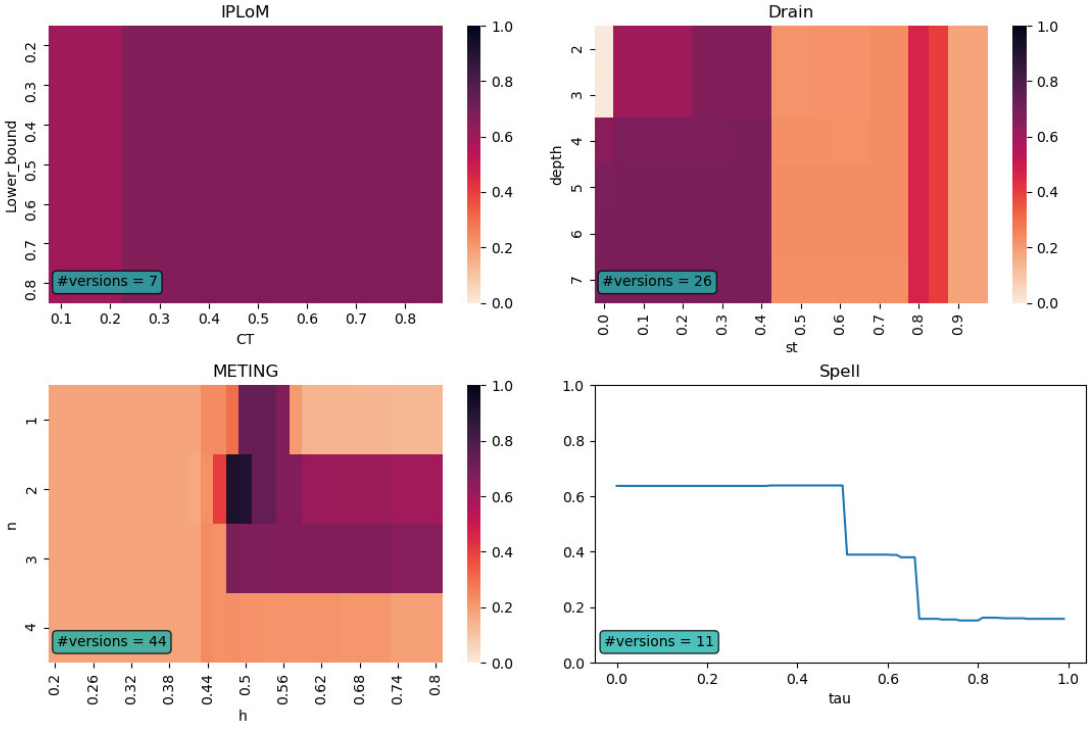
To discuss the impact of modulation power on robustness, we study the sensitivity to hyper-parameters for the four parametric methods : Figures 4.1a and 4.1b show the evolution of accuracy according to hyper-parameter values with (i) heatmap representations for the 2-parameter methods IPLoM, Drain, METI*N*G; (ii) a curve plotting for Spell and its unique hyper-parameter. We experiment on a challenging dataset, Linux, and a globally well-parsed one, Zookeeper. For each dataset and for all methods, 120 combinations are tested. IPLoM seems totally insensitive to its hyper-parameter `Lower_bound`, and poorly sensitive to `CT`. This lack of modulation explains its heterogeneous results : it provides 2 excellent results in the Zookeeper dataset (see the #versions indicator), and 7 low-accuracy configurations for Linux logs. Spell presents a slightly better sensitivity towards its unique hyper-parameter τ, allowing the algorithm to browse several parsing possibilities (19 configurations for Zookeeper). Drain also shows a more important number of combinations (26 for Linux) with a certain sensitivity to its hyper-parameter `st`. Yet, both methods show poorly-smooth variations, with long homogeneous regions (wide zones of uniform color or plateaus) and rough frontiers (brutal changes in color, or drops of the curve). For instance, in Linux, Drain presents large homogeneous regions for $0.1 < st < 0.4$ and $0.4 < st < 0.8$ while showing brutal changes in $st = 0.4$ and $st = 0.8$. Similarly, in Linux, Spell curves presents a large plateau, that drops suddenly. This roughness explains their inability to scan interesting configuration regions in the Linux dataset. METI*N*G appears to be the only method able to reach such successful parsing configurations, thanks to its important sensitivity to its two hyper-parameters and the important number of configurations proposed (44 configurations for Linux).

4.2.2 Error analysis through parameter requirements

We analyse the parsing errors made by the methods in order to better understand their behaviours and limitations. Rather than simply censusing the splitting errors, we emphasize and



(a) Zookeeper



(b) Linux

Figure 4.1 – Parameters' influence on accuracy with heatmaps for 2-parameter methods (IPLoM, Drain and ours) and curve plotting for Spell (which only has one parameter) over two datasets. The accuracy value is represented by (i) the color scale in the heatmaps, (ii) the y-axis on the curve plotting. `#versions` : the number of distinct clustering versions proposed by the method on the two reference datasets.

explain some typical and unavoidable mistakes of the different methods. We focus on the more accurate and robust parametric methods. We show that (i) the strong syntactic assumption are invalidated in some datasets, (ii) the hyper-parameter optimization of these parametric methods obliges the methods to arbitrate between groups of logs.

Logs of different lengths. The framework contains examples of groups gathering logs of different lengths (`E8` in Pr, `E146` in TB). Since Drain and IPLoM rely on the assumption that logs of a same group necessarily have the same length, they fail in parsing these datasets. In turn, Spell and METI*N*G opt for a more flexible representation of logs that allows to group logs of different lengths.

Variable first words. Drain assumes that the fix tokens of a log are positioned at the beginning, and imposes d common first words in each group, where d is a hyper-parameter. Yet, some groups violate this assumption, like in the Openstack dataset, with the group `E11` :

```
[instance: 54b44eb-2d1a-4aa2-ba6b-074d35f8f12c] Terminating instance
[instance: 17288ea8-cbf4-4f0e-94fe-853fd2735f29] Terminating instance
```

Since these logs only have one first word in common, d must be set to 1. Otherwise, the two logs would be separated. However, this strong constraint prevents the separation of other groups of the dataset (`E22` and `E20`). This example traduces a problem of parameter optimization, with the necessity to perform an arbitrage. METI*N*G succeeds in parsing this dataset since it does not impose any constraint on the positions of fix and variables words.

Frequent variable parts. Due to the limited size of the datasets, some examples appear to belie the frequent pattern mining hypothesis. In Healthapp, the 3-gram "0 0 0" is globally more frequent than the fix parts of the groups `E39`, `E47`:

```
E39 = onExtend:1514038813000 0 0 0
E47 = REPORT : 0 0 0 0
```

In this case, METI*N*G chooses this n-gram to perform the split, and fails in retrieving the corresponding groups. For its part, Spell groups logs according to their LCS rate : the length of the longest common sequence divided by the average length of logs. Since these logs have a high LCS rate, Spell also fails in separating them. Nonetheless, we observed, for the two methods, that a better preprocessing, e.g. removing the numbers, would lead to a perfect score of 100% accuracy.

Template inclusion. In some datasets, the template of a group is included in the template of another, such as in OpenSSH with the groups `E19` and `E20`, defined by the patterns:

```
E19 = pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=.*
E20 = pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=.*
user=.*
```

Since the LCS of these two logs is the whole template of `E19`, Spell regards these logs as very similar and fails in separating them. In METI*N*G, the selected n-gram to separate these logs necessarily comes from `E20`. Yet, in this example, `E20` is minority compared to `E19`, so the splitting n-gram has a very low frequency and triggers the stopping criteria. Drain also fails in splitting this group, since the logs share numerous common first words. IPLoM is not concerned by this issue and succeeds in parsing the dataset.

Alternation fix/variable words. Some group patterns in the dataset show an important alternation, with consecutive fix and variable words, such as the group E8 of Proxifier, presented in logs L_1 and L_4 of Figure 3.2(a). Spell finds a low LCS rate for these logs and separate them. IPLoM separates them since they have different lengths. Drain would not stand a chance to gather these logs : they have different lengths and start with a variable token. Only METI*N*G is flexible enough to retrieve the important similarity of these logs thanks to their common 2-gram *received lifetime*. This explains METI*N*G's tremendous improvement of accuracy score for the Proxifier dataset.

Other general observations. Apart from these situations, we observed that IPLoM tends to have a limited division power. Indeed, IPLoM gathers logs from the same length, then splits them only if they have no common words, and performs a last binary split according to its hyper-parameters. Hence, the method sometimes lacks of division opportunities and logs might be incorrectly gathered (E7 and E8 in ZooKkeeper). In opposition, METI*N*G has a dedicated hyper-parameter, h, which can modulate its division power.

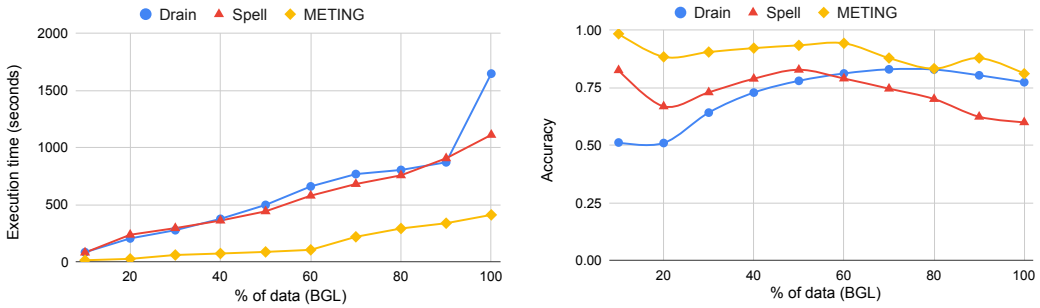
Furthermore, Spell proceeds in a online way and builds the LCS of a group on-the-fly. Hence, it is very dependant on the order in which logs appear, and commits errors when a variable part is chosen as LCS. Drain also suffers from this dependency : its syntactic assumptions are actually used to point the arriving logs to the right groups. Our solution to extend METI*N*G in an online fashion cope with these issues thanks to a first offline pass to build a first version of the dendrogram.

4.2.3 Efficiency and scalability

We study the ability of METI*N*G to scale to huge datasets, and to the online functioning. To do so, we compare it to the most accurate and efficient log parsers, that can also be applied online, namely Drain and Spell. We use the BGL dataset which contains around 4 700 000 logs, and for which LogPAI provides a full set of labels.

Efficiency and scalability on huge datasets. We first measure the execution time of the three methods for different sizes of the datasets (percentages of the whole dataset). We optimized the hyper-parameters on the full dataset, and applied the same configuration to all the subsets. Figure 4.2a shows that METI*N*G clearly outperforms both Drain and Spell for every size of data, in terms of execution time. This high efficiency is explained by the offline functioning of METI*N*G : at the beginning, all the logs are preprocessed so as to replace some regular expressions (numbers, dates...) by a unique token. Then all the logs with the same signature are treated as a unique sample. The number of unique logs is far inferior to the number of logs, especially in such a huge dataset. The complexity of METI*N*G is proportional to the number of unique log (after the preprocessing), while the online methods parse all the logs one after the other. Their complexity is at least linear.

Figure 4.2b also shows that METI*N*G presents a better accuracy than the two other methods. In the study of the 2000-log subsamples, we assumed that METI*N*G results on the BGL could be improved with a more representative and bigger dataset, which seems to be confirmed. Besides, the optimization performed on the full dataset seems to fit even the first subsample. This stability



(a) Execution time of log parsers on different percentages of the BGL dataset.

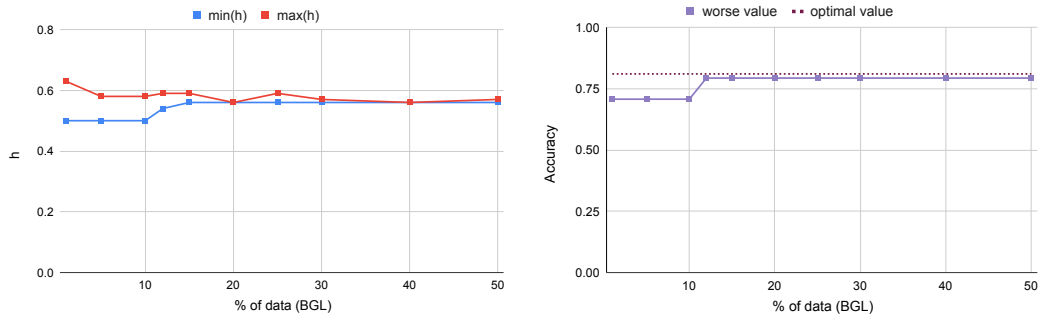
(b) Parsing accuracy of log parsers on different percentages of the BGL dataset.

Figure 4.2 – Study of the efficiency and accuracy of Drain, Spell and METI*N*G on the full dataset of BGL, with subsamples of different sizes. The hyper-parameters are optimized on the full dataset version (100%).

is an important feature : if the parameter optimization is stable, it becomes feasible to perform it on a small subsample of data (which can even be manually labelled) and to apply it on full large datasets.

Parameter optimization stability. Figure 4.3a presents the best h coefficients obtained by parameter optimization performed on different sizes of the dataset (the best n is always $n = 2$). The optimal value for the full dataset (namely $h = 0.56$) is systematically included among the best values of smaller datasets, which means that the behaviour of METI*N*G with this configuration is stable. Nonetheless, if we optimize h based on a subset of 1% of the logs, we are likely to choose other values, that are not optimal for the full dataset (e.g. $h = 0.50$). We measure the impact of this error by calculating, for each subsample, the worse accuracy score of the full dataset. For instance, the subsample of 1% promotes, among others, the value $h = 50$, which leads, for the full dataset, to an accuracy of 0.707. Since the best accuracy of the full dataset is 0.810, we measure a loss of 0.103. Figure 4.3b presents this loss : with only 12% of the dataset, we can limit this loss to 0.017. It becomes feasible to perform a semi-manual labelling of such a sample, to optimize the hyper-parameters on it.

Semi-manual labelling algorithm. We propose an algorithm for semi-manually labelling the logs of a small subsample. The user can manually label the data thanks to regex matching, based on domain knowledge. Instead of manually attributing the event type to logs one by one, the user shall propose a template for the first log, express it as a regex, and automatically label all the logs matching this template. Let us study the group `E8` of the Proxifier dataset, presented in Figure 3.2 with the log L_4. The user might propose the template `<*> close, <*> bytes sent, <*> bytes received, lifetime <*>`. All the logs matching this template are instantly labelled to the same group. However, log L_1 from `E8` does not match this first proposition of template. The user can then refine its template to match both. Algorithm 26 describes this proposition. Each task noted $task_u$ denotes an action performed by the user. $\texttt{check\_similar}_\text{u}$ implies that the user shall check within the template list T if the log could be matched by a template of the list, with a slight modification of this template. $\texttt{propose\_template}_\text{u}$ invites the user to propose a template for a set of logs. All the other actions are automated, notably `match` that checks if a regex matches a log. By



(a) The minimum and maximum value of h that generate the best configuration (with $n = 2$) for subsamples of BGL of different sizes.

(b) Comparison between the best accuracy obtained by optimization on the full dataset, and the worse accuracy of the full dataset, obtained by optimization on subsamples.

Figure 4.3 – Study of the stability of the hyper-parameters of METI*N*G on the full dataset of BGL. **(a)** The area between the lines covers the best h values ($n = 2$ is systematically the best value). For each subsample, the randomly choosing a configuration among the best can lead to a wrong choice for the final optimization. **(b)** The worse accuracy obtained on the full dataset by choosing among the best configurations of the subsample.

incrementally removing the already-labelled logs and despite these adaptation steps, the user can efficiently label a small amount of log data, which is fortunately sufficient for METI*N*G to learn the best parameter setting. In addition, this step can be used to retrieve the regex tokens to delete for the preprocessing step (numbers, IP addresses...).

Online extension. Finally, we assess the scalability of our method to the online functioning. While the structure of the dendrogram makes the online adaptation feasible in an algorithmic perspective, the online parsing of logs imposes computational efficiency constraints : the time needed to parse an arriving log must be lower than the time separating two arriving logs in average. Table 4.4 shows the average execution time required to parse one log with our online extension. We observe that it is 20 times slower than Drain and Spell, since it now needs to evaluate the splitting condition for each new log. Nonetheless, it is still sufficiently fast to parse BGL logs. It is even fast enough to parse HDFS, the dataset with the highest generation rate.

4.3 Template extraction results

As a proof of concept of our template extraction method, we propose to display an example of the generated output of our method, compared to the basic templates proposed by LogPAI, both for the labelling of the group template, and as the output result of the implemented methods.

Figure 4.4 presents the output XML generated by our template extraction method on the group `E6` of the HDFS dataset (which can be obtained by the ground truth labels, or by our log parser). While the template label proposed by LogPAI is simply `block* namesystem.addstoredblock blockmap updated <*> is added to <*> size <*>`, our method aligns the variables parts, and detects their types. It has for instance recognized an IPv4 address, and an integer variable. For the categorical variables, our tool plots the 5 most frequent values encountered, along with their frequencies. For the numerical values, some statistics are retrieved (minimum, maximum, mean). This representation can be highly valuable for the end-users, since it can help them to quickly

```
input  : L : a small list of logs
output: T : A list of objects of the class Group_template
T ← list();
match-is-found ← False ;
while |L| > 0 do
    ℓ ← pop_head(L) ;                                   // extract first log
    i ← 0;
    while i < |T| and not match-is-found do
        t ← T[i];
        if match(t, ℓ) then
            t.logs.append(ℓ) ;              // add ℓ to the list of logs matching t
            match-is-found ← True ;                              // template found
        end
    end
    if not match-is-found then
        t ← check_similar_u(ℓ, T);
        if t ≠ None then
            // update template and add the new log
            t.logs.append(ℓ);
            t.template ← propose_template_u(t.logs);
        else
            // create new Group_template and add to the result list
            t ← Group_template(ℓ);
            T.append(t);
        end
    end
end
return T
```

Average parsing time			Generation rate	
Drain	Spell	METING	BGL	HDFS
2.24E-04	2.07E-04	4.52E-03	3.90E+00	1.25E-02

Table 4.4 – Average time (in seconds) for Drain, Spell and METING (online) to parse a new arriving log, compared to the generation rates of BGL and HDFS. A method is able to parse a dataset in an online fashion if its parsing time is inferior to the generation rate.

```
<Application app_name="HDFS" Log_count="313">
  <log-entry Application="HDFS" nb_logs="313" is_periodic="False" event_Code="E6">
    <Exemple String="block* namesystem.addstoredblock  blockmap updated  10.251.43.115 50010 is added to blk_3050920587428079149 size 67108864"/>
    <fixed-part value="block* namesystem.addstoredblock blockmap updated "/>
    <variable-part Type="IPV4">
      <modality value="10.251.194.102" nb_occurrences="1" frequency="0.0032"/>
      <modality value="10.250.5.237" nb_occurrences="1" frequency="0.0032"/>
      <modality value="10.251.123.132" nb_occurrences="1" frequency="0.0032"/>
      <modality value="10.251.38.214" nb_occurrences="1" frequency="0.0032"/>
      <modality value="10.251.198.33" nb_occurrences="1" frequency="0.0032"/>
    </variable-part>
    <fixed-part value="50010 is added to "/>
    <variable-part Type="STRING">
      <modality value="blk_-7603740654202748654" nb_occurrences="1" frequency="0.0032"/>
      <modality value="blk_4984720043834039361" nb_occurrences="1" frequency="0.0032"/>
      <modality value="blk_-524128780228307052" nb_occurrences="1" frequency="0.0032"/>
      <modality value="blk_-9181958042427679309" nb_occurrences="1" frequency="0.0032"/>
      <modality value="blk_-3184700311670257792" nb_occurrences="1" frequency="0.0032"/>
    </variable-part>
    <fixed-part value="size "/>
    <variable-part Type="INTEGER" min="3530010.0" max="67108864.0" mean="63644114.060702875"/>
  </log-entry>
</Application>
```

Figure 4.4 – An example of the XML output generated by our template extraction method for the group E6 of the dataset HDFS.

detect an abnormal behaviour in the logs, either because a categorical value has appeared, or because a numerical value is out of a confidence interval of its nominal distribution.

5 Extension to the stemming task for text mining

The description of a log parser that groups logs and extract the templates of the groups can be transcribed to the task of stemming in the context of text mining. While log parsing aims at grouping logs, and extract their common words, stemming consists in grouping words and extract their common letters. In the log parsing context, the frequent n-grams are the words that we assume to constitute the log template. In the stemming context, the frequent n-grams are the letters that form the *stem*, the morphological root of a word.

5.1 Context

Stemming is the process of replacing a word with its morphological root. The stem can be a substring or concatenation of substrings of the word, or even a modified substring. Beyond the stem itself, the stemming process generates *conflation groups*, groups of words with the same stem. The grouping of words automatically reduces the size of the vocabulary since it removes the morphological variants of the words of the studied set (Jivani et al., 2011). Stemming relies on the assumption that morphologically related words have similar meanings and represent the same concept. As such, stemming has become an almost essential preprocessing step in numerous text mining tasks, like information retrieval, or natural language processing.

The most used stemmers consist in applying lexical rules on words to remove irrelevant parts. These rules are based on the morphological forms of words and are hence necessarily specific to the language. However, if the rules of known and widely spoken languages are abundantly specified and established, it is difficult to find an implementation of these algorithms for languages or dialects with little automatic tools. Moreover, the number of rules necessary, as well as the quality and the effectiveness of the stemmer strongly depends on the language on which it is applied (Kraaij and Pohlmann, 1996). Linguistic studies commonly distinguish two types of languages (Moral et al., 2014) :

- Analytical (or isolating) languages, with few morphological structures, where variants are more likely to be expressed by help words than by word variations (e.g. Chinese, Vietnamese, modern English);
- Inflectional languages, with strong morphological structures, including :
  - Fusional languages, such as Latin (French, Italian) or Germanic (German, Dutch, Swedish, (Braschler and Ripplinger, 2004)) languages. They use many prefixes and suf-

fixes to change the grammatical nature or even the meaning of words;
— Agglutinating languages, such as Finnish, Japanese, Berber or Basque. In these languages, the suffixes not only represent inflections, but can also be the concatenation of other morphemes. For example, in Basque, *etxe* means *house* and *etxean* means *in the house*.

Jivani et al. (2011) and Braschler and Ripplinger (2004) affirm that stemming is more effective on languages with complex morphologies like inflectional languages. Yet, on such languages, a large number of rules would be necessary, in order to remove all the affixes, which would make the stemmer inflexible, difficult to interpret and to implement. Stemmers are more interesting for flexible languages, but rule-based stemmers are difficult to apply on such languages.

Corpus-based stemming represents an interesting alternative. The words are processed within a text corpus, grouped by similarity and a stem is extracted from the created group. As opposed to rule-based stemmers, these methods offer a unique algorithm regardless of the language used, and modulate their behavior using parameters. As an example of the modulation power of corpus-based method, Frakes and Fox (2003) define the *strength* of a stemmer as its ability to group many words under the same stem (we speak of strong or light stemming). While corpus-based stemmers consist in creating the conflation groups, it remains however necessary to generate the stem itself : it can be crucial to provide readable and interpretable stems for tasks like query expansion or dictionary look-ups (Hull, 1996).

Finally, stemmers based on corpus of literature rely on grouping methods with significant spatial complexities. They often require pairwise word distance calculations, which implies the implementation of quadratic matrices (Adamson and Boreham, 1974; Majumder et al., 2007). In the case of word processing, these matrices generally reach large dimensions, since most Western languages have $\tilde{1}$00 000 words. This strong constraint on resources makes these algorithms unsuitable for large corpora, where a large part of the words of the language is likely to be used.

In this chapter, we propose RFreeStem, an adaptation of METI*N*G to the context of stemming. RFreeStem is based on corpus, and can be applied to any language. It relies on the same computationally efficient heuristic than METI*N*G. The template extraction of METI*N*G can also directly be used to extract the stem of the words. Section 5.2 describes the related work on stemming. In section 5.3, we detail the required modifications to adapt our log parsing contribution to the context of stemming. We compare RFreeStem to the state-of-the-art reference, Porter (1980) in section 5.4.

5.2 Related work

This section lists some of the most important studies on stemming, including both rule-based and corpus-based methods.

5.2.1 Rule-based methods

Most of the popular stemming methods are based on the application of rules. These rules generally aim to remove the affixes (suffixes and prefixes) in each word individually, to obtain the stem. In 1968, Lovins (1968) proposed a first method of *suffix stripping*. Simple and fast, the algorithm matches the end of the word with the longest element in a list of suffixes and applies rules

to modify the corresponding ending. The Lovins algorithm is considered unreliable since its list of suffixes is based on a technical vocabulary. Technical and scientific terms, however, tend to derive from Latin languages and therefore to be very inflectional, whereas common modern English is rather isolating [1].

The most popular algorithm was proposed by Porter (1980). With 5 successive steps, it is slower than its 2-step predecessor, but was found to be empirically more accurate (Willett, 2006; Paice, 1996), especially for information retrieval tasks. Initially, Porter's algorithm only defined rules for the English language. However, its popularity has motivated the implementation of variants in different languages (project `snowball`, Porter (2001)). However, it remains difficult to find such implementations for poorly endowed or very inflectional languages (for example, Bahasa Indonesian (Tala, 2003), Dutch (Kraaij and Pohlmann, 1994)).

The S-stemmer (Savoy, 2006) offers a light stemming, which only deals with the removal of plural forms, and therefore has little compression power. Conversely, Paice (1990) created a strong stemming algorithm, which successively applies deletion and replacement rules. Jivani et al. (2011) find that this method tends to generate over-stemming errors. (Dawson, 1974) adapted Lovins' method with many more suffixes and rules, which unfortunately makes the algorithm inflexible and poorly interpretable (Jivani et al., 2011). Krovetz (1993) proposed an approach based on inflections and derivations, with its stemmer *KSTEM*. The method relies on a lexicon without inflection to deduce and remove inflections, before analyzing derivations (variants that change the grammatical nature of words). The author recognized that its accuracy is highly dependent on the lexicon provided. We even add that finding such an exhaustive lexicon is difficult for poorly endowed languages.

Rules-based methods are undeniably the most widely used in information retrieval, thanks to their algorithmic efficiency and the simplicity of their implementation (Harman, 1991). Nevertheless, adapting these methods to very inflectional languages would lead to a large number of necessary rules. Therefore, other research papers on stemming are inclined to study corpus-based approaches.

5.2.2 Corpus based methods

In order to cope with the strong linguistic dependence of rule-based stemming techniques, corpus-based stemmers are adaptable to different languages, and different datasets. Most of these methods are based on statistical studies of the text. Hafer and Weiss (1974) developed an algorithm which splits words into two parts : if the first part belongs to the text corpus, this first part becomes the stem. The splitting algorithm is based on the evolution, within the letters of the word, of the *successor variation* : the number of distinct characters that follow a string in all the words of the corpus. Yet, the resulting grouping of words strongly depends on the text corpus. Besides, systematically choosing the first part of the word as a stem seems to be a biased, language-dependent approach. The method was initially developed for English, which generally has more suffixes than prefixes, which is less true for Latin languages.

The N-Gram stemmer (Adamson and Boreham, 1974) uses a common string grouping method

1. The English language is actually slightly inflectional, due to its heritage from Old English. Thus, modern English is less isolating than languages like Mandarin or Indonesian (Moral et al., 2014).

Method	Rule-free	Assumption	Efficient
Lovins (1968)	✘	Rules	✓
Porter (1980)	✘	Rules	✓
Paice (1990)	✘	Rules	✓
Dawson (1974)	✘	Rules	✓
Krovetz (1993)	✘	Rules	✓
Hafer and Weiss (1974)	✓	Stem is at the beginning of words	✘
Adamson and Boreham (1974)	✓	Adaptation of frequent pattern mining assumption	✘
Majumder et al. (2007)	✓	Long suffixes	✘
Soricut and Och (2015)	✓	Stem is a unique block	✘
RFreeStem	✓	Adaptation of frequent pattern mining assumption	✓

Table 5.1 – Summary of the stemming methods according to desirable features.

to create groups of words : a hierarchical single-linked grouping of words. The word pairs distances are calculated with a distance of *Dice*, corresponding to the number of distinct shared bigrams — sequence of two consecutive letters. Using an unsupervised grouping method makes the stemmer dependent on the corpus being used, but also provides a very flexible way to manage the strength of the stemmer. Nonetheless, single-link clustering is known for its high computational complexity in space, due to the creation of a quadratic distance matrix. Likewise, *YASS* (Yet Another Stripping Stemmer, (Majumder et al., 2007)) groups words with hierarchical grouping methods. It also defines word distance metrics that encourage detection and removal of long suffixes. According to Jivani et al. (2011), long suffix search makes the method more suitable for languages which are inflectional and suffix-rich.

Finally, Soricut and Och (2015) proposed a hybrid method, since it aims to automatically learn rules, based on a corpus of text. The unsupervised learning approach makes this method flexible and potentially adaptable to many languages. However, the rules sought in the corpus only concern the deletion of affixes. This method is therefore based on the strong assumption that the stem forms a single block surrounded, possibly, by a prefix and a suffix. However, compound words do not verify this assumption, and are frequent in languages like German.

5.2.3 Summary of the existing methods

We summarize the existing methods for stemming in Table 5.1. From our reading of the literature we present some desirable features for a stemming method : (a) a rule-free method offers the possibility to apply the stemmer to multiple languages, (b) even for the rule-free methods, no strong language-related assumption should be used to create the groups, (c) distance-based stemmers are computationally demanding. As presented in column Rule-free of Table 5.1, only half of the presented methods are rule-free. Despite their reliance on corpus, which should enable a multi-language usage, these methods rely on strong morphological assumptions on words constructions, as presented in column Assumption. As for the log parsing case, we consider the

frequent pattern mining assumption as rather flexible. The only method that uses an adaptation of this version, Adamson and Boreham (1974), based on the n-grams similarity, still presents an important algorithmic complexity. We therefore propose RFreeStem, an adaptation of METI*N*G to the stemming context. RFreeStem is based on the learning of corpus and does not add any strong morphological assumption to its process. With its heuristic functioning, RFreeStem is also computationally efficient.

5.3 RFreeStem, an adaptation of METI*N*G for stemming

We notice that our proposition could be a valuable contribution to the task of stemming in text mining. We therefore propose RFreeStem, the stemmer that extends the method of METI*N*G. This contribution is published in (**?**) and (**?**).

5.3.1 Problem introduction

From our lecture of the literature, we identified several limits of the existing stemming methods, and detail here how RFreeStem can promote interesting features for a stemmer.

Stemmer flexibility. Corpus-based stemmer can adapt to multiple languages, contrary to rule-based ones. Indeed, corpus-based stemmer often rely on an unsupervised learning of text statistics within the corpus. The usage of parameters can also help improve the modulation power of the stemmers; especially, they can help control the strength of the stemmer. Generally speaking, the ability of a stemmer to adapt to multiple languages implies to avoid the usage of strong assumptions on the structure of words (assumptions on length of the suffixes, on the position of the stem in the word...). To answer this requirements, RFreeStem is based on a corpus, and adopts a hierarchical representation of words, based on the study of n-grams. Our stemmer builds a hierarchical organization of groups by successive divisions thanks to an unsupervised learning. RFreeStem, like METI*N*G, contains three hyper-parameters : (i) n, the size of the n-grams, (ii) ρ, a coefficient that manages the depth of the hierarchical representation, directly linked to the strength of the stemmer (ρ is correlated to h in METI*N*G), (iii) lim, the number of n-grams that are run in each division.

Computational efficiency. Most of the corpus-based methods that rely on word clustering calculate a pairwise distance matrix of the words. Similarly to the log parsers, the calculation and storage of such a quadratic matrix generates issues of spatial and temporal complexity. The heuristic used in RFressStem and described for METI*N*G is instead computationally efficient.

Availability of the stems. While rule-based stemmers directly generate the stems associated to the words, corpus-based methods are rather based on the clustering of words. Hence, the extraction of stems needs to be performed a posteriori, in a post-processing phase. METI*N*G proposes such a treatment for the template extraction, which is directly applied in RFreeStem to extract the fix letters, that constitute the stems, and the variable letters.

5.3.2 Required adaptions for RFreeStem

We detail here how to formally adapt the solution METI*N*G to the context of stemming. First, the log parsing problem consists in grouping logs together based on their words, while the stemming task consists in grouping words together based on their letters. Hence, when we talk about n-grams in the context of log parsing, we refer to sequences of consecutive words, while the n-grams in the stemming context are sequences of consecutive letters. With this translation, only a few more adaptions need to be performed to fit the contextual assumptions, and are described in this subsection.

Additional notations and definitions. For our definition of RFreeStem, in addition of the formerly mentioned notations, we use the absolute value of a number $x \in \mathbb{R}$, noted abs and defined by

$$\mathsf{abs}(x) = \begin{cases} x & \text{if } x \leq 0 \\ -x & \text{otherwise.} \end{cases}$$

We also use the *Dice* coefficient, defined in Kondrak (2005). This coefficient is a measure of distance between two words and relies on the comparison of their n-grams. We use this measure to evaluate the homogeneity of a group of words. For two words w_1 and w_2, the *Dice* coefficient is defined as :

$$Dice(w_1, w_2) = 2 \times \frac{|N(w_1) \cap N(w_2)|}{|N(w_1)| + |N(w_2)|}$$

To generalize this calculation to a group of words W, we propose the extension :

$$Dice(W) = |W| \times \frac{|\bigcap_{w \in W} N(w)|}{\sum_{w \in W} |N(w)|} \tag{5.1}$$

Especially, the value $1 - Dice(W)$ can be seen as a measure of the homogeneity of a group of words, in term of shared n-grams.

The scoring function of the n-grams. In METI*N*G, we simply assumed that the most frequent n-grams (sequence of words in a log) were the most likely to be fixed parts, according the frequent pattern mining assumption. This assumption is not necessarily verified for the stemming case. Indeed, the most frequent n-grams (sequence of letters in a word) of a lexicon of a language are often the most frequent suffixes and prefixes (for example *tion*). We therefore replace our former scoring function by the function ϕ, that attributes to any n-gram g the following score :

$$\phi(g) = \frac{1}{2} \times \left(1 - Dice(W_g) + \mathsf{abs}\left(\frac{|N(W_g)| - \ell_{ref}}{\ell_{ref}}\right)\right) \tag{5.2}$$

Here, we evaluate a n-gram in terms of quality of the group that would be generated if this n-gram is chosen. The *Dice* component measures the homogeneity of the group that would be formed if the n-gram g is used for the division, while the last member measures the relative difference between a reference size of an n-gram group ℓ_{ref}, and the actual size of this group. ℓ_{ref} shall be chosen so as to correspond to an ideal size, meaning the size of the group corresponding to the most frequent n-gram, with the exclusion of the frequent affixes. For instance, ℓ_{ref} can be the q-percentile of the word lengths distributions, where q is to be defined.

Stopping criteria. Since the scoring function was modified, we estimate that it already takes into account the diversity of the groups created throughout the process. Moreover, the *Dice* coefficient already provides a measure of homogeneity. Hence, we decide to replace the condition checked by the function `continue` by the following condition :

$$\texttt{continue}(W,G) \iff Dice(W) < \rho \cdot \frac{1}{|G|} \cdot \sum_{G_i \in G} Dice(G_i) \tag{5.3}$$

In this equation, W is a group of word, G is the partition proposed by the method `division` and ρ is a hyper-parameter that controls the division power of the methods — the usage of ρ in RFreeStem is similar to the usage of h in METI*N*G.

5.4 Validation

This section presents the evaluation of RFreeStem. We run RFreeStem and the stemmer of Porter (1980), the current state-of-the-art reference, on raw textual data before applying an automatic text categorization task to it. We compare the two methods in terms of improvement of text categorization performance.

5.4.1 Experimental framework

Our evaluation is divided into two experimental studies : (i) first, we evaluate the effectiveness of our stemmer on the English language by comparing two tasks : automatic text categorization (**Task 1**) and sentiments analysis (**Task 2**); (ii) we then compare the improvement produced by stemming for different languages, on the sentiment analysis task. For each experimental study, we will compare the different versions of our algorithm (due to the possible values of its hyper-parameters) with (i) Porter's version and (ii) the version without stemming. For the sake of fairness in the comparison of results, we randomly select 2000 documents for each dataset, while respecting the initial proportions of the different classes.

5.4.1.1 Presentation of data and mining tasks

Task 1. Text categorization is a popular automatic text processing task that aims to automatically determine the class of a document in a corpus. To test the effectiveness of our stemmer as a preprocessing step for text categorization, we applied our method to the AustLII dataset [2]. We study a set of 2000 citations from different XML files. These files contain both the legal citation text and a manually tagged category, from the following :

class	cited	referred to	applied	followed	considered	discussed	Others(13)
#documents	938	351	228	169	129	88	97
#terms	16 609						

2. `http://www.austlii.edu.aucorpusofcasereports/` (Galgani et al., 2012) uses it for categorization

Language	Source	Positive		Neutral		Negative		#terms
English	Movie reviews	1000	50%			1000	50%	7 736
French	Amazon reviews	1634	81.7%	177	8.9%	189	9.4%	13 656
German	Amazon reviews	1600	80.0%	244	12.2%	156	7.8%	30 986
Urdu	Tweets	595	29.7%	883	44.2%	582	29.1%	7886

Table 5.2 – Presentation of the datasets for **Task 2** - 2000 documents in each case.

The dataset is strongly unbalanced since 46.9 % of citations are labeled as *cited*, while two-thirds of the classes (grouped in *Others*) represent only 4,9 % of the data. Unbalanced datasets are recognized to be more difficult to process, so it will be interesting to see if our stemmer simplifies this task.

Task 2. Sentiment analysis has become a very popular task, especially with the huge expansion of social networks and their textual data. The goal is to assess feelings in comments or reviews written by humans. To assess RFreeStem's ability to improve the results of this complex task, we choose a frequently used movie review dataset [3]. Originally mentioned by Pang and Lee (2004), the corpus is still commonly used today (Ba et al., 2016; Shen et al., 2018).

As the literature agrees that stemming is more suited to inflectional languages, we propose to compare our stemmer on similar datasets from different languages. We therefore studied Amazon reviews in French and German [4] for the sentiment analysis task (**Task 2**). Since the products are rated with a score of 1 to 5, we extract categorical labels with the following simple transformation rule : if the score is strictly greater than 3, assign to the positive class, otherwise, if it is strictly lower to 3, assign to the negative class, otherwise, assign to the neutral class.

Finally, since our method is based on a corpus, it is not dependent on the language and it can be applied to any language, in particular those which are poorly endowed with resources and tools. We propose to study a Roman Urdu dataset [5] for sentiment analysis with 3 classes (**Task 2**).

Table 5.2 presents the distribution of document classes for each dataset of **Task 2**, as well as the number of terms initially present. While the English corpus corresponds to a binary and balanced classification, the other three datasets contain, in addition to the positive and negative classes, the neutral class. Amazon's datasets (German and French) are highly polarized with over 80% of reviews being positive. Conversely, the Urdu dataset is rather similar to a Gaussian distribution, with comparable positive and negative classes, and a higher proportion of neutral comments. Finally, there is a great disparity between the number of terms for each language. We partially attribute this diversity to the inflectional character of French and German, while English and Urdu are isolating languages. In addition, German composes new words by concatenating different morphisms, which tends to increase the size of the vocabulary used. It is therefore very likely that stemming will have a positive impact on the representation of these two languages.

3. `http://www.cs.cornell.edu/people/pabo/movie-review-data/`
4. `https://s3.amazonaws.com/amazon-reviews-pds/readme.html`
5. http://archive.ics.uci.edu/ml/datasets/Roman+Urdu+Data+Set, Sharf and Rahman (2018)

5.4.1.2 Evaluation metrics

Our evaluation consists in observing the improvement of the studied text processing tasks induced by stemming. To measure this improvement, we calculate different evaluation measures. First, we measure the accuracy and the F1-measure of the final classification results. Finally, we are interested in the compressive power of the stemmer. To measure it, we use the ICF (Index Compression Factor) (Frakes and Fox, 2003). The ICF is expressed as $\frac{m-s}{m}$ where m is the total number of words in the corpus, and s the number of different stems proposed (the number of groups created).

5.4.1.3 Experimental protocol

The process we apply is as follows : for each dataset, and for each stemming method evaluated, we apply the stemming method to the raw dataset and therefore generate a lexicon where each entry is a pair {word : stem}.

We then run a supervised classification algorithm on the data by replacing each word with its stem (or keeping the original words, for the version without a stemmer). We implement a naive Bayesian classifier, trained on 70 % of the data, respecting the proportion of the labels. The remaining 30 % are predicted with the trained classifier, and the results are compared to their class labels. Documents are represented by a term-documents matrix, to which the classifier is applied. We only select the most frequent terms, which are more likely to be significantly represented and learned during the training process. To do so, we define a minimum frequency required, f_{min}, and remove the terms which are present in less than f_{min} documents. This frequency-based filtering is applied after the stemming phase, and before the classification. A method which presents satisfactory results on small values of f_{min} has the advantage of adopting a precise and efficient representation. In addition, we run each classification calculation several times (20 iterations), since the Bayesian classifier is based on stochastic calculations.

As for the log parsing study, we measure the sensitivity of our method to its hyper-parameters. To select the best methods, we consider several criteria :

— The mean value of the F1-measures over the f_{min} values;
— The best value of the F1-measures over the f_{min} values;
— The mean value of the accuracy over the f_{min} values;
— The best value of the accuracy over the f_{min} values.

For the needs of our algorithm, we fix by experimental observation the two values of lim and q — used to modulate respectively the visiting of n-grams and the ideal size of the groups N_g— in order to exclude respectively the n-grams having a too low score, and the n-grams representing current affixes. We take lim so that we go through 75% of the n-grams at each iteration, and $q = 99$ (the ideal size ℓ_{ref} of $|W_g|$ is that of the percentile 99% of the different $|W_g|, g \in N_W$).

We run each configuration of our algorithm 20 times for each f_{min} studied, and vary f_{min} from 1 (all terms are kept) to 10 (only terms appearing at least 10 times are kept). Each configuration of RFreeStem corresponds to a value of the pair of parameters n and ρ, and therefore contains 200 results. The same goes for Porter's assessment, and that of the raw data. We propose to vary the hyper-parameters of our method in the following way : (i) ρ is between 0 and 1, we propose a step of 0.1, (ii) n is integer, which we propose to vary from 1 to 9. We thus obtain 81 different configura-

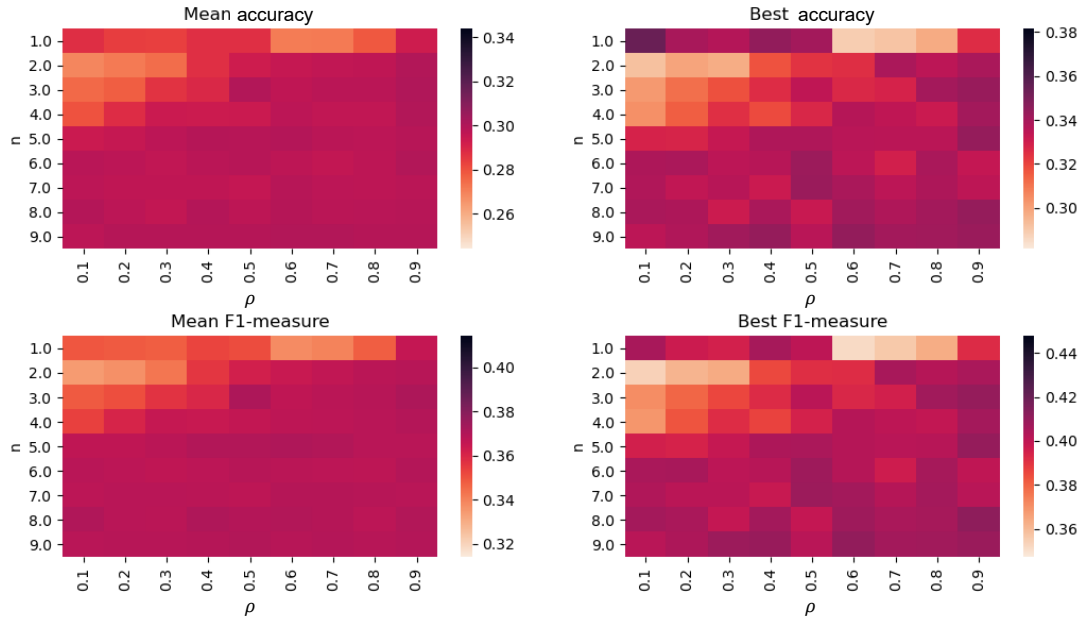


Figure 5.1 – Study of the accuracy and F1-measure for on the multinomial classifier (**Task 1**) in English (AustLII) with the heatmaps of the 4 indicators for the 81 configurations of RFreeStem.

Method	Parameters			#stems	#remaining terms	Accuracy	F1-measure	ICF
	n	ρ	f_{min}					
RFreeStem	1	0.1	1	2748	2468	**0.354**	0.406	**0.835**
	8	0.9	1	16553	16020	0.344	**0.412**	0.003
Porter			1	12000	11374	0.324	0.391	0.278
No stemming			1	16609	15795	0.339	0.406	0

Table 5.3 – Best results obtained for the multinomial classifier (**Task 1**) in English (AustLII) with the best configurations of RFreeStem. #stems : the number of stems obtained after stemming. #remaining terms : number of terms that have a frequency superior to f_{min}, after stemming and removing stop words.

tions. We study the values of the 4 previously mentioned indicators regarding these configurations, and present them in the form of a heatmap. We also study the versions that provide the highest results for the F1-measure and the accuracy : they are presented as a couple (n, ρ), associated to their best f_{min} value. We compare these versions to Porter's stemmer and those of the version without stemming (also associated to their best f_{min} value).

5.4.2 Results and discussion

5.4.2.1 Comparison of the two tasks on the English language

We first compare the results obtained for the two tasks (**Task 1** and **Task 2**) for English. The data in **Task 1** present many classes, some of which contain very few examples. It is therefore particularly difficult to train the classifier to recognize these classes, which explains the weak results observed in Table 5.3. The heatmaps (Figure 5.1) show a smooth behavior of our stemmer with respect to the evolution of these parameters : the map is globally homogeneous and the color

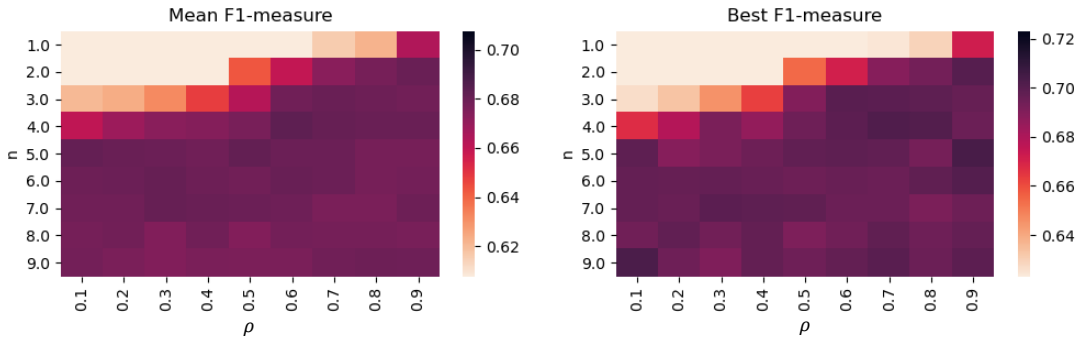


Figure 5.2 – Study of the F1-measure (the accuracy is almost identical) for the binary classifier (**Task 2**) in English (movie reviews) with the heatmaps of the 2 F1-measure indicators for the 81 configurations of RFreeStem.

Method	Parameters			#stems	#remaining terms	Accuracy	F1-measure	ICF
	n	ρ	f_{min}					
RFreeStem	5	0.9	2	7704	3042	**0.704**	**0.704**	0.004
Porter			1	6757	6666	0.703	0.703	**0.127**
No stemming			2	7736	3050	0.699	0.699	0

Table 5.4 – Best results obtained for the binary classifier (**Task 2**) in English (movie reviews) with the best configurations of RFreeStem. #stems : the number of stems obtained after stemming. #remaining terms : number of terms that have a frequency superior to f_{min}, after stemming and removing stop words.

transitions are smooth. This indicates that RFreeStem is not dependant on the optimization of its parameters. Many of our versions improve the version without stemming, while Porter's method shows disappointing results. We note two particularly interesting configurations : *($n = 1$, $\rho = 0.1$)* offers a strong stemming and achieves the best accuracy (0.354) while tumbling down the number of terms needed, with an ICF of 83.5%, while (n = 1, $\rho = 0.4$) offers a much lighter stemming, with the best F1-measure (0.412).

The **Task 2**, illustrated by the movie reviews dataset, seems much easier. There are only two classes, which are both fairly represented. Hence the F1-measure and the accuracy become almost identical : we only detail the F1-measure. The results presented in Table 5.4 are much higher than those of the previous study. The heatmaps (Figure 5.2) highlight demarcated areas : the upper left area corresponds to strong stemmers, which group together many words with few stems. It seems that these solutions are not beneficial for this dataset, with F1-measures that do not exceed 0.42. We explain this by the informal aspect of the comments : modern English is generally not very inflectional, and in the case of humanly-written reviews, is also informal. We would therefore tend to say *not happy* rather than *unhappy*. Conversely, the AustLII corpus contains legal citations, in a more formal discourse, with words of Latin ethymology, and therefore more inflectional. We conclude that informal corpus need more words to incorporate the original meaning of documents, and also have fewer inflectional variants likely to be removed in the stemming process.

These observations are confirmed by the small ICF of the best versions of the methods (Table 5.4). Nevertheless, most versions of our methods improve the F1-measure of the original version, and some achieve results equivalent to Porter's (around 0.70), however requiring fewer terms

Method	Parameters			#stems	#remaining terms	Accuracy	F1-measure	ICF
	n	ρ	f_{min}					
French								
RFreeStem	3	0.2	1	5984	5120	**0.811**	0.772	**0.562**
	7	0.3	3	13410	3679	0.763	**0.776**	0.018
Porter			1	10718	9432	0.802	0.763	0.215
			2		4687	0.775	0.773	
No stemming			1	13656	12068	0.789	0.748	0
			3		3653	0.761	0.774	
German								
RFreeStem	3	0.1	1	18757	16907	**0.797**	0.754	**0.395**
	8	0.6	4	30841	5978	0.781	**0.781**	0.005
Porter			2	24624	10335	0.790	0.769	0.205
			4		4369	0.771	0.773	
No stemming			1	30986	27843	0.778	0.751	0
			4		5879	0.776	0.775	

Table 5.5 – Best results obtained for the binary classifier (**Task 2**) in French and German (comments of Amazon products) with the best configurations of RFreeStem. #stems : the number of stems obtained after stemming. #remaining terms : number of terms that have a frequency superior to f_{min}, after stemming and removing stop words.

after post-processing (around 6 000 terms for Porter, 3 000 for RFreeStem). From this first study, we observe that **Task 1** is difficult, and that Porter's algorithm degrades the results. The data are flexional, since it results from a sustained language, and RFreeStem manages to significantly improve the initial results, in particular by proposing two versions which considerably reduce the size of the data. The **Task 2** is easier, and the original data already show satisfactory results. Light stemmers, like Porter's and some of our versions, are the most suitable because they keep many terms intact, and thus maintain the meaning of comments. Stemming therefore has less impact on this second example. We believe that beyond the task, this can be explained by the choice of the language. We expect stemming to have a greater effect on inflectional languages, and show this in the following study.

5.4.2.2 Comparison between the languages on Task 2

We now study the Amazon product reviews in French and German. These two languages are inflectional : each stem has many variations and so many words can be summarized together while respecting their meaning. However, as with movie reviews, these corpora contain informal human speeches (which is common in sentiment analysis). In informal speech, French, like English, tends to be less flexible. The results presented in Table 5.5 are however encouraging. For the French dataset, the configuration *(n = 3, ρ = 0.2)* succeeds in offering an important compression (56%) while improving the accuracy of Porter's stemmer (0.811 vs 0.802 for Porter), and satisfactory results for the F1-measure, equivalent to Porter's. Both this version and Porter's improve the accuracy of the raw words version (0.789). This confirms the importance of stemming for an inflectional language such as French. Conversely, the configuration *(n = 7, ρ = 0.3)* improves the F1-measure

Method	Parameters			#stems	#remaining terms	Accuracy	F1-measure	ICF
	n	ρ	f_{min}					
RFreeStem	3	0.7	2	7856	2681	**0.529**	**0.538**	**0.004**
No stemming			2	7886	2686	0.520	0.531	0

Table 5.6 – Best results obtained for the binary classifier (**Task 2**) in Urdu (Tweets) with the best configurations of RFreeStem. No Porter version is implemented for this language. #stems : the number of stems obtained after stemming. #remaining terms : number of terms that have a frequency superior to f_{min}, after stemming and removing stop words.

of Porter's version and the raw words version. This version associates a very light stemming (ICF of only 0.018, with 13 000 terms) to an aggressive term reduction : with $f_{min} = 3$, we remove all the stems that are present in less than 3 documents, and since the stemming is light, many terms are concerned. These two methods show the modulation power of our method, which is able to propose two opposite stemming strategies that both offer reliable results.

Similar results are observed with German reviews. RFreeStem extract two different configurations : a strong stemmer, with *($n = 3$, $\rho = 0.1$)*, that highly reduces the vocabulary size (ICF if 39.5%) and improves the accuracy, and a light stemmer, *($n = 8$, $\rho = 0.6$)*, associated to an aggressive stem, that improves the F1-measure of Porter's version. We generally believe that RFreeStem is more suitable for the German language than Porter's algorithm. Indeed, the process of Porter's stemming does not include any treatment for complex compound words, whereas Braschler and Ripplinger (2004) claims that, for the German language, the decomposition task of these complex parts is critical, even more than the discovery of the stem. In contrast, our algorithm is theoretically able to recover such linguistic complex structures, which may explain these encouraging results.

Finally, we study the behavior of our method on a language for which few automatic processing resources are available, like Urdu, an Indo-Aryan language, mainly spoken in Pakistan and India. For this language, no version of Porter has been implemented to our knowledge. We therefore only compare our results to those obtained with the raw data in Table 5.6. The balanced distribution of the data between the different classes results in a similar behavior between accuracy and F1-measure. Hence, the accuracy and the F1-measure share the same best configuration, *($n = 3$, $\rho = 0.7$)*, a very light stemmer (ICF=0.004) associated to a strong stem filtering (less than half of the terms are kept). This configuration reaches a F1-measure of 0.538 and an accuracy of 0.529 versus respectively 0.531 and 0.520 for the raw words. Our method is therefore able to globally improve the accuracy for this poorly-resourced language.

6 Part conclusion

In this first part, we presented a new structure inference method, composed of a log parser METI*N*G, Modular Event Type Inference with *N*-Grams, and its associated template extraction, published in (**?**). METI*N*G is a parametric log parser based on the frequent pattern mining assumption. Both the usage of parameters and the choice of a flexible syntactic assumption enable METI*N*G to adapt to the wide diversity of the reference datasets. METI*N*G creates a dendrogram of logs by successive splits of the logs based on the simple presence of frequent n-grams. This heuristic functioning enables our method to be computationally efficient. We enriched our log parser with a generic and comprehensive template extraction method, which is implemented as a post-processing step, and independently of the log parser.

Our evaluation study shows that METI*N*G is able to parse the reference datasets that were not properly treated by the existing log parses, thanks to both its flexible syntactic assumption, and its high modulation power, enhanced by its sensitivity to its two hyper-parameters. METI*N*G also shows the best results for the OBIS logs, even without specific preprocessing. We study the behaviour of the methods on specific conditions, and found out that METI*N*G is capable of dealing with a wide set of challenges (e.g. groups with logs of different lengths, with strong alternation between fix parts ...), which explains its impressive improvement on some datasets compared to the state-of-the-art references, Drain, Spell and IPLoM. METI*N*G is faster than the state-of-the-art references, and can be optimized on a small subset of data, due to the convergence of its hyper-parameters : we propose a hyper-parameter setting method based on semi-manual labelling. METI*N*G can also be applied in an online fashion, and is fast enough to parse the reference dataset with the highest generation rate. Finally, we also present an example of the output of the template extraction method, depicting the potential added value of the global flow of structure inference.

To emphasize the generalization power of our method, we propose an extension of METI*N*G to the study of stemming algorithms, in text mining. We present RFreeStem (**??**), a rule-free stemmer which can be applied on multiple languages, including poorly-endowed ones. Similarly, RFreeStem relies on the presence of frequent n-grams through a simple presence check, and is therefore more computationally efficient than the traditional clustering-based rule-free stemmers. We experimentally show that RFreeStem improves the performance of two text mining tasks — document classification and sentiment analysis — compared to the data without stemming. Moreover, RFreeStem often outperforms the state-of-the-art reference, Porter, while providing a stemming solution on poorly-endowed languages, which have no Porter version implemented.

To conclude, the output of the structure inference workflow is two-fold : (i) the log parsing

method outputs a labelling of the logs which enables to represent the logs by their event types, as per required by most of the log mining algorithms, (ii) the template extraction enables to retrieve knowledge on the parameters of the logs (the variable parts), which are used to create knowledge lexicons of the logs. These lexicons contain rich information on the nominal behaviour of the complex log systems. On the OBIS logs, the structure inference has a strong industrial added-value : a part of the logs does not have an event type, it is therefore impossible to apply anomaly detection on it. In the second part, we study how these newly extracted knowledge and representation can be used to detect anomalies in logs.

Part

Anomaly detection in a log dataset

1 Context and challenges

In this chapter, we exhibit the context and challenges of detecting anomalies within log datasets. From the inferred log structure, and especially, the event type representation of logs, log anomaly detection methods are used to alert on any abnormal behaviour on the system, through the analysis of logs. In section 1.1, we first describe the stakes and importance of detecting anomalies in systems. Section 1.2 presents the main existing work on automated log anomaly detection, and describes their limitations, justifying the need for a new solution. Finally, section 1.3 presents NoTIL, our contribution, and the organization of the part.

1.1 Context presentation

In the last decades, IT systems have scaled to many use cases (e.g. performance computing (Oliner and Stearley, 2007), distributed systems (Xu et al., 2010), online cloud-based services...), which has triggered the fast growing of their size and complexity (Lin et al., 2016; Du et al., 2017; Xia et al., 2020). Since these systems might provide services to millions of users (He et al., 2016b, 2018b), they need to be available on a 24x7 basis, generating important constraints on the service quality. However, they contain numerous sources of potential faults and vulnerabilities, including malicious attacks to the systems (e.g., network intrusion, account compromising...), operational bugs and unexpected behaviours (Lu et al., 2018b). The latter can be generated during the deployment in production of the systems, due to important gaps between the testing and production environment (Lin et al., 2016), or at any time during the in-service life of the system, due to external environment, or unexpected usage (He et al., 2016a; Borghesi et al., 2019; **?**).

The failure or loss of trustworthiness of the services and applications of IT systems can trigger the unavailability of the systems (He et al., 2016b), which can have significant consequences, ranging from degraded user's experience (He et al., 2018b; Meng et al., 2019) to important financial losses (Zhang et al., 2016; Liu et al., 2019a) : Farshchi et al. (2015) quotes several industrial examples of companies estimating the hourly cost of unexpected system downtime between $100K and $540k, while other firms estimate their average yearly loss to $100 million. More spectacularly, a single downtime of Amazon in 2017 costed the firm more the $150 million (UpGuard, 2019; He et al., 2020a). In this context, the ability to accurately detect anomalies is key to insure the efficiency of incident management (He et al., 2016b; Xia et al., 2020). As a result, the anomaly detection is considered as a major issue and has raised much research and industrial interest (Zhang et al., 2016; Du et al., 2017).

Log analysis constitutes a robust solution to system anomaly detection. Large-scale systems generate numerous execution logs, which are recognized to be universally available (Lin et al., 2016; Meng et al., 2019; He et al., 2020a). A *log anomaly* is an unexpected behaviour of the log data, and can be paired to a system anomaly. For instance, a late appearance of logs in a sequence may indicate a *performance anomaly* corresponding to an abnormal temporal irregularity in a service response (Fu et al., 2009a; Du et al., 2017). Generally speaking, logs include the monitoring of critical event and provide resourceful information on the system execution, which helps the diagnosis of system errors and failures (Lin et al., 2016; Xia et al., 2020). The Oracle website [1] acknowledges that (i) error logs are commonly used to directly detect well-know problems, and (ii) the logging of contextual information can help to trace the execution path and provides context for diagnosis.

Hence, log anomaly detection is recognized as an efficient mean to perform system anomaly detection (He et al., 2016b; Du et al., 2017; Borghesi et al., 2019). As a result, log anomaly detection has raised much research interest (Lou et al., 2010; Lu et al., 2018b; Meng et al., 2019; Liu et al., 2019a) and is regarded as a challenging and open question. First, the unstructured nature and heterogeneity of logs (discussed in Section 1.2.3) make their treatment difficult. Second, the high volumes of log data generated (as formerly depicted in Table 1.1) results in an important challenge to process and diagnose the system. It is especially difficult to understand the actual execution paths depicted by the logs while many parallel paths might be generated at the same time (Du et al., 2017; Xia et al., 2020).

The traditional and first historically used way to detect anomaly consists in manual look-up of the logs by the system engineers (Lin et al., 2016; He et al., 2016b). However, manually analyzing large and complex log datasets represents a cumbersome and error-prone task (He et al., 2016a, 2018b; Zhang et al., 2019), especially when the engineer only has a partial comprehension of the overall system (Meng et al., 2019) — some open source systems are implemented by hundreds of developers (He et al., 2016b).

As an alternative, engineers might explore the log database through the usage of keyword searching, based on their domain knowledge. These keywords (e.g. “fail”, “down”, “error” ...) can directly detect some specific failing pattern (Lin et al., 2016; Meng et al., 2019). However, the complexity of large-scale systems make the keyword anomaly detection inaccurate. This method tends to generate numerous false positives, since logs containing these keywords include “failover” mechanisms, i.e. situations that the system is able to recover on its own, which do not generate any abnormal behaviour of the system (He et al., 2016b; Lu et al., 2018b). Similarly, some failures might impact several components, leading to multiple reporting (several error logs) of the same anomaly (Lin et al., 2016). Keyword search also tends to generate false negatives, since the keywords are not able to detect anomalies with complex signatures among logs (Meng et al., 2019). Regular expressions can also be implemented to match and detect the patterns of anomalies. Yet, due to the fast-changing character and large-scale of log data, the maintenance of the list of such regex is unfeasible.

In conclusion, system anomaly detection should be based on the automated anomaly detection of logs (He et al., 2016b; Meng et al., 2019). The following section present the existing propositions and their limitations.

1. https://docs.oracle.com/javase/10/docs/api/java/util/logging/package-summary.html

1.2 Limitations of existing log anomaly detection

Supervised methods are the first historically used to detect anomalies in logs (Chen et al., 2004; Liang et al., 2007). They train binary classifiers, based on both normal and abnormal samples. Yet, they suffer from the scarcity of abnormal samples, which prevents them to properly learn to define the abnormal class. In addition, they require fully-labeled datasets, which is often unrealistic for production logs (He et al., 2018b; Xia et al., 2020). Finally, they can only detect the class that has been learned, and cannot detect unknown anomalies (Du et al., 2017), which are however the most valuable information for end-users. Unsupervised anomaly detection (Xu et al., 2009a; Lou et al., 2010; Lin et al., 2016) relies on data mining analysis and representationsof data. They do not include a training phase, and hence require neither labelled data, nor balanced data (He et al., 2016b; Xia et al., 2020). Yet, unsupervised anomaly detection is reported to render mitigated accurate results (He et al., 2016b). Indeed, these methods tend to only detect outliers, points that are rare and significantly different from the majority, regarding selected features (He et al., 2016a). He et al. (2018b) however concludes that outliers are not necessarily anomalies to detect : they can come from normal, yet rare user behaviours, or rarely-used system features.

Novelty detection methods offer an interesting alternative to traditional supervised and unsupervised learning (Fu et al., 2009b,a; Du et al., 2017; Meng et al., 2019). A nominal behaviour model is learnt on anomaly-free data. New coming data are compared to this model and detected as anomalies if their behaviour does not match the nominal one. These methods represent a compromise between unsupervised and supervised learning : (i) they include a training phase, which improves their prediction accuracy (ii) they do not require samples of the abnormal class, since the training is performed on the normal one only (iii) they detect novelties, i.e. points that differ from the training samples, but are not necessarily rare. For these reasons, novelty detection is now the most common approach to detect anomalies in log systems.

Among them, deep learning methods are often preferred to traditional machine learning methods, since they can efficiently learn complex representations within the different features. Especially, Long-Short Term Memory (LSTM) neural network are popularly used to catch sequential dependencies (Du et al., 2017). For instance, DeepLog trains a prediction task with an LSTM network, which efficiently represents complex sequential dependencies based on multiple previous logs. DeepLog is regarded as one of the most up-to-date and accurate reference for the detection of sequential anomalies. Many later works have been inspired from this method and propose different data representations (Meng et al., 2019) or different network architectures (Lu et al., 2018b; Liu et al., 2019a).

Besides, the log partition consists in determining how to create samples out of the flow of log data (He et al., 2020a). Most of the novelty detection approaches (Du et al., 2017; Meng et al., 2019; Zhang et al., 2019) generate samples thanks to the sequence identifier (e.g. user, session or task ID...). However, such identifier is seldom available. Alternatively, temporal partitions build samples based on the timestamps of the logs, creating — fix or sliding — windows of times. This process does not require the definition of a sequence identifier, and also enables to introduce a temporal component to the analysis. While this practice is common for supervised and unsupervised learning (He et al., 2016b), to the best of our knowledge, none of the existing novelty detection methods adopts this log partition.

In most of the modern novelty detection methods based on deep learning, logs are represented as temporally-ordered sequences of event types (Du et al., 2017; Lu et al., 2018b; Liu et al., 2019a). While this representation is sufficient to model sequential anomalies (e.g. log missing, abnormal path, unexpected log), it is however insufficient to catch quantitative anomalies (e.g. abnormal concentration of logs) and temporal anomalies. LogAnomaly (Meng et al., 2019) adds an attention mechanism to extend the sequential anomaly detection, and improves the accuracy of DeepLog on several reference datasets. Nonetheless, both these representations discard the time-frame separating logs. This information is however key to define performance anomalies. The *performance anomaly detection* field focuses on detecting periods of slowdown or unavailability of a system or a service (Bonnevay et al., 2019), and ranges from the detection of deny-of-service attacks (Du et al., 2017) to the performance monitoring of cloud infrastructures (Tan et al., 2012). For these anomalies, the time elapsed between logs must be integrated to the model in order to detect temporal irregularities in logs' appearance. The only available state-of-the-art proposition that takes time into account might be the extension of DeepLog (Du et al., 2017), which studies the time elapsed between consecutive logs of the same event types. This univariate representation is however incapable of modelling complex relationships between event types (**?**).

We conclude that the state-of-the-art lacks an accurate novelty detection approach, based on sophisticated deep learning models, that could treat both sequential and temporal anomalies and models complex and temporal relationships between event types. This could be achieved thanks to a data representation that takes the time elapsed between logs into account.

1.3 Contribution and part organization

To answer the challenges of detecting anomalies in logs, our contribution, in this part, consists in a novelty detection method named NoTIL, **No**velty detection based on **T**emporal **I**rregularities in **L**ogs. Based on the idea of counting the event types over time with a sliding window, NoTIL takes the time elapsed between logs into account by capturing the frequency of logs' appearance. NoTIL takes advantage of the LSTM ability to model complex temporal dependencies and applies it to model temporal correlations and detect temporal irregularities. With this design NoTIL (i) is a novelty detection approach and hence, (a) does not need any label, (b) does not need abnormal samples for training, (c) benefits from the accuracy of a training phase on the nominal class, (ii) adopts a data representation that takes time into account, which enables NoTIL to catch both sequential and temporal anomalies in logs (contrary to DeepLog and LogAnomaly).

In the remaining of this part, we first census, in Chapter 2 the different proposition for anomaly detection, both in traditional machine learning and deep learning, with supervised, unsupervised and novelty-detection-based learning. We also present the features selected in different methods, as well as the log partition chosen. In Chapter 3, we present our novelty detection methods, NoTIL. We provide details on the feature selection, the log partition and the prediction task used to differentiate normal and abnormal samples. We propose and present different neural network architecture for this prediction task. Finally, Chapter 4 assesses our anomaly detection methods, and compares it to some state-of-the-art propositions. We perform a comprehensive evaluation by performing this comparison on both simulated data and some real-world reference datasets.

2 Related work

In this chapter, we present the main state-of-the-art methods for the anomaly detection task on logs. In section 2.1, we first present these methods according to their supervision types (supervised, unsupervised, based on novelty detection), since this represents an important conceptual distinction. We then focus on the data generation, by presenting the different methods to partition the logs into samples, and to select relevant features. Finally, we review the common practices for the evaluation of log anomaly detection, and present the main conclusions of the state-of-the-art evaluation on existing anomaly detectors.

2.1 Types of supervision

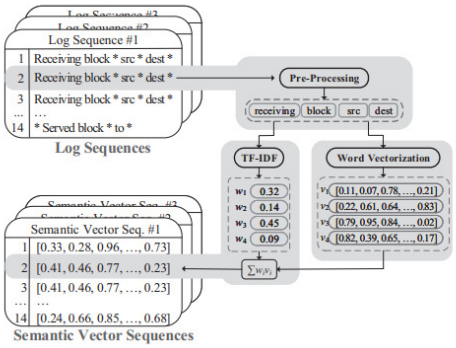
We propose a first level of classification based on the type of supervision, namely supervised, unsupervised, and novelty-based. While supervised methods aim at learning both the normal and abnormal class, unsupervised methods mine the dataset based on the idea that abnormal samples are rare, and far from the majority of nominal points. Novelty detection methods represent an alternative, by detecting anomalies as deviations from a nominal behaviour, trained on only normal samples. This nominal behaviour is often learned through an intermediate prediction task. This section censuses the main works in the three categories and highlights some general limitations.

2.1.1 Supervised methods

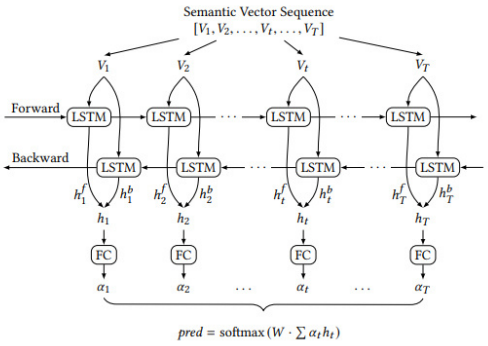
Supervised methods can be described as binary classifiers : they train a model to distinguish the normal and abnormal classes, and require samples of each class. They also heavily rely on the selected features to contain an obvious segregation between the two classes.

2.1.1.1 Traditional machine learning

Chen et al. (2004) propose to train a *decision tree* to learn the decision of detecting a sample as abnormal. Decision trees are tree structures that represent the decision making of a classification task, based on intermediary splitting considerations. Each level of the tree represents a condition that is learned during training process to enable the final classification decision. Similarly, Support Vector Machines (SVM) are commonly employed in the binary classification case for log anomaly detection. SVM algorithms learn a hyper-plane that separates the nominal and abnormal samples according to the selected features, with the goal of maximizing the distance of each points to the



(a) The generation of semantic vectors from logs, based on word vectorization and TF-IDF. Each log is represented by a numerical vector. The vectorization is learned during the training.



(b) The bi-directional LSTM used for binary classification. LSTM can model sequential relations. The bi-direction enables to model both forward and backward relations. The fully connected (FC) layer learns attention weights for each event type.

Figure 2.1 – Two main features describing the architecture of LogRobust (Zhang et al., 2019).

learned frontier. Liang et al. (2007) propose to extract several macro-indicators of the logging system (e.g. number of events in a time window, accumulated number of events ...) and feed this representation to the SVM. Kimura et al. (2018) propose a proactive detection of failures based on an SVM classifier with Gaussian kernels. In this methods, several other indicators are studied, like the periodicity, burstiness, or correlation with failures. Finally, He et al. (2016b) conduct an experimental study in which they compare different supervised log anomaly detectors. They implemented an adaptation of the Logistic Regression algorithm (Hosmer Jr et al., 2013) in order to classify normal and abnormal samples. The logistic regression is a statistical model that calculates the probability for a sample to be abnormal. The function expression is optimized during the training phase.

Generally speaking, the anomaly detection models are recognized to involve complex relationships between the event types. Yet, traditional machine learning models tend to be insufficient to model such complex correlations, hence why deep learning methods have recently raised important research interest (Du et al., 2017).

2.1.1.2 Deep learning

More recent research papers have studied the opportunity to use deep learning models to train the binary classifier. LogRobust (Zhang et al., 2019) implements an Attention-based bi-directional LSTM, as presented in Figure 2.1. The main effort of the method consists in proposing a semantic vectorization that is robust to changes in logs (Figure 2.1a). The vectorized logs are then passed to a bi-directional LSTM, which classifies the logs into either normal or abnormal samples (Figure 2.1b). LSTM are designed to model sequential dependencies. Bi-directional LSTM can model relations forward and backward in time. The network includes an attention mechanism : the final fully connected layers (FC in Figure 2.1b) learn attention weights for each event type. The model can manage the impact of each event type on the final prediction, and especially moderate the weight of irrelevant event types.

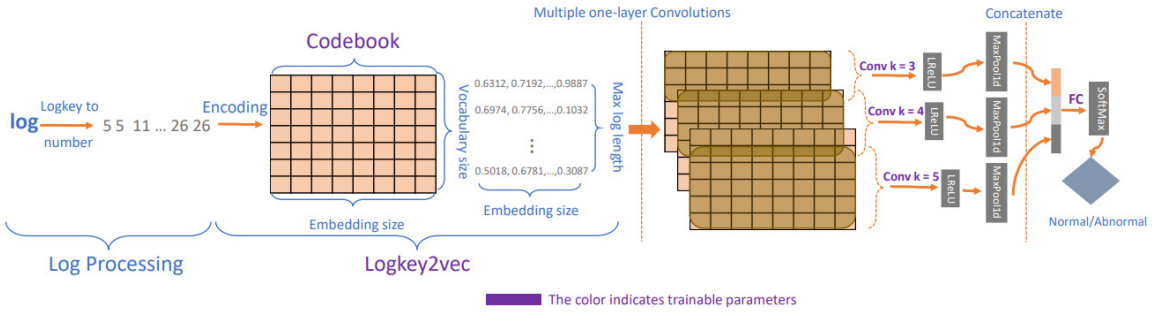


Figure 2.2 – The CNN architecture presented in Lu et al. (2018b) with an embedding phase called Logkey2vec, and multiple 1D convolutions, that are concatenated and output as binary classifier. The embedding projects the event type to a higher dimension, and the convolutions extract complex hidden features from this high dimensional representation.

Lu et al. (2018b) use the popular Convolutional Neural Networks (CNN) to build a binary classifier for anomaly detection predictions, presented in Figure 2.2. This technique includes an embedding phase, where event types are converted to vectors with a deep learning encoding layer : this method projects the sequence of event types onto a higher dimensional space, enabling the modelling of complex relations between event types. These vectors are passed as input of multiple one-layer convolutions, which are then concatenated and fed into a fully connected layer to retrieve the final binary prediction. The convolutions are reputed to efficiently tackle high dimensional data (especially graphical data (Chalapathy and Chawla, 2019)). Since the event types are projected on a higher dimension, CNN can efficiently extract complex hidden features in this high dimensional representation.

Finally, LogGAN (Xia et al., 2020) trains a binary classifier based on adversarial learning, presented in Figure 2.3. LogGAN includes a Generative Adversarial Network (GAN, (Goodfellow et al., 2014)), a deep learning model divided in two components : (i) the generator, that creates candidate samples based on real samples and (ii) the discriminator, that learns to separate real and fake samples. The learning of the two components is set to be adversarial, since the loss function of the generator encourages it to generate samples as realistic as possible, while the discriminator favours a clear distinction between real and fake data. The two components are implemented with LSTM models, and trained with both normal and abnormal data. As such, when the model is trained, the generator component is readily able to provide accurate predictions for normal and abnormal samples. The mechanism of generating data is presented as a solution to the imbalance between normal and abnormal sample.

2.1.2 Unsupervised methods

To cope with the strong reliance on labelled data, unsupervised methods were intensively studied in the last decades (Xia et al., 2020). There exist different types of approaches, yet, all are based on the assumption that abnormal samples are outliers : they are rare, and far from the majority of samples (which represents the nominal samples) according to the selected features. In this subsection, we census different types of unsupervised anomaly detection approaches.

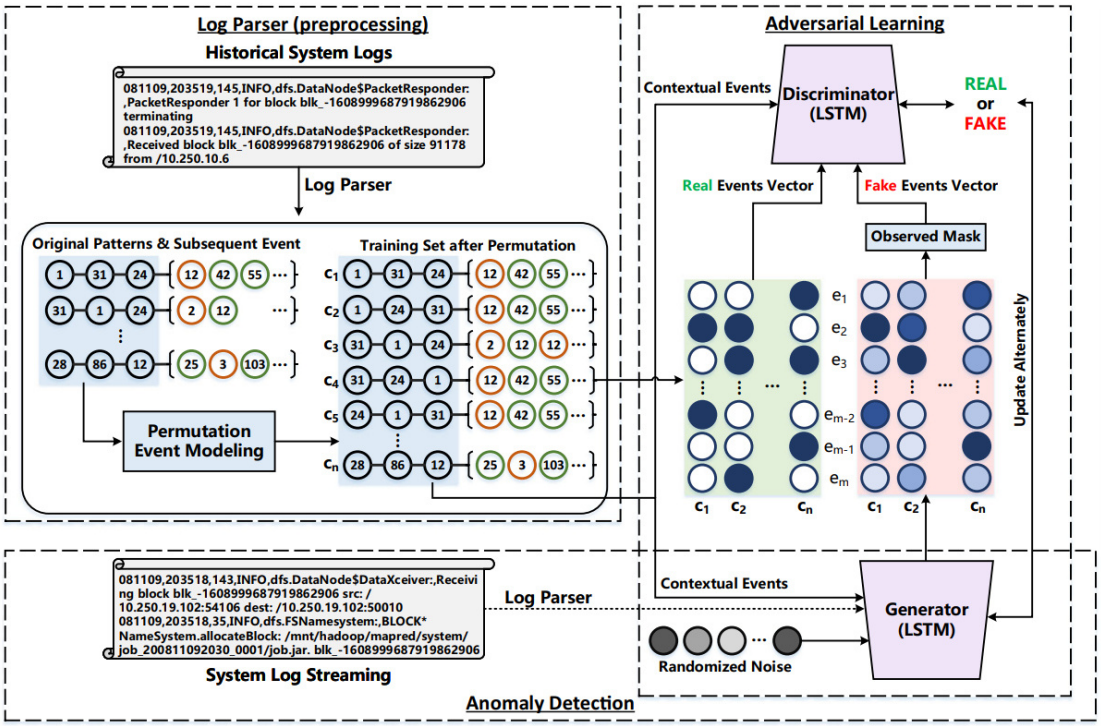


Figure 2.3 – The global process of LogGAN (Xia et al., 2020), including (on the right side) the adversarial learning architecture. The architecture contains a generator LSTM, which generates realistic data, and a discriminator LSTM, which detects synthetic data.

2.1.2.1 Dimensionality reduction

Some unsupervised methods focus on reducing the dimensionality of the selected features so as to capture the most important characteristics of the data. The most common technique is undeniably the Principal Component Analysis (PCA). PCA consists in projecting the studied points on meaningful axes (the principal components). This projection operates a dimensionality reduction that maximizes the maintained variance between points, while enabling more robust distance measurements (see the *dimensionality curse* (Aggarwal et al., 2001)). Distance calculations can be applied to detect the most distant points, that are alerted as anomalies. Xu et al. (2009a) and Xu et al. (2010) apply this method to detect outliers among the event counts of logs — number of event types per temporal windows.

Similarly, Xu et al. (2009b) adapt the PCA approach to the online detection of abnormal event traces (i.e. abnormal sequences of events). The event traces are first passed to a frequent pattern miner, which builds a pattern dictionary of the most frequent event patterns. To improve the computational efficiency and the temporal reactivity of the method, only the traces that do not correspond to these frequent patterns are passed to the PCA detection. The authors acknowledge that PCA is more accurate for anomaly detection than frequent pattern mining, which confirms that this first additional phase shall be interpreted as a filter of traces, since the most frequent traces are surely not abnormal, under the assumption of outlier detection.

2.1.2.2 Clustering

As another classical unsupervised technique to detect anomalies, clustering algorithms group logs into clusters. The idea is that nominal samples form the majority of logs and should be gathered in a main cluster, while abnormal samples are outliers and should be clustered outside the nominal cluster. Lin et al. (2016) propose a clustering method where the log sequences are represented by event-count vectors and grouped thanks to an agglomerative hierarchical clustering technique (Johnson, 1967), based on a similarity definition between log sequences. The cluster creation is divided in two steps : first a 2-cluster partition is created based on logs from the testing environment (in order to mimic the normal and abnormal classes), then logs from the production (logs of interest) are successively passed to the clustered representation. If they are clustered to the normal group, they are diagnosed as normal. If they are clustered to the abnormal group, or if they do not match any of the clusters — distance superior to a threshold, in this case, a new abnormal group is created — they are diagnosed as abnormal. It appears that this method can detect and clearly identify new types of anomalies. However, He et al. (2016b) report that this method renders inaccurate results on the 2 reference datasets evaluated (HDFS and BGL), while presenting poorly efficient algorithmic complexity — hierarchical clustering techniques are evaluated to have a complexity of $\mathcal{O}(n^3)$.

Alternatively, He et al. (2018b) propose Log3C, a full anomaly detection framework that integrates a new clustering approach called Cascading Clustering. Cascading Clustering leverages the computational efficiency of the agglomerative hierarchical clustering by incrementally clustering small temporal fragments of the full datasets. At each step, clusters are created, from which templates are extracted. The rest of the logs (unclustered) are compared to the templates for matching, and are either associated to the created clusters or left for the next clustering iteration. The authors report a more accurate and computationally efficient behaviour than PCA (section 2.1.2.1) and Invariant Mining (section 2.1.2.3). However, the anomaly detection also relies on the search of correlations between logs and other system performance indicators. These indicators are not necessarily available along with log datasets.

Finally, Log2vec incorporates the word2vec embedding (Goldberg and Levy, 2014), which offers an efficient vector representation of words in the context of natural language processing. As presented in Figure 2.4, Log2vec successively builds (i) a rule-based graph, constructed through the mining of relationships between event types (e.g. causal or sequential relationships within a day, logical relationships among objects...), (ii) a word embedding, based on word2vec, learned from the log sequences and applied to the previously obtained graph to obtain a graph embedding, (iii) a clustering representation of the sequences, which is used as an anomaly detector. The clusters that have a number of elements lower than a threshold are labelled as abnormal. Despite the interesting inclusion of the powerful word2vec embedding, the graph construction phase relies on an important set of predefined rules, that are not necessarily adapted to the domain context of logs. Especially, most of these rules rely on the definition of a host, which is not defined in most of the reference datasets we study.

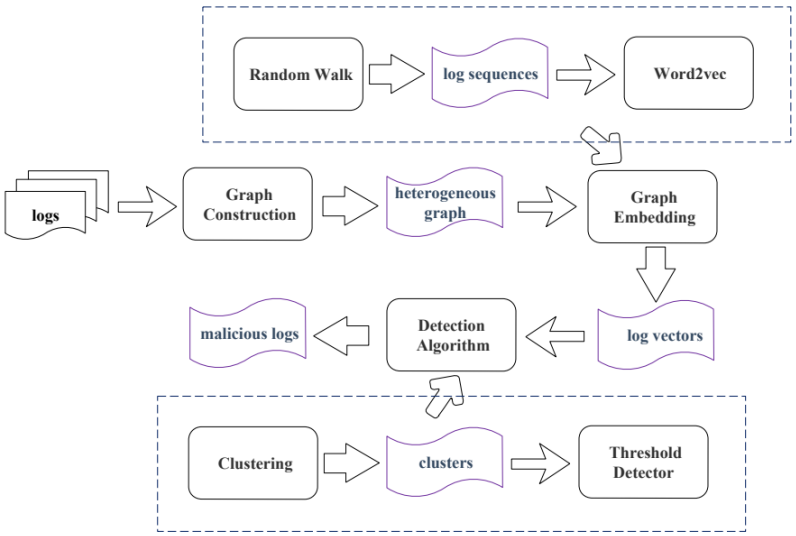


Figure 2.4 – The general framework of anomaly detection in word2vec (Goldberg and Levy, 2014), with the graph embedding phase (up) and the anomaly detection phase (down).

2.1.2.3 Frequent pattern mining

In the outlier detection field, frequent pattern mining approaches aim at finding statistical characteristics that are present in the majority of logs, and to detect anomalies as violations of these high-support features. Lou et al. (2010) first propose to mine invariant relations between event types into event sequences (IM). For instance, if A, B, C are three event types, and $n(A), n(B), n(C)$ are the number of occurrences of each event type in the sequences, the relation $n(A) = n(B) + n(C)$ is an invariant if it is verified in the majority of the sequences. This majority is defined with a support threshold, and all the sequences that do not verify the invariants are regarded as abnormal.

Similarly, Farshchi et al. (2015) apply the frequent pattern mining approach to detect anomalies in cloud applications. The method mines statistic correlations between the event counts and some system metrics extracted from the cloud environment — of course, the definition and availability of such metrics is highly arguable for other applications, such as the reference datasets. A multiple regression technique (Ordinary Least Squares) is applied to retrieve linear regressions between the events and the understudied metrics. These regressions are regarded as assertions. Hence, the regression error of a sequence is described as the deviation from the assertions, and used to detect anomalies.

Finally, Yamanishi and Maruyama (2005) build a finite mixture of Hidden Markov Model (HMM) as a statistical representation of the majority of supposedly normal logs. The HMM is built to represent the relations between the event types, and is dynamically updates, in order to adapt to online anomaly detection. Each log sequence is provided with an anomaly score which is directly based on the matching of the sequence to the model, thanks to traditional statistical tests.

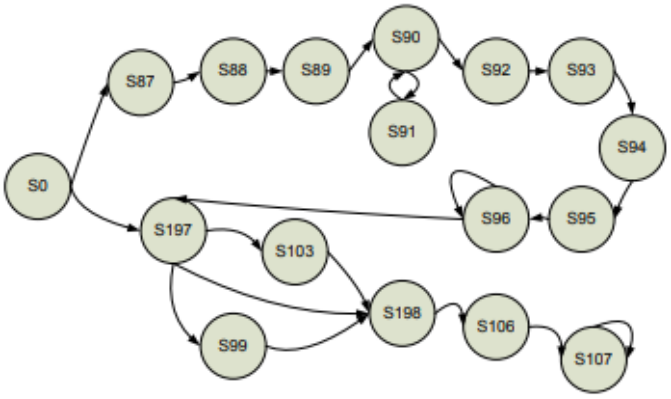


Figure 2.5 – An example of Finite State Machine, as in Fu et al. (2009b). Here, the nodes represent abstract states, mined from the sequences of logs.

2.1.3 Novelty detection based methods

Novelty detection methods can be seen as a middle ground between supervised and unsupervised approaches. These methods rely on learning a model on only normal samples. The trained model therefore represents the nominal behaviour of the data. New arriving logs are then tested. If they behave similarly to the learned behaviour, they are regarded as normal, otherwise they are alerted as anomalies. It could be argued that the training phase require labels to distinguish abnormal and normal samples, or at least the assurance that the training set is anomaly-free. Nevertheless, the novelty detection approach is not simply a one-class classifier. This concept rather relies on the idea of detecting novelties, or deviations from a trained model, learned on a reference dataset. Hence, instead of defining a unique abnormal class to learn on, like supervised methods, or defining anomalies as outliers, like unsupervised methods, novelty detection approaches regard the anomalies as deviations from a reference datasets. This subsection details the different approaches of novelty detection in traditional machine learning and data mining, as well as in deep learning.

2.1.3.1 Data mining and machine learning approaches

The most common data mining technique for novelty detection is the workflow mining approach. These methods construct a representation of the sequences of logs based on the order of appearance of event types in the sequences. Fu et al. (2009b) first present a Finite State Machine (FSM) on the normal sequences of logs, presented in Figure 2.5. Anomalies are detected as sequences which execution cannot be traced as a path inside the constructed FSM model. The article extended the workflow model to learn the nominal time elapsed during transitions. This enables the detection of performance anomalies (i.e. late arrival of logs in a sequence). Similarly, Lu et al. (2018a) build a FSM of the starting and ending of tasks in cloud environment to detect concurrency anomalies.

As an alternative to the FSM, Nandi et al. (2016) build a Control Flow Graph (CFG). While the states and transitions of the finite state model represent abstract concepts (hidden states), the nodes of the CFG represent the event types, while the edges represent the transitions from one

event type to another. The CFG is built through a two-stage process and evaluates the probability of each transition (learned from the nominal samples). The authors propose to detect two types of anomalies : (i) the sequential anomalies, defined as violations of the execution workflow and (ii) distribution anomalies, defined as violation of the probabilities of transitions.

Lastly, Shang et al. (2013) propose a different approach than workflow mining, while respecting the novelty detection assumption. The method compares the logs from the test environment, which are regarded as a reference, and study the deltas to this model in production logs. This method aggregates sequences with the set of unique event types, ignoring repetitions. This too simplistic representation is accountable for the difficulties of the method to efficiently detect complex sequential dependency, and to accurately detect anomalies.

Moreover, despite its interesting interpretability, workflow mining is recognized to lack of representation power (Du et al., 2017). Indeed, the sequential relations represented are limited to a few different cases, and do not take into account the concurrency issues of logs (i.e. several tasks executing in parallel). For this reason, more sophisticated methods are preferred through the study of deep learning techniques.

2.1.3.2 Deep learning approaches

The state-of-the-art reference in log novelty-detection-based deep learning is undeniably DeepLog (Du et al., 2017). The authors detect anomalies as deviations from the nominal behaviour by training a prediction task on normal data, and evaluating the prediction results for new arriving logs. If these new logs manage to render accurate prediction results, they are considered as similar to the data the predictor was trained on, and labelled as normal data. Otherwise, they are considered as abnormal. The proposed prediction task consists in predicting the next event based on a sequence of several logs. This task is trained with an LSTM model. DeepLog has enhanced the capacities of log anomaly detection by demonstrating much more accurate results than the state-of-the-art unsupervised and supervised techniques, and is now widely used as a reference or baseline (Meng et al., 2019; Xia et al., 2020).

More recently, articles have focused on extended the proposition of DeepLog by adding attention mechanisms. Attention mechanisms consist in incorporating weights for the input dimension in order to learn which parts of the input data are relevant for the prediction task. These methods offer two advantages : (i) they enable the incorporation of external metrics or other features selection, in order to improve the accuracy of the prediction, (ii) the trained weights can be analyzed and offer a intelligible explanation of the neural network prediction. The authors of Brown et al. (2018) propose and compare several attention mechanisms based on the traditional key-query-value representation. Figure 2.6 presents the example of the dot product attention.

Alternatively, Meng et al. (2019) propose LogAnomaly, a method that takes into account the quantitative aspect of log sequences, and leverages the sequential LSTM by an attention vector that counts the number of event types occurring in the understudied sequences. The authors aim at detecting both sequential and quantitative anomalies in logs.

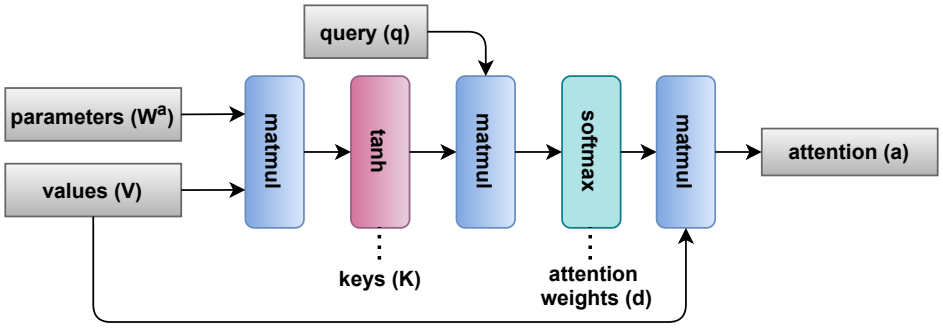


Figure 2.6 – An example of attention mechanism presented in Brown et al. (2018). This example is called the dot product attention. Attention mechanisms are generally described according to a (key-query-value) triplet input.

2.2 Log data representation

Besides the type of supervision used, a critical characteristic of a method consists in choosing a meaningful representation of the logs. The challenge of selecting a log data representation is divided in two questions : (i) How to partition the logs to create samples? (ii) Which feature(s) to select? We present the different propositions to answer to these questions in this section and exhibit their advantages and drawbacks.

2.2.1 Log partition

The log partition task consists in extracting samples to pass to the anomaly detector from the parsed log data. As so, the input of a log partition is the sequence of event types, along with their associated timestamp. Two main log partitioners exist (He et al., 2020a) : the timestamp of the log, or a sequence identifier. We describe here the two possibilities.

2.2.1.1 Sequence identifier based log partition

A vast majority of the aforementioned methods rely on a sequence identifier to partition the log datasets (Du et al., 2017; Lu et al., 2018b; Meng et al., 2019). As presented in Section 1.2.3, some log datasets define a sequence identifier that groups logs corresponding to a same task, user, session... This grouping offers the advantage of correctly segregating the logs into meaningful entities. Yet, this indicator is seldom available (or even defined) in the reference log datasets. Even when it is, it only concerns a part of the log datasets (only 40% of OpenStack logs (**?**)). Hence, relying on this sequence identifier is sometimes not feasible, or compromises the use of the whole dataset, removing interesting contextual logs. Similarly, this log partition prevents from detecting global anomalies (as opposed to individual ones). For instance, it would not be possible to detect a DNS attack, where multiple users encumber simultaneously and maliciously a network to saturate it.

2.2.1.2 Temporal log partition

Alternatively, the samples can be created by scanning the time frame and creating temporal windows (Xu et al., 2009a; Lou et al., 2010; He et al., 2016b). Two approaches exist : fix and sliding windows. In the first case, the time frame is simply cut into consecutive windows of the same length w. This process can be biased since it leads to arbitrary cuts in the time frame. Sliding window are shifted from a number $\delta < w$, creating overlapping windows of the same size (He et al., 2016a). Since the timestamp information is systematically available, these methods are always applicable. It is worth noticing that even the methods that recommend a partition based on the sequence identifier, such as DeepLog, advice to use the temporal partitions when the log databases do not define any sequence identifier (such as BGL). Moreover, this log partition naturally integrates the temporal component to the study : transition times between the logs are necessarily taken into account, which is generally an important feature, especially for the detection of temporal anomalies.

2.2.2 Extracted features

In parallel, the selection of the features to extract plays a crucial role in the representation power of the algorithm. From the partitioned dataset of logs, the feature extraction consists in choosing the information to represent from the logs, that will be passed to the anomaly detection algorithm. This representation should at least keep the discriminative features from the original data that explain the segregation between normal and abnormal samples.

The most common way to represent a log sample is by studying the temporally-ordered sequence of event types (Xu et al., 2009b; Nandi et al., 2016; Du et al., 2017). In this case, the time is only used to order the logs in the sequence, yet, the value of the time separating consecutive logs is ignored. This representation is unable to detect temporal anomalies like the performance anomaly, or a distribution anomaly, like the aforementioned DNS attack. To cope with this lack of representation power, LogAnomaly also study, for each sample, the event counts of the logs, and uses it as attention weights for the sequential prediction task. This method is therefore able to detect abnormal concentrations of event types in sample. Nevertheless, if it uses the sequence identifier log partition, the time elapsed between log is discarded. Hence, it cannot detect purely temporal anomalies.

DeepLog also proposes an independent extension of its anomaly detection, by studying the parameters of logs. In this case, each event type is processed independently, and a vector is built, for a given event type, by concatenating (i) all of its parameters values, extracted from the log parsing, (ii) the time elapsed since the last log of this event type. Firstly, this method offers an interesting integration of the time elapsed between logs. However, since event types are processed independently, — because they do not have the same number of parameters, hence, not the same dimension, — no temporal correlation can be learned between the different event types, which greatly limits the potential of detectable anomalies. Second, the integration of the parameters of logs offers the possibility of detecting a much greater set of abnormal behaviours. Yet, for the same reason, the model cannot learn dependencies between (i) the log parameter values and the sequence of event types, (ii) the log parameter values of different event types.

Another common method is to directly use the event counts of the sequence of logs (Xu et al.,

2009a; Lou et al., 2010; He et al., 2016b). When associated to the temporal partition, this representation offers the possibility of studying the temporal concentration of event types. It can therefore detect temporal and distribution anomalies (e.g. performance anomalies, DNS attacks). Yet, depending on the size of the window, this aggregation might remove the original order of relevant sequences. Therefore, the window size should be carefully chosen so as to preserve the features of interest.

In our description of the existing methods, we noticed many examples of features selection that were based on external data (cloud metrics, network KPI...). While improving the specific use cases they study, these methods present a poor generalization power, since such external values are not applicable to all the log datasets we study, and are highly domain-dependent. Instead, we focus on more realistically retrievable features.

2.3 Assessment of anomaly detection

In this section, we review the main studies and conclusions of the assessment of log anomaly detection. We rely on both (i) the evaluation parts of the articles presenting the methods, where new propositions are compared to the state-of-the-art references as a validation, and (ii) the reference experimental review of He et al. (2016b), which presents some interesting conclusions from its evaluations of several supervised and unsupervised methods.

2.3.1 Anomalies to detect

The methods focus on different types of anomalies to detect, which shape the decisions of the methods' implementations. The most widely studied anomaly in logs is the sequential anomaly (Du et al., 2017; Brown et al., 2018; Lu et al., 2018b; Meng et al., 2019). This includes any disruption in the order of the nominal execution, which can be a different execution path, or simply a missing log. Other methods also focus on distribution anomalies, defined by unexpected concentration of logs (too many or too few), and include the DNS attack use case (Nandi et al., 2016; He et al., 2016b; Meng et al., 2019). Finally, some papers offer to study temporal anomalies, which can be defined as discrepancies compared to the nominal temporal relationships between event types (Xu et al., 2009a; He et al., 2016b). Note that these anomalies are not necessarily exclusive, for instance, a nominal sequence A, B can be abnormal if B happens before A. This is a sequential anomaly, and yet, the value of the time elapsed between the logs is also necessarily abnormal. Yet, we did not find any of the state-of-the-art method that is able to treat all of these anomalies together.

2.3.2 Metrics used

In the survey of He et al. (2016b), as well as in the evaluation sections of the articles concerning log anomaly detection, external evaluation metrics are generally used to compare the detected anomalies to the ground truth, for datasets with available labels. Commonly, the traditional F1-measure is calculated to count :

— the true positives, i.e., the anomalies that are actually detected;

— the false negatives, i.e., the anomalies that were missed;

— the false positives, i.e., the normal points that are wrongfully labelled as abnormal.
Differently to a classification task, only the abnormal class is included in the measure. Indeed, averaging the F1-measures of the normal and abnormal class would lead to a misleading high recall when optimizing the method : since most of the data are normal, systematically predicting a normal data leads to a high score.

The counts of the true positives, the false negatives and the false positives are generally based on the chosen log partition. Hence, the log partition should be performed in accordance with the information provided by the labels : if the label only provides the normal or abnormal class of a sequence, based on a sequence identifier, further information is necessary to label the temporal windows. The best case scenario might be when the logs themselves are labeled, which provides the finest possible labelling, which can be extended to the labelling of the created log samples (Liang et al., 2007; Meng et al., 2019).

Some other articles value the fact of providing a temporally accurate anomaly detection in case of online functioning, in order to enable a reactive action. In some cases, the sooner the anomalies are detected, the better (Kimura et al., 2018; Meng et al., 2019). In this case, the articles also compare the time of alerting of the different understudied methods. Moreover, providing a more temporally precise anomaly detection to the end-users can enhance the manual post-processing task (Xia et al., 2020).

In our evaluation study, Chapter 4, we calculate the F1-measure on temporal windows. This enables to introduce a temporal detection anomaly. Moreover, we refine the classical F1-measure to incorporate a temporal tolerance between the timestamp of the anomaly and its detection.

2.3.3 Main conclusions

We finally present some of the conclusions we have extracted from our analysis of the evaluation parts of the understudied articles and from the survey.

2.3.3.1 Type of supervision

We first describe here the main conclusions of both the theoretical principles of the type of supervision, and the experimental assessments found in the state-of-the-art. In Table 2.1, we census the features of evaluation of the three types of methods, and detail some important

Supervised methods are more accurate than unsupervised ones. He et al. (2016b) performs an experimental study on different types of anomaly detectors. The authors conclude that supervised methods perform more accurate results than unsupervised ones on the understudied reference datasets. Thereafter, most of the novelty detection approaches use the same baselines for comparison and confirmed these results (Du et al., 2017; Meng et al., 2019). The generally limited accuracy of unsupervised methods is justified in He et al. (2018b) by the fact that outliers are not necessarily anomalies. This means that anomalies should be rare and clearly distinct from the majority of points, regarding selected features. He et al. (2018b) acknowledge as a limitation of their approach that these two criteria are not necessarily sufficiently verified to enable unsupervised methods to reach the detection scores of the supervised approaches.

	Supervised	Unsupervised	Based on novelty
Does not require labels	✘	✓	✓
Does not require balanced data	✘	✓	✓
Can catch new types of anomalies	✘	✓	✓
Can catch more than outliers	✓	✘	✓
Has high accuracy (in benchmarks)	✓	✘	✓

Table 2.1 – Comparison of the different type of supervision for the anomaly detection problem. While supervised methods are accurate, they are not convenient to apply. Unsupervised methods only detect outliers and are therefore inaccurate. Novelty detection methods are preferred since they are accurate and can be easily used.

Supervised methods are inconvenient. A first drawback of these approaches is that they rely on the availability of the labels, while it is difficult to obtain fully-labeled data in production environment (He et al., 2018b; Xia et al., 2020). Moreover, their reported high accuracy is only valid in case of approximately balanced data. Yet, in anomaly detection scenarios, the abnormal samples are minority compared to normal samples. In this situation, a binary classifier is likely to learn that systematically predicting the normal class results in an optimal loss score. Moreover, the scarcity of abnormal samples also generates difficulties in precisely defining the abnormal class while training. Finally, since the model is as a two-class classifier, supervised methods are not designed to detect new and unknown types of anomalies, but rather to recognize the anomalies they have been trained to detect.

For all of these reasons, supervised methods are seldom used in real-world applications, and unsupervised methods are rather preferred, especially because they do not require any extensive labelling efforts.

Novelty detection offer a fair alternative. Novelty detection methods constitute a fair alternative. They require neither labelled data nor balanced data. They can catch new types of anomalies, that are not necessarily outliers (Chalapathy and Chawla, 2019), and experimentally present accurate results. For this reason, we consider that a novelty detection approach is a desired feature for an anomaly detection method.

2.3.3.2 Detectable anomalies

The ability to catch a large variety of types of anomalies is key for a method to be generic, and to be recommendable for the detection of new systems, with unknown anomalies. It is also desirable in the case of root cause analysis, where any abnormal behaviour can be a part of the explanation of a broader observed anomaly.

Deep learning methods catch more complex dependencies. The comparison of modern deep learning methods to traditional machine learning techniques, both in supervised and supervised way unanimously shows the superiority of the deep learning models. Du et al. (2017) argue that novelty detection based on flow mining also fails in catching the complex dependencies of event types, especially with the issue of representing concurrency. While most of the neural architecture

			Data	Detectable anomalies				
Method	DL	Novelty	driven	Seq.	Temp.	Dist.	C. seq.	C. temp.
Decision tree	✘	✘	✓	✓	✓	✓	✘	✘
Liang et al. (2007)	✘	✘	✘	✓	✓	✓	✘	✘
Kimura et al. (2018)	✘	✘	✘	✓	✓	✓	✘	✘
He et al. (2016b)	✘	✘	✓	✓	✓	✓	✘	✘
Logistic regression	✘	✘	✓	✓	✓	✓	✘	✘
LogRobust	✓	✘	✓	✓	✘	✘	✓	✘
Lu et al. (2018b)	✓	✘	✓	✓	✘	✘	✓	✘
LogGAN	✓	✘	✓	✓	✘	✘	✓	✘
PCA	✘	✘	✓	✓	✓	✓	✘	✘
Xu et al. (2009b)	✘	✘	✓	✓	✘	✘	✘	✘
Lin et al. (2016)	✘	✘	✓	✓	✘	✘	✘	✘
Log3C	✘	✘	✘	✓	✓	✓	✘	✘
Log2vec	✓	✘	✘	✓	✓	✓	✓	✘
IM	✘	✘	✓	✓	✓	✓	✘	✘
Farshchi et al. (2015)	✘	✘	✘	✓	✓	✓	✘	✘
HMM	✘	✘	✓	✓	✓	✓	✘	✘
FSM	✘	✓	✘	✓	✓	✓	✘	✘
Lu et al. (2018a)	✘	✓	✘	✓	✓	✓	✘	✘
CFG	✘	✓	✓	✓	✘	✓	✘	✘
Shang et al. (2013)	✘	✓	✓	✘	✘	✓	✘	✘
Deeplog	✓	✓	✓	✓	✓	✓	✓	✘
LogAnomaly	✓	✓	✓	✓	✘	✓	✓	✘
Total	6	6	15	21	14	17	6	0
NoTIL	✓	✓	✓	✓	✓	✓	✓	✓

Table 2.2 – Summary of the state-of-the-art methods for the anomaly detection task. DL : the method relies on deep learning. Novelty : the method is based on novelty detection. Data driven : the method only uses the logs (no domain-specific data required). Detectable anomalies : the method is designed to detect the corresponding type of anomaly, among sequential (Seq.), temporal (Temp.), distribution (Dist.) and complex sequential and temporal (C. seq. and C. temp).

(CNN, GAN ...) can be considered, the LSTM are commonly preferred for their capacity to model complex sequential relations.

The importance of log partition. The choice for the log partition is crucial to select the features used to model the logs. In our study we chiefly observed two representations of the logs : (i) a temporally-ordered sequence of the event types, and (ii) a temporal event count of the logs. While the sequential representation is able to detect sequential anomalies (e.g., violation of the order, unknown execution path, missing log), they cannot detect purely temporal anomaly (e.g. performance anomalies). The log partition choices therefore have a direct impact on the genericity of the method, and its ability to detect a large number of anomaly types.

Table 2.2 summarizes the desired characteristics of anomaly detection methods. Only 6 out of 22 studied methods are based on novelty detection, among which only DeepLog and LogAnomaly are based on deep learning. Hence, we choose these two methods as baselines. However, none of the existing method can catch all the types of anomalies simultaneously. Especially, none of them is designed to detect complex temporal anomalies.

We therefore propose NoTIL, a new anomaly detection method based on the detection of novelties in logs. NoTIL represents the logs with a temporal event count of the event types, based on a customisable window size. Our method can therefore detect temporal and distribution anomalies, but also sequential ones, depending on the chosen size of the window. Moreover, NoTIL relies on a deep learning model to learn the nominal behaviour in logs, which enhances the modelling of complex sequential and temporal patterns. As a result, we estimate that NoTIL is the only method able to catch all the types of anomalies. The next chapter details the implementation of NoTIL.

3 Log anomaly detection with NoTIL

In this chapter, we present our contribution for the anomaly detection problem. We detail NoTIL, a novelty detection-based approach that aims at alerting on unexpected behaviours of the log system, compared to a reference learned behaviour. We propose two versions of NoTIL : NoTIL-LSTM and NoTIL-AE, that represent two different prediction tasks. NoTIL is published in (**?**).

3.1 Overview of the contribution

In this section we first sum up the different desirable features for the log anomaly detection task. In parallel, we provide an overview of the implementation of our solution to answer these requirements. From our reading of the literature, we estimate that the following features are desirable for an algorithm that performs anomaly detection on logs.

Novelty detection approach. Our reading of the literature first exhibits the superiority of novelty detection approaches over traditional supervised and unsupervised methods. Most of the recent novelty detection methods rely on the application of an intermediate prediction task. For these methods, a prediction task is learned on an anomaly-free training set, and applied on a testing set. The result of the prediction task is used to detect anomalies : in the testing set, the normal samples are supposed to be like the anomaly-free training samples, and therefore, should present satisfactory results. On the contrary, the abnormal samples, which were not included in the training phase, are new to the predictor, and are therefore expected to provide low prediction results on the abnormal samples. This difference of behaviour towards the intermediate prediction task is used to separate normal and abnormal samples.

NoTIL matches the definition of a novelty detection approach. It learns a nominal behaviour on anomaly-free logs through a prediction task. We propose two different predictions : (a) forecasting the next element, and (b) reconstructing the input. For both of them, the output is either labelled as normal, if the prediction task is successful, or alerted as anomaly otherwise.

Deep learning model. The recent increasing interest in deep learning techniques, confirmed by the results of the evaluation of the most recent papers, indicates that deep learning methods are more accurate for the anomaly detection of logs. Indeed, these models are able to efficiently learn complex relationships between the features. NoTIL relies on deep learning models for its

prediction task, for which we proposed two different implementations : a simple LSTM, reputed to catch the sequential dependencies of data (here the temporal relations), and an auto-encoder, composed of stacked LSTM, which is also able to extract relevant features, here, from the different event types.

Adapted data representation. We finally note that none of the modern deep learning anomaly detectors based on novelty detection contains a data representation that fits all the types of anomalies, i.e., temporal, sequential and distribution anomalies. For instance, none of them is able to tackle performance anomalies, that are manifested within the logs as an abnormally long time-frame separating two consecutive logs of different event types. Contrary to the state-of-the-art methods, NoTIL represents the logs with a counting time window of the event types, taking into account the time elapsed between logs.

In the remaining of the chapter, we first formalize the problem of detecting anomalies in logs. We then detail the implementation of NoTIL that answers the aforementioned challenges.

3.2 Problem formalizing

This section formalizes the problem of detecting anomalies in a dataset of logs. We first introduce some notations, then present some definitions linked to the context of log anomaly detection. Finally, we provide a comprehensive problem statement.

3.2.1 Notations and definitions

Let L be a dataset of logs, occurring in a discrete time-frame $[\![1..T]\!]$, $T \in \mathbb{N}^+$. E is the set of possible event types in L. Table 3.1 lists the main notations used in this chapter, and the definitions associated to the dataset L.

We note $\mathcal{A}$ the set of anomalies in L. We note $\mathcal{P}$ the definition space of the anomalies (such that $\mathcal{A} \subset \mathcal{P}$) : depending on the datasets, $\mathcal{A}$ can be :

— a subset of abnormal sequences (e.g. HDFS) : $\mathcal{P} = S$;

— a subset of abnormal logs (e.g. BGL) : $\mathcal{P} = L$;

— a subset of abnormal instants (e.g. OpenStack) : $\mathcal{P} = [\![1..T]\!]$.

Our anomaly detection is based on the detection of abnormal instant $\hat{\mathcal{T}}_{ano} \subset [\![1..T]\!]$. We therefore define a function α that aims at converting the predicted temporal anomalies to the desired set for comparison : $\alpha : \hat{\mathcal{T}}_{ano} \mapsto \alpha(\hat{\mathcal{T}}_{ano}) \subset \mathcal{P}$. Appendix B.1 details this transformation. Hence, $\mathcal{A}$ and $\alpha(\hat{\mathcal{T}}_{ano})$ can be compared to evaluate the accuracy of the detection. We define the function Δ, the detection score :

$$\begin{array}{rccc} \Delta & : & 2^{\mathcal{P}} \times 2^{\mathcal{P}} & \rightarrow & \mathbb{R}^+ \\ & & \mathcal{A},\ \alpha(\hat{\mathcal{T}}_{ano}) & \mapsto & \Delta\left(\mathcal{A},\ \alpha(\hat{\mathcal{T}}_{ano})\right) \end{array}$$

Δ measures the similarity between two subsets of a same set $\mathcal{P}$. For instance, the F1-measure can implement Δ.

Notations	
$S = \{s_1, .., s_k\}$	Set of k unique elements
$[\![a..b]\!]$	The set of integers between a and b (included)
$\|S\|$	The cardinality = number of elements in S
$\forall x \in \mathbb{R}, \lfloor x \rfloor$	The floor function defines as $max\{m \in \mathbb{Z} \mid m \leq x\}$
Dataset L, with event types E, and timeframe $[\![1..T]\!]$	
$\ell \in L$	A log of L
$e_\ell \in E$	The event type of ℓ
$t_\ell \in [\![1..T]\!]$	The timestamp of ℓ
$\mathcal{T}_{train}, \mathcal{T}_{valid}, \mathcal{T}_{test} \in [\![1..T]\!]$	The temporal instants associated to training, validation and testing datasets
$\mathcal{A} \subset \mathcal{P}$	The anomalies in L
Predictor Ψ	
(x_t, y_t)	Input and output of the prediction task at t
θ	The parameters that are learning during the training (e.g. neural network weights)
φ	The hyper-parameters that are optimized during the validation (e.g. neural network number of layers)
θ_φ	The optimized value of θ, with the hyper-parameters φ
$\mathcal{L}$	The loss function
$\hat{\mathcal{T}}_{ano} \subset [\![1..T]\!]$	The predicted abnormal instants
$\alpha(\hat{\mathcal{T}}_{ano}) \in \mathcal{P}$	A function that tranposes the predicted abnormal instants to the space of anomalies (e.g. abnormal sequence)
$\Delta\left(\mathcal{A}, \alpha(\hat{\mathcal{T}}_{ano})\right)$	Similarity measure between two comparable sets (e.g. F1-measure)

Table 3.1 – Notations used throughout the chapter to formalize the problem of novelty detection.

3.2.2 Problem statement

We formally describe the problem statement of detecting anomalies in log datasets. Novelty detection methods are based on an intermediate prediction task and usually process in three steps :

— the training phase, which aims at training the predictor to fit the training set, an anomaly-free subsample of the data. It consists in optimizing the trainable parameters of the model θ (e.g. the weights and biases of a neural network);

— the validation phase, which aims at optimizing the hyper-parameters of the model so as to separate normal and abnormal samples of the validation set. These hyper-parameters include the parameters regarding the architecture φ (e.g. number of layers, neurons of a neural network), and a detection threshold $\tau \in \mathbb{R}$, that divides the samples between normal and abnormal data, according to their prediction;

— the testing phase, which evaluates the anomaly detection on a testing set.

Training phase. The prediction task is first trained on a anomaly-free dataset. We define $\mathcal{T}_{train} \subset [\![1..T]\!]$, the anomaly-free subset of instants used for training. We call Ψ the parametric predictor. For a given hyper-parameter configuration φ, we aim at finding the best trainable parameters θ. For any $t \in \mathcal{T}_{train}$, let (x_t, y_t) be the associated couple of input and output for the prediction task. The predictor associates a prediction to x_t, $\hat{y}_t = \Psi(\varphi, \theta, x_t)$. This prediction is compared to the real output y_t with a loss function $\mathcal{L}$, that we want to minimize over the samples :

$$\underset{\theta}{\text{minimize}} \sum_{t \in \mathcal{T}_{train}} \mathcal{L}(\varphi, \theta, x_t) \tag{3.1}$$

We call θ_φ the optimized value of θ for the hyper-parameter configuration φ.

Validation phase. We apply the trained predictor to the validation set. We define $\mathcal{T}_{valid} \subset [\![1..T]\!]$, the subset of instants used for validation, and $\mathcal{A}$ the subset of abnormal samples in the validation set. The detection threshold τ is used to separate abnormal and normal samples according to the loss of the prediction task. We define $\hat{\mathcal{T}}_{ano} \subset \mathcal{T}_{valid}$, the set of detected anomalies :

$$\hat{\mathcal{T}}_{ano} = \left\{ t \in \mathcal{T}_{valid},\ \mathcal{L}(\varphi, \theta_\varphi, x_t) > \tau \right\} \tag{3.2}$$

The validation consists in optimizing the hyper-parameters φ and τ so as to maximize the detection score :

$$\underset{\varphi, \tau}{\text{maximize}}\ \Delta\left(\mathcal{A},\ \alpha(\hat{\mathcal{T}}_{ano})\right) \tag{3.3}$$

Testing phase. The optimized anomaly detection is then applied to a testing dataset, defined by the subset of instants $\mathcal{T}_{test} \subset [\![1..T]\!]$. The detected anomalies $\hat{\mathcal{T}}_{ano}$ are extracted as per in equation 3.2, with the previously optimized parameters φ, θ_φ and τ. Finally, we measure the detection score with $\Delta\left(\mathcal{A},\ \alpha(\hat{\mathcal{T}}_{ano})\right)$.

In conclusion, proposing a novelty detection method consists in defining the prediction task with :

— the features that define the couple input and output (x_t, y_t);
— the architecture of solution and the associated hyper-parameters φ and θ;
— the expression of the associated loss function $\mathcal{L}$.

3.3 NoTIL, a temporal novelty detection method

In this section, we detail the implementation of NoTIL, our proposition for the anomaly detection task. NoTIL stands for Novelty detection based on Temporal Irregularities in Logs. In subsection 3.3.1, we first detail our data creation process which includes both the temporal log partition method, and the feature selection. In subsection 3.3.2, we propose two versions for the prediction task of NoTIL.

3.3.1 Data representation

This subsection details our choices for the data representation. We both describe our temporal log partition method, and the features we select for each created sample.

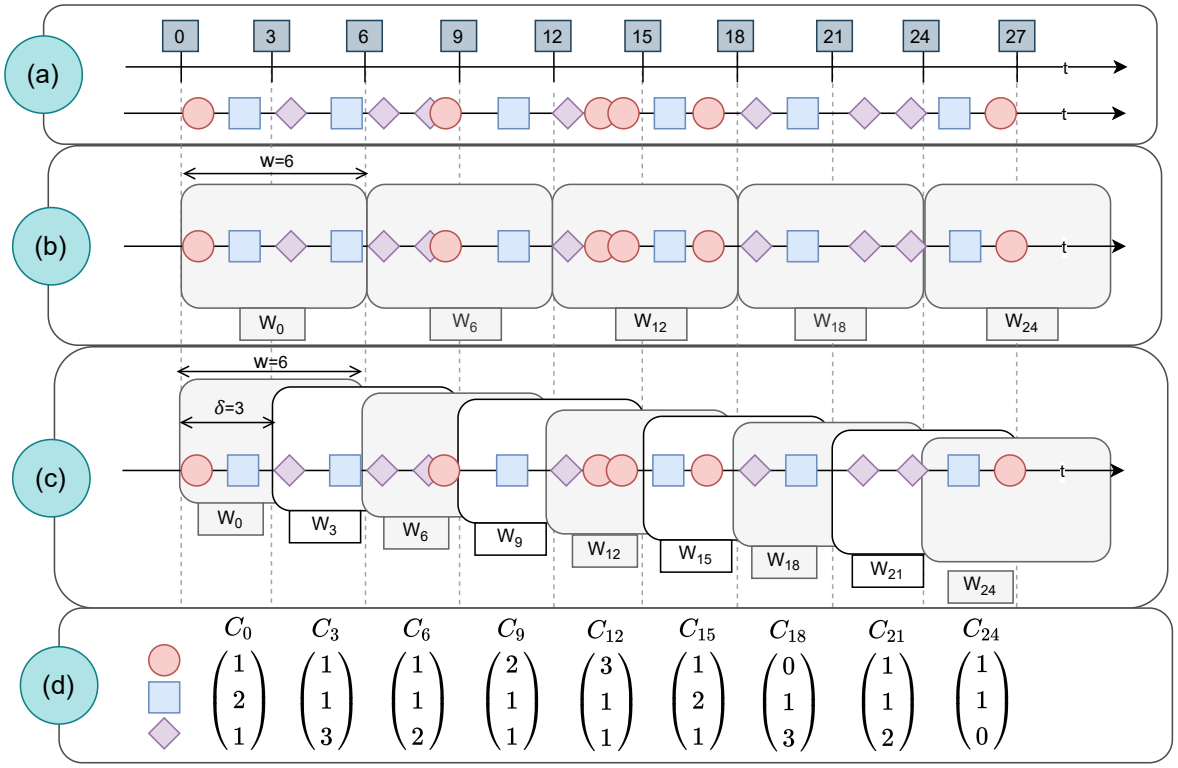


Figure 3.1 – Illustration of the chosen data representation with (a) an example of timestamped events (logs), where each pictogram represents a different event type, (b) the cut of the global time frame to generate fix windows of size $w = 6$; each window is designated by its starting timestamp, (c) the creation process of sliding windows of size $w = 6$ and with a sliding offset $\delta = 3$, (d) the event count vectors obtained by counting, in each sliding window, and for each event type, the number of logs of this event type in the window.

Log partition. We first present our proposition for log partition, illustrated in Figure 3.1. We take the example of logs shown in Figure 3.1(a), where each pictogram represents a different event type. Temporal log partition enables the introduction of time into the anomaly detection framework. Hence, we decide to adopt a temporal log partition. We propose to cut the overall time frame into temporal windows, as presented in Figure 3.1. To create these windows, we define the *window size* $w \in \mathbb{N}^*$. For an instant $t \in [\![1..T]\!]$, the window W_t represents the state of the system — defined by the logs occurring — between the instants t and $t + w$. A log ℓ belongs to a temporal window W_t iff $t \leq t_\ell < t + w$. We call $\mathbb{W}$, the set of windows created on the overall datasets. The definition of $\mathbb{W}$ requires to specify the set of instants t that leads to the generation of a window W_t. Two main strategies exist : the fix and the sliding windows.

Fix windows. The fix window partition is illustrated in Figure 3.1(b). A new window is started every w instant. The instants t that are associated to a window W_t are the multiples of w :

$$\mathbb{W} = \left\{ W_t \mid t < T \wedge \exists\, n \in \mathbb{N},\ t = w \times n \right\} \tag{3.4}$$

In other words, creating fix windows simply consists in cutting the overall time frame into disjoint consecutive windows of size w. We observe that this process arbitrarily separates consecutive and potentially close elements. In Figure 3.1(b), the last square pictogram of W_0 is temporally close to

the first diamond pictogram of W_6, and yet, they are separated. The separation strongly depends on the value of w. To avoid, this strong dependence, sliding windows are often preferred.

Sliding windows To build sliding windows (Figure 3.1(c)), we define δ as the sliding offset. The sliding windows remain of size w, but are started every δ instant :

$$\mathbb{W} = \{ W_t \mid t < T \ \wedge \ \exists\, n \in \mathbb{N},\ t = \delta \times n \} \tag{3.5}$$

Generally, we choose $\delta < w$, so that we do not skip any instant, and choosing δ as a divider of w insures to have an equal representation of all the timestamps. With this representation, we avoid to cut the samples in a way that risks to separate meaningful parts of the understudied logs. We therefore choose the sliding window mechanism.

Feature selection We propose to represent each window by its event count vector. The event count vector of a window W_t is noted $C_t \in \mathbb{N}^{|E|}$, and counts, for each event type $e \in E$, the number of occurrences of e in the window W_t. Formally, the i^{th} element of C_t corresponds to the number of logs of the i^{th} event type, and is expressed as :

$$C_t^{(i)} = \left| \{ \ell \in L \mid t_\ell \in [\![t..t+w]\!] \ \wedge \ e_\ell = e_i \} \right|$$

In Figure 3.1(d), we represent these event counts, which are obtained by summing, for each sliding window, the number of every pictogram (representing different event types). This representation can take time into account and measure important concentrations of logs. Moreover, with the appropriate selection of w and δ, this representation can still keep a track of the order of the logs, and therefore detect sequential anomalies.

3.3.2 Novelty detection approach

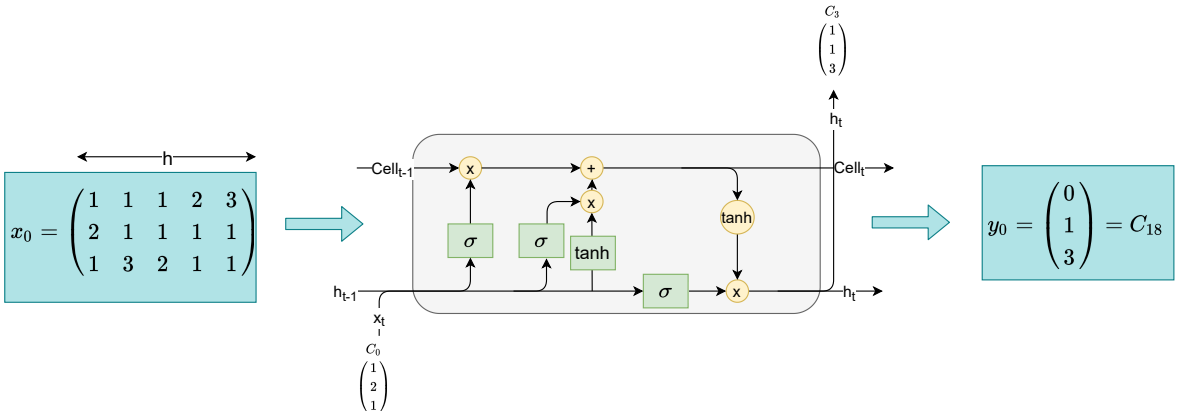
NoTIL is a novelty detection method. Its definition relies on the description of its associated prediction task. In this subsection, we detail two propositions for the prediction task, which implement the previously mentioned predictor Ψ : NoTIL-LSTM and NoTIL-AE are two versions of our contributions, each associated to a different prediction task.

3.3.2.1 Prediction task : time series forecasting

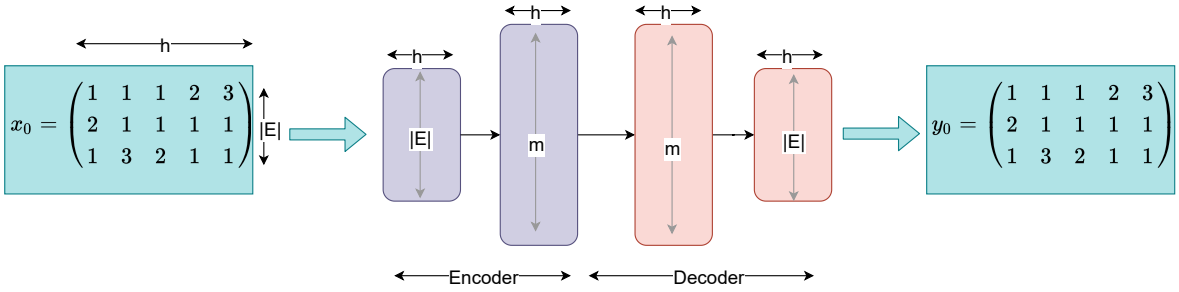
We first define NoTIL-LSTM. As in DeepLog and LogAnomaly, we train a model for the task of forecasting the next window. The idea is to predict an event count vector based on the past event counts. We aim at modelling the sequential relationships between event counts, and to detect transgressions of these relations as anomalies.

Input - Output. Du et al. (2017) define a hyper-parameter h, called *look-back*, so that h consecutive windows are used to forecast the next window. Hence, a couple input - output $(x_t,\ y_t)$ is expressed as :

$$x_t = \begin{pmatrix} C_t & C_{t+1} \dots & C_{t+h-1} \end{pmatrix},\ y_t = C_{t+h} \tag{3.6}$$



(a) An illustration of the LSTM functioning, including the detailed implementation of a LSTM cell. The input contains h consecutive event count vectors, that are passed one after the other to the LSTM, which updates its hidden state (h_t) and cell state (C_t) and predicts the next vector. The final prediction is the $h+1$ event count.



(b) An illustration of the auto-encoder functioning. The goal is to reconstruct the input after a set of transformations that aims at modifying the dimension (number of events). Note that each transformation is made by an LSTM network.

Figure 3.2 – Presentation of the mechanisms of the two proposed prediction tasks, with an example of the first sliding windows of Figure 3.1

Due to the sliding window mechanism, consecutive windows share common instants and logs. Especially, the last vector of the input share much more instants with the output vector than the first vector of the input, as depicted in Figure 3.3(a). This configuration might introduce an important bias in the prediction task. To avoid it, the input x_t is associate to the output $y_t = C_{t+\beta}$, where $\beta = \lfloor \frac{w}{s} \rfloor + h - 1$ (Figure 3.3(b)).

Predictor architecture. As for most of the recent studies, we opt for a deep learning technique to train the prediction task. Indeed, deep learning algorithms are reputed to be powerful in modelling complex relationships between features. Moreover, the sequential relation between the elements of the input and the output motivates us to use a Recurrent Neural Network (RNN). We especially use the popular LSTM (reputed to avoid the vanishing / exploding gradient issue, faced by the other RNN). Figure 3.2a details the architecture of the LSTM, which includes several gate mechanisms, and aims at sequentially introducing the elements of the input in order to predict the next one. The final iteration is the prediction of the output.

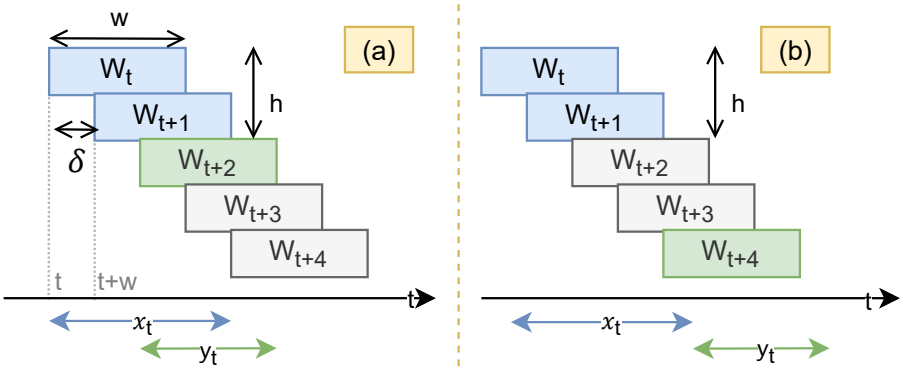


Figure 3.3 – A set of $h = 2$ input windows of size $w = 6$ and shifted by $s = 2$ (blue) and its associated output (green) in an overlapping scenario (a) and a non-overlapping one (b)

Loss function. For each input x_t, the LSTM predicts an output $\hat{y}_t$, which is compared to the real output y_t. We measure the error of prediction with the Mean Square Error (MSE), defined as:

$$\mathcal{L}(\varphi, \theta, x_t) = \|\hat{y}_t - y_t\|_2$$

The loss is used both as the value to optimize during the back-propagation of the LSTM during the training phase, and as a measure of the error for the validation and test phases.

3.3.2.2 Prediction task : time series reconstruction

We now define NoTIL-AE. We study the reconstruction of a time series as the prediction task. In this case, the goal is to train a model that is able to reconstruct the input, after a series of operations that modify the dimension, as presented in Figure 3.2b.

Input - Output. The relation between the input and the output becomes trivial. The input x_t is associated with $y_t = x_t$, where x_t is defined as follow :

$$x_t = \begin{pmatrix} C_t & C_{t+1} \dots & C_{t+h-1} \end{pmatrix} = y_t$$

Predictor architecture. To perform this reconstruction, we use an auto-encoder, a deep learning architecture that stacks several neural networks with different input and output sizes. The stacked neural networks create intermediate states of the matrix x_t. These stacked neural networks are generally divided in two successive groups : the encoder modifies the original input in order to create an intermediate representation, while the decoder transforms this representation into the output, with the original dimension. Auto-encoders are reputed to be able to retrieve the most relevant features, thanks to these intermediate dimension changes. For each neural network, we choose an LSTM architecture, for their ability to capture sequential patterns. Note that the dimension that is reduced or augmented during the process is the number of events $|E|$, while the dimension h is left untouched, and used for the LSTM intermediate network to perform iterative predictions.

Loss function. To compare an input $x_t = y_t$ to its reconstruction $\hat{y}_t$, we also use the MSE loss function, calculated between the full output matrices.

3.3.3 Temporal precision of the detection

During the validation and test phases, a prediction error is attributed to the output, which is labelled as normal or abnormal. For the forecasting version, the output y_t is a temporal window of size w. For the reconstruction task, the output y_t is the whole matrix of consecutive windows of size w. The temporal precision of the anomaly detection might not be satisfactory if w is high. We propose a more accurate temporal labelling, which can reach the precision of s. In the Figure 3.1(c), the blue square visible in the window W_4 appears for instance in both W_4 and W_5. We propose to label every separate sub-windows of size s by averaging the errors of the corresponding full windows. In the example, the error associated to the small window, of size s, that only contains the blue square, is the average of the errors of W_4 and W_5.

4 Validation

This chapter evaluates the two versions of our method NoTIL. We assess the efficiency of our anomaly detection method on a wide range of types of anomalies. We compare the accuracy results of NoTIL to the most up-to-date state-of-the-art methods, namely DeepLog and LogAnomaly, and analyze the relevance of the different anomaly detection methods.

4.1 Evaluation framework

In this section, we detail our evaluation framework. We provide an overview of the understudied data, list the state-of-the-art methods we compare to, detail our proposition for the accuracy metric and provide some guidelines on the optimization of the hyper-parameters.

4.1.1 Data presentation

We first provide an overview of the datasets on which we perform the anomaly detection task. We propose to evaluate the understudied methods both on simulated and real-world datasets.

Simulated data. We first generate artificial datasets in order to study the behaviour of the methods in a minimal configuration. Ours goals are to (i) identify and classify the types of anomalies, in their minimal configurations, (ii) draw conclusions on the behaviour of each method regarding each type of anomaly. Due to the high complexity of real world datasets, we believe that this step is crucial to understand the behaviour of the methods, and explain their results on more sophisticated datasets.

We pay a particular attention to the labelling of the anomalies. As we perform a temporal anomaly detection, we precisely need to determine, in each situation, which instants we consider as abnormal. In the description of each scenarios, we therefore provide the temporal labelling. Generally speaking, when a temporal or sequential relationship is not respected, we hold the second member of the relationship as accountable. Indeed, we work in a prediction functioning, meaning that a log or a temporal window is generated based on the previous logs. Hence, if this new log does not respect the nominal behaviour regarding the already generated logs, we consider this latter log as abnormal.

Real-world data. We also assess our log parser on real datasets. Some of the reference datasets are labelled with identified anomalies, either at a sequence level, or at a log level. We select an example of each (OpenStack and BGL), and apply the detection of anomalies. These contexts are generally far more complex than the minimal scenarios. They contain other contextual logs, noise in the temporal relationships, and complex sequential patterns are involved. We are interested in analyzing whether the same behaviours can be observed on the identified types of anomalies represented in these datasets. We also study the robustness of the methods to more complex contexts.

4.1.2 Baselines

We compare the two versions of NoTIL to some of the most up-to-date novelty detection approaches. We select :

— DeepLog (Du et al., 2017), the state-of-the-art reference. It is composed of two independent detection methods, that we distinguish with the following notations :
  — S-DeepLog, the main contribution, represents the logs sequences of event types;
  — P-DeepLog extends the method to the treatment of parameters. These parameters include the difference of timestamps of consecutive logs, for a given event type;
— LogAnomaly (Meng et al., 2019), a recent extension of the sequential version of DeepLog, with an additional attention mechanism.

P-DeepLog is capable of detecting temporal irregularities, as long as these irregularities are visible at the event-type level (i.e. it is not an anomaly between different event types). Since it processes each event type individually, P-DeepLog is globally not suited for the detection of sequential anomalies. Both LogAnomaly and S-DeepLog are designed to detect sequential anomalies. Nonetheless, they do not take into account the time elapsed between logs. Hence, they cannot detect temporal anomalies, or changes in the concentration of a log, unless they impact the observed order of event types.

4.1.3 Evaluation metric

Measuring the accuracy of an anomaly detector in our context consists in comparing the temporal windows that are labelled as abnormal to the windows that are retrieved as abnormal. We can compare these two sets with a traditional accuracy measure, such as the F1-measure. However, we observed that the classic definition of this measure is poorly suitable for the detection of temporal anomalies. Indeed, if an anomaly happens in an interval t, and is detected at $t+1$, the anomaly is regarded as not detected, introducing a false positive and a false negative. To illustrate the problem, an algorithm that would detect the anomaly at $t+20$ would have exactly the same score. Even worse, an algorithm that would not detect the anomaly at all, would have a better F1-measure, since no false positive would be counted. It clearly illustrates the need for an adaptation of this indicator.

We propose to introduce a parameter of tolerance $q \in \mathbb{N}$, and to redefine the false positives and false negatives as follow : for $q \in \mathbb{N}$, a real (resp. predicted) anomaly in W_t is a false negative (resp. positive) if no detected (resp. real) anomaly is observable within the sequence $(W_{t-q}...W_{t+q})$, otherwise it is a true positive. This way, we are able to tolerate the detection of anomalies with a delay. Yet, since the W_t are sliding windows, the more accurate the temporal detection is, the more

windows the actual and the predicted anomalies will share. Hence, temporally accurate detected anomalies still have a better score than slightly delayed ones.

We mention that q is not a hyper-parameter of the model, but instead, defines a user requirement on the temporal accuracy of the precision. q can be chosen to adapt to the domain requirements, yet, cannot be inferior to the temporal precision of detection (in our propositions, the size of the window decay δ). It is even preferable that q is a multiple of the size of the prediction. Generally, we choose q so as to contain a full sequence (when it is defined).

4.1.4 Experimental protocol

Execution environment Our experiments are conducted on a server with Ubuntu 18.04 installed and 192Go of RAM. The processor is a Bi Xeon Silver, 2.90GHz. We run the algorithms on the GPU, which is a better fit for the large matrices involved in the training of neural networks. Our algorithms are developed in Python with Pytorch framework (similarly to the implementation of (Wu, 2018)), and run on a NVIDIA 1080Ti GPU, with CUDA v11.

Parameter optimization NoTIL, like the other state-of-the-art methods contains different types of parameters : the trainable parameters of the prediction model θ and the hyper parameters φ. The trainable parameters are optimized as part of the training phase of the prediction task. This training is specifically the mechanism that learns the nominal behaviour of logs. Conversely, the hyper parameters need to be set a priori, and are optimized thanks to a validation dataset. We distinguish : (i) the window-related parameters w and δ, used to build the temporal windows, (ii) the neural network hyper-parameters, which set up the configuration of the network to be trained, and (iii) the detection threshold τ, which aims at separating the normal and abnormal samples, according to their prediction result, at the end of the process. This threshold can be seen as a lever to balance false positives and false negatives.

The neural network parameters are numerous (number of epochs, number of hidden layers, size of hidden layers...). Each of the neural network parameters and the window-related parameters has a wide space of possible values, detailed in Appendix B.2. It becomes infeasible to perform an exhaustive grid-search optimization. We rather opt for a random search approach : we randomly select the values of the parameters on their definition space, train the model, and retrieve the validation score. We repeat this operation 20 times, and select the configuration that provides the best final result.

Nonetheless, the values of the detection threshold τ are not chosen randomly and a priori. Instead, once the network is trained — for a given tested configuration — we perform a grid-search on this parameter only. Indeed, the involvement of this parameter is crucial, and particularly close to the final result. We therefore pay a much greater effort on the determination of this parameter. Besides, this grid search is also feasible due to the format of the parameter : we determine the threshold as a percentile p, where the point that render a prediction error that is superior to the p-percentile of the errors is considered as abnormal. This way, the space of value is, by definition limited to the interval $[0-1]$, and since anomalies are supposedly rare, even in the validation set, we can even select a higher subset (e.g. $[0.9-1]$). We however warn that the use of a percentile threshold, learned on a validation set and applied on a test set only works in the assumption that

both sets have similar proportions of anomalies; otherwise, it would be wiser to learn a threshold directly on the raw error values.

4.2 Simulated data assessment

This section reports the assessment of NoTIL on simulated data. We propose 10 different minimal scenarios that correspond to different types of anomalies. We first present these scenarios, explaining both the nominal behaviour they model and the anomalies occurring in these models. We then detail the generation mechanism of these scenarios. Finally, we provide the results of the anomaly detection task on these simulated data.

4.2.1 Scenarios presentation

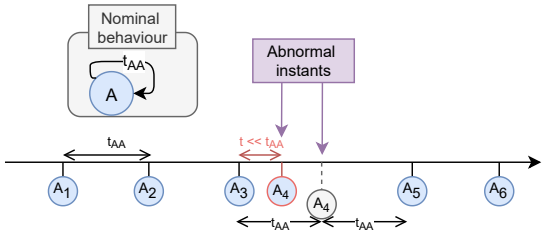
We propose a new topology of scenarios that represent minimal examples of the anomalies we want to model. In most of the cases, we also generated datasets where the temporal relationships are not strictly respected, but instead, contain some noise. For each scenario, we define (i) the nominal context of this scenario, (ii) the perturbation that defines the understudied anomaly, (iii) the temporal instants we choose to regard as abnormal, in order to obtain a temporal labelling of the simulated datasets.

4.2.1.1 Temporal anomalies

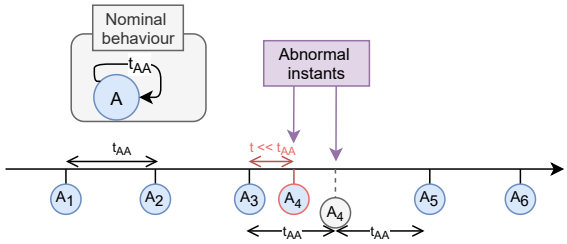
We first provide some examples of strictly temporal anomalies, presented in Figure 4.1. In these scenarios, the anomaly does not alter the temporally-ordered sequence of event types. Instead, the temporal relation between the logs is not respected for the abnormal samples. These types of anomalies are visible in the real-world datasets, such as in OpenStack, and can represent any performance anomaly. We propose 3 scenarios, in which the anomaly can be manifested as either an early or late appearance of logs compared to the expected behaviour. Whereas each of the illustrated example chooses one of the option, all of them can be applied for early and late arrival anomalies. We choose to label as abnormal both (i) the moment when the log occurred, which is abnormal since the log should not have occurred at this timestamp, (ii) the moment when the log should have occurred, which is abnormal since the log did not actually occur.

periodic_t. The first scenario consists in a set of logs containing only one event type A. As presented in Figure 4.1a, the logs of A happen at a regular time interval, hence why the event type A is described as periodic. The anomaly described in this scenario consists in observing an abnormal time period between two consecutive occurrences of logs of A. In the example, A_4 occurs at an abnormally shorter time period after A_3. We choose to label as abnormal both the instant when A_4 occurs, since at this moment, no occurrence of A should be observable, and the moment when A_4 should have occurred, since no occurrence of A is observable whereas it should be.

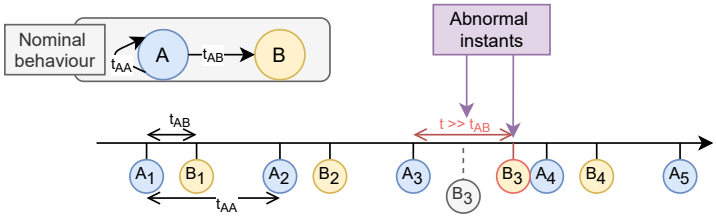
sequence_t. The temporal anomaly in a repeated sequence of logs (Figure 4.1b) is observable in a nominal scenario of two event types arriving one after the other : the appearance of A triggers



(a) Temporal anomaly on a periodic event type (**periodic_t**). In this scenario, a unique event type is repeated at a regular time interval. The anomaly consists in an abnormal value of this temporal interval : a log A occurring too soon (as presented with A_4) or too late. The abnormal instants are the moments when A_4 should have occurred and the moment when it actually occurred.



(b) Temporal anomaly on a sequence of two event types (**sequence_t**). In this scenario, two event types A and B generate logs one after the other. The anomaly consists in an abnormal value of the temporal relation A – B : a log B occurring too soon or too late (as presented with B_3) after the corresponding A. The abnormal instants are the moment when B_3 should have occurred and the moment when it actually occurred.

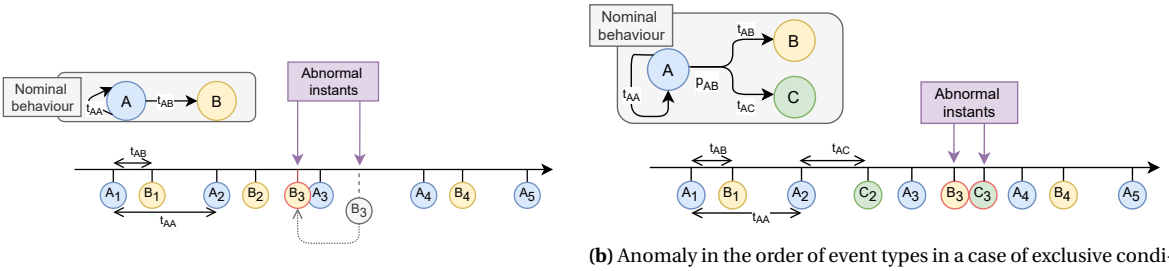


(c) Temporal anomaly on a periodic sequence of two event types (**periodic_sequence_t**). In this scenario, the event type A is periodic and generated at regular interval. Each occurrence of A generates a B at a regular interval. The anomaly description and the abnormal instants are similar to **sequence_t**.

Figure 4.1 – Scenarios with a temporal anomaly. The nominal behaviours are schematized in the grey boxes. The temporal axis contains both nominal and abnormal sequence executions. The purple arrows indicate the instants that we label as abnormal. These scenarios mainly represent the case of log occurring too early or too late compared to the nominal behaviour. We represent either an early or late appearance, yet both are applicable.

the occurrence of B after a time interval of t_{AB}, and this occurrence of B is responsible for the generation of a log of A after a time interval of t_{BA}. In this scenario, a temporal anomaly is observed when one of the two temporal relations is violated. Here, the relation t_{AB} is not respected for the couple A_3 and B_3, since B_3 happens after a time t that is significantly larger than the nominal t_{AB}. As such, the following of the scenario is impacted : since B_3 is delayed, and even though the system is back to normal right after, the log A_4 is only generated t_{BA} after the abnormal log, and is therefore late compared to what it would have been in a nominal version. Nevertheless, the log A_4 behaved as expected, and we do not want to propagate the abnormal labelling to the rest of the session. Hence, we simply label the expected and actual arrival times of B_3 as abnormal.

periodic_sequence_t. This scenario concerns a nominal behaviour with two event types (Figure 4.1c). Contrary to **sequence_t**, one of the event type, A, is periodic and self-generated, while the other event type is triggered by the appearance of the first. Here, the A occurrences are generated every t_{AA} instants, while the B are triggered after the appearance of A and occur after a time interval of t_{AB}. We define the temporal anomaly as an early or late arrival of B after the corresponding A. The Figure 4.1c exhibits the same example as for the scenario **sequence_t**. Yet, in this case, the following of the generated data is not impacted by the anomaly : since the A occurrences are generated independently of those of B, A_4 occurs at the expected timestamp, namely t_{AA} after A_3.



(a) Anomaly in the order of a sequence of two event types (**periodic_sequence_o**). The nominal behaviour is the one described in the scenario **periodic_sequence_t**, Figure 4.1c. Here, the anomaly consists in an inversion of the order of appearing : B_3 happens before A_3. We label as abnormal the timestamp of B_3, as well as the instant when B_3 should have occurred.

(b) Anomaly in the order of event types in a case of exclusive condition **XOR** (**xor_o**). The nominal behaviour is a periodic event type A, repeated at temporal intervals, followed by either a log B or a log C, with a probability p_{AB} that B is selected, and two different nominal separating times t_{AB} and t_{AC}. In the abnormal scenario, an occurrence of A, here A_3, is followed by both a log B (B_3) and a log C (C_3). The abnormal instants are the timestamps of B_3 and C_3.

Figure 4.2 – Scenarios with anomalies concerning the order of execution of event types in a sequence. The nominal behaviours are schematized in the grey boxes. The temporal axis contains both nominal and abnormal sequence executions. The purple arrows indicate the instants that we label as abnormal.

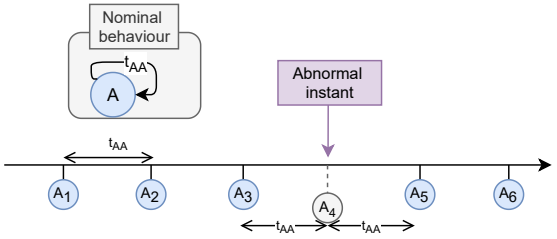
As a result, an abnormal time interval is also observable between B_3 and A_4. Similarly, we label as abnormal instants the expected and actual time of occurrence of B_3.

4.2.1.2 Abnormal order in sequences

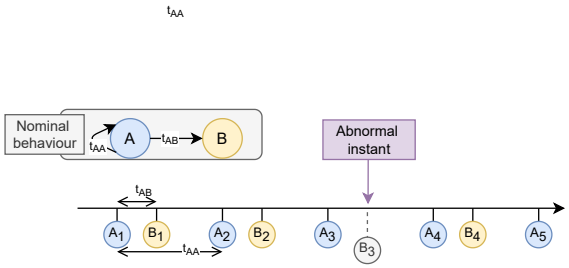
We now focus on sequential anomalies, with anomalies that concern the order of appearance of logs in a sequence. The logs which appearance modifies the order of the sequence are regarded as abnormal, and their timestamps are labelled as abnormal instants. These anomalies are frequent in the reference datasets (e.g. BGL, HDFS), and are the most widely treated by the state-of-the-art methods ((Fu et al., 2009b; Meng et al., 2019). While these anomalies can be described as sequential, they can also be observed in a temporal perspective, since the disturbances in the order of logs necessarily impact the nominal time intervals separating the logs. We present two different scenarios with distinct nominal behaviour.

periodic_sequence_o. The nominal behaviour of this scenario is described as the periodic sequence scenario, as for **periodic_sequence_t**. In this case, an anomaly occurs when a log of B happens before the corresponding log of A, locally reversing the order of apparition in the sequence. In Figure 4.2a, the log B_3 happens before the log A_3, whereas it should have occurred after a time interval t_{AB}. This modification of the order also has an observable impact on the undertudied time intervals between the logs; especially, the time interval between A_3 and B_3 becomes negative. We label as abnormal both the moment when B_3 should have happened, and the moment when it actually occurred.

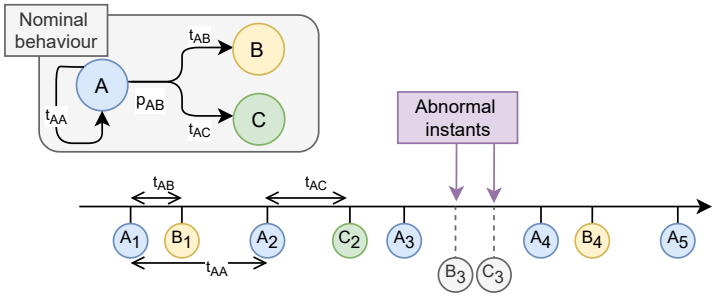
xor_o. First of all, we define a new nominal scenario of exclusive condition (Figure 4.2b. In this case, the logs of A are systematically followed by either a log of B or a log of C. This relation is expressed as a **XOR**, and defines a probability p_{AB} of selecting B after an occurrence of A. Similarly, the probability that A is followed by C is $1 - p_{AB}$. Moreover, we define two distinct temporal intervals : t_{AB}, that nominally separates an occurrence of A from the corresponding occurrence of



(a) Missing log anomaly in the context of a single periodic event type (**periodic_m**). The nominal scenario is described in **periodic_t**, Figure 4.1a. The anomaly is an occurrence of the event type that is missing, here A_4. The timestamp of A_4 is labelled as abnormal.



(b) Missing log anomaly in the context of a 2-event-type sequence (**periodic_sequence_m**). The nominal scenario is described in **periodic_sequence_t**, Figure 4.1c. The anomaly occurs when a log A, here A_3 is not followed by a log B as expected. In this case, we label the expected timestamp of the corresponding occurrence, B_3, as abnormal.



(c) Missing log anomaly in a XOR scenario (**xor_o**). The nominal behaviour is described in Figure 4.2b. The understudied anomaly occurs when an occurrence of A, here A_3 is followed by neither a log of B, nor a log of C. We label as abnormal the instants when both the occurrences B_3 and C_3 should have occurred.

Figure 4.3 – Scenarios with a missing log anomaly. The nominal behaviours are schematized in the grey boxes. The temporal axis contains both nominal and abnormal sequence executions. The purple arrows indicate the instants that we label as abnormal. These scenarios describe the unexpected absence of a log, compared to the normal behaviour. The abnormal instants are those when the logs should have occurred.

B, and t_{AC}, for the time interval separating A and C. In this scenario, the anomaly consists in observing an occurrence of A, here A_3 that violates the XOR relation by being followed by both an occurrence of B and an occurrence of C. In this case, we estimate that both the B and C occurrences are abnormal, and label their timestamps as abnormal instants.

4.2.1.3 Missing logs

Another type of scenarios focuses on the absence of an expected log in a sequence : a log that should happen is missing. We label the moment when the log should have happened as abnormal. Note that the log-level labelling, which is the most temporally accurate labelling in the existing real-world dataset, is not adapted for this kind of anomalies. Instead, only the sequence labelling can alert on a sequence that contains such an anomaly. A temporal labelling would of course be more suited to detect missing logs, yet, is seldom proposed in the reference datasets. Nevertheless, we are interested in proposing a method that is able to detect such abnormal behaviours, since they might be the root cause of an investigated issue.

periodic_m. In this scenario the nominal behaviour is the one described for the scenario **periodic_t**, with a single periodic event type. The anomaly, presented in Figure 4.3a, is described as the absence of an occurrence of the event type. Here, the log $\mathtt{A}_4$ is missing, and we label its expected timestamp as abnormal. It is worth noticing that, while this scenario describes a sequential anomaly, the local observation of the sequence of event type is not sufficient to detect it. Indeed, the sequence of event type is still A, A, A... Instead, including a temporal consideration enables this detection.

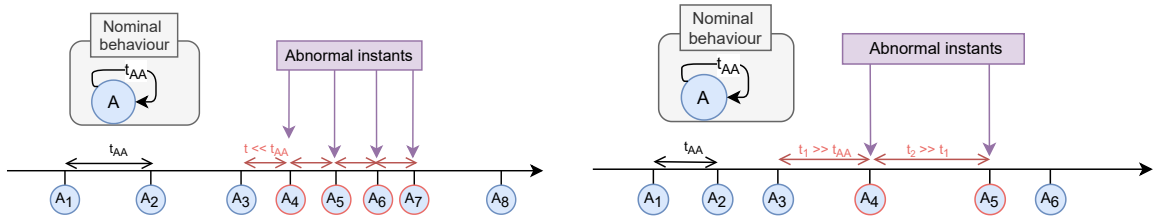
periodic_sequence_m. The scenario **periodic_sequence_m** is described by a nominal execution of periodic sequence of two event types (similar to **periodic_sequence_t**), along with a missing log anomaly. Here, the log $\mathtt{B}_3$ is missing, whereas it should have been triggered by the occurrence of $\mathtt{A}_3$. The expected time of arrival of $\mathtt{B}_3$ is labelled as abnormal. In this case, the sequence of event types is modified, so the anomaly can be observable by a sequential study. It is also observable by studying the time elapsed between logs.

xor_m. In the **xor_m** scenario, we suppose that a the nominal functioning involves a **XOR** relation, as per presented in **xor_o**. This time, the abnormal sequence is defined by the absence of both logs of B and of C after an occurrence of a log of A. The log $\mathtt{A}_3$ of Figure 4.3c illustrates this abnormal behaviour. Detecting the anomaly in this scenario is much more difficult than in the two previous ones. Indeed, in the previous scenarios, a unique path is presented as the unique acceptable nominal behaviour, and therefore the violations of the relationships are straightforward to detect. In this context, the nominal behaviour contains example of missing B and missing C, separately. The detection might become even more difficult if the distribution between B and C is unbalanced (p_{AB} far from 0.5). In this case, it might be confusing to distinguish the rare nominal path from the anomalies.

4.2.1.4 Group anomalies

We finally describe scenarios of multiple consecutive anomalies. We focus on a minimal scenario with a single event type, and propose two kinds of temporal group anomalies. In these scenarios, the anomaly of a log is propagated to the n following logs, either in a similar way, or with an amplification. We choose to label as abnormal the appearance instants of the logs with the propagated anomalies. We do not label the expected arrival time of these logs, since their might become irrelevant with the accumulation of temporal disturbances. These scenarios can be good approximations of the distribution anomalies presented in Section 2.3, since they can simulate an abnormal concentration of logs of an event type.

periodic_t_n. The nominal version of this scenario is the single periodic event type, presented for **periodic_t**. The group anomaly consists in a series of n consecutive logs that present the same temporal anomaly. In Figure 4.4a, the $n = 4$ consecutive logs $\mathtt{A}_4$, $\mathtt{A}_5$, $\mathtt{A}_6$ and $\mathtt{A}_7$ happen after a time t compared to their previous occurrence, where t is significantly smaller than the nominal periodicity t_{AA}. All the timestamps of these occurrences are labelled as abnormal. However, it becomes irrelevant to label the instant when the logs should have happened : for instance, the nominal oc-



(a) Temporal anomaly on a group of periodic logs, with a constant temporal error (**periodic_t_n**). The nominal scenario corresponds to the one of (**periodic_t**). The anomaly consists in a abnormal value of the periodicity (too soon or too late), which is evenly propagated on several consecutive logs. We label as abnormal the timestamps of all the logs that propagate the anomaly.

(b) Temporal anomaly on a group of periodic logs, with an amplification of the temporal error (**periodic_t^n**). The nominal scenario corresponds to the one of (**periodic_t**). The anomaly is an abnormal value of the periodicity, propagated on several consecutive logs, with an amplification (acceleration or deceleration). We label as abnormal the timestamps of all the logs that propagate the anomaly.

Figure 4.4 – Scenarios with group anomalies, i.e. anomalies occurring on several consecutive logs. The nominal behaviours are schematized in the grey boxes. The temporal axis contains both nominal and abnormal sequence execution. The purple arrow indicates the instants that we label as abnormal. These scenarios represent temporal anomalies in a single event type nominal scenarios, where the temporal anomaly is propagated to the following logs.

currence of time of A_7 is likely to be far after the actual occurrence time of A_8, a timestamp when the system is now returned to a nominal behaviour.

periodic_t^n Similarly, Figure 4.4b exhibits a scenario of temporal group anomalies, yet, with an amplification mechanism. This time, the anomaly propagated is amplified at each new log encountered. In the example, A_4 happens after A_3 with a time interval t_1 that is much greater than the expected t_{AA}. Thereafter, the log A_5 arrives with an even more important delay, since t_2 is not only significantly higher than t, but also than t_1. This scenario represents the phenomenon of a deceleration. Similarly to the previous scenario we label the timestamps of the actual abnormal logs as abnormal.

4.2.1.5 Composed scenarios

In all of these minimal scenarios that represent a single type of anomaly, we identified 4 different nominal cases : (i) a unique event type happening periodically, (ii) a sequence of 2 event types, (iii) a periodic sequence of 2 event types, (iv) a XOR scenario with 3 event types. Except for the sequence of 2 event types, which is only visible in one scenario (**sequence_t**), we propose to create 3 composed datasets that mix the different types of anomalies for each nominal scenario. The three new scenarios are described as follow :

- **periodic_c** is composed of the periodic anomaly **periodic_t**, the missing log anomaly **periodic_m** and the two group anomalies **periodic_t_n** and **periodic_t^n**;
- **periodic_sequence_c** is composed of the periodic anomaly **periodic_sequence_t**, the order violation anomaly **periodic_sequence_o**, and the missing log anomaly **periodic_sequence_m**;
- **xor_c** is composed of the order violation anomaly **xor_o**, and the missing log anomaly **xor_m**.

We first generate each scenario with the corresponding injected anomalies. We then add contex-

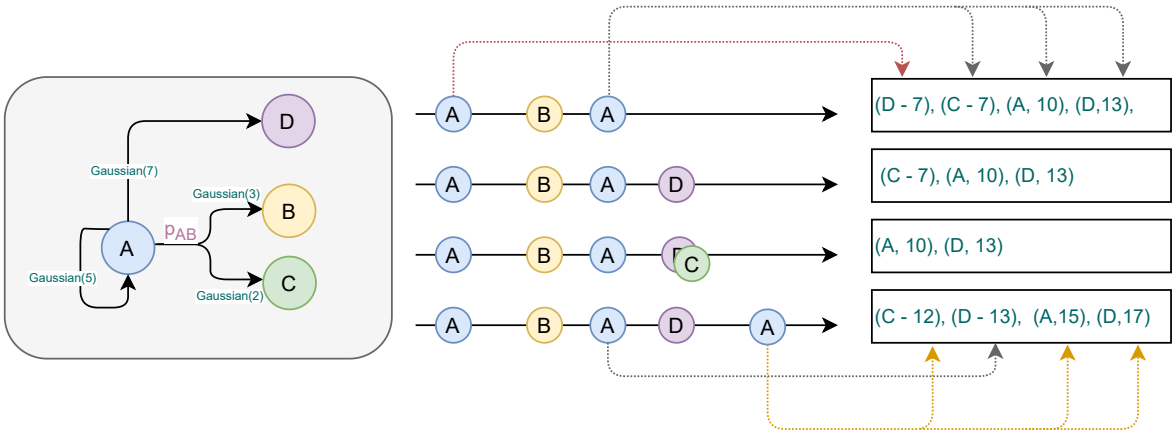


Figure 4.5 – Our dataset generator. Left : the pattern to generate. Nodes represent event types, and edges represent temporal transitions, with their associated distributions, e.g., Gaussian(mean). The edge between A and (B, C) is conditional : it represents a **XOR**, with the associated probability p_{AB}. Center : 4 consecutive steps of generation of logs. Left : the associated queues of future logs. Each generated log that has successors (i.e. A occurrences, in the center) adds future logs to the queue (dotted lines).

tual logs, i.e. logs that are not linked to the main sequence of interest, but happen in parallel and independently. These versions of the scenarios are closer to the situation of real datasets, where the anomalies are triggered in complex environments, with other event types generating logs in parallel.

4.2.2 Dataset generation

Based on these scenarios, we generate artificial datasets thanks to a data simulator we implemented, depicted in Figure 4.5. We first translate the nominal scenario into a graph representation (left-side of the figure), where each node represents an event type, and each edge is a generation relationship : the edge from A to D indicates that an occurrence of A triggers the generation of D. These relationships are temporized, to indicate the time interval separating the logs. More specifically, instead of simply labelling the edges with a numeric value of the time interval separating the logs, we propose to model the temporal relationships with the distribution of the time interval. For instance, in the figure, "Gaussian(7)" over the edge A - D indicates that the time separating the two logs follows a Gaussian distribution of mean 7. This enables to add noise, and irregular time separation. Hence, a Poisson distribution for an edge can symbolize a rare transition, while Gaussian distribution represent normalized relationships. Hence, depending on the chosen distribution, the creation of an edge involves the providing of the parameters of the distribution parameters (e.g. mean and variance for the Gaussian distribution).

Moreover, for a node A, there can be several outer edges. This means that the generation of A triggers the generation of several other event types. In Figure 4.5, A has 3 outer edges, meaning that each occurrence of A generates three new events. Eventually, we also allow each individual edge to be divided in several probabilistic edges, in order to represent the **XOR** mechanism : the sum of the probability of the probabilistic edges is 1, and when A occurs, one of the probabilistic edge (B or C) is randomly selected as a successor, accordingly with the probability distribution. With

this conditional mechanism, we also construct alternative paths to the nominal path, in order to represent anomalies : sequential anomalies are created with an alternative path that changes the order of the nodes traversed, while temporal anomalies are created by alternative edges, with the same nodes, but with different time distributions.

Finally, we authorize the nodes to be fictive, meaning that traversing these nodes does not trigger the generation of an event. These nodes can be used to model complex path, and especially, to represent the absence of a log. They can also be used to symbolize mechanisms such as the synchronization, i.e., two paths wait for each other to end, or even a parallel starting of path. When an abnormal path is added, it is marked as abnormal, and a temporal labelling strategy is specified.

To generate the datasets, we perform a traversing of the graph, with a Breadth-first search (BFS) mechanism. Hence, when a node is traversed, its successors are added to a temporal queue, with a corresponding generation time, defined by the corresponding edge. Nodes are traversed according to the queue order, with the associated generation time. Figure 4.5 also illustrates this iterative process, with successive representation of the generated logs (in the center) and the associated state of the queue (on the right side). At each step, the first node in the queue is added to the set of logs, and each of its successor is generated and placed into the queue, according to their timestamp, for later traversing. For instance, in the first step of the figure, the A log is added, and each of its successor are generated (select which node in the case of **XOR**, and select the timestamp), and added to the queue, ordered by timestamp (grey dotted lines).

This temporally ordered queue mechanism enables to generate the logs in the temporal order, and to easily manage the interruption of the generation. Indeed, to stop the generation, we provide a maximum timestamp T. We stop the generation when we traverse a log from the queue which timestamp t is greater than T. We also use this mechanism to execute the labelling strategy : when an abnormal path is traversed, the corresponding instant is added to the list of abnormal times.

For the artificial scenarios presented in this section, we simulate a time-frame of $T = 5000$ for each of the training, validation and testing sets, and inject 1% of anomalies in the validation and testing sets. For every scenario, we specify the corresponding nodes and edges (represented in the nominal behaviour, augmented with abnormal paths) by (i) providing a list of nodes, (ii) providing edges between these nodes, with the corresponding time and probability distributions. We propose 5 different versions of each minimal scenarios (with different times and probability distributions). As we shall observe in the result analysis, we paid attention to propose configurations with different difficulties : we introduce noise in some configurations by increasing the variance of the temporal distribution of edges, or create some hardly detectable anomalies (e.g. a slight temporal delay, with a nominal duration that is already poorly steady). We aim at measuring the robustness of the methods towards these challenges.

4.2.3 Detection results

We now evaluate the F1-scores of the different methods on the artificial scenarios. We first present the results of the minimal scenarios, then detail the F1-scores obtained for the composed scenarios.

periodic_t							
	D6	D7	D8	D9	D10	Mean	Std
S-DeepLog	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P-DeepLog	**0.96**	**0.92**	**0.94**	0.90	0.83	**0.91**	0.05
LogAnomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NoTIL-LSTM	0.94	0.65	0.90	**0.94**	0.83	0.85	0.12
NoTIL-AE	0.93	**0.92**	0.90	0.93	**0.86**	**0.91**	0.03
sequence_t							
	D21_at	D22_at	D23_at	D24_at	D25_at	Mean	Std
S-DeepLog	0.13	0.62	0.18	0.23	0.10	0.25	0.21
P-DeepLog	0.18	0.86	0.11	**0.78**	0.95	0.58	0.40
LogAnomaly	0.12	0.62	0.16	0.08	0.00	0.20	0.24
NoTIL-LSTM	0.16	**0.97**	**0.94**	0.57	**0.96**	0.72	0.36
NoTIL-AE	**0.38**	0.96	**0.94**	0.67	0.90	**0.77**	0.25
periodic_sequence_t							
	D21_t	D22_t	D23_t	D24_t	D25_t	Mean	Std
S-DeepLog	0.14	0.23	0.10	0.10	0.17	0.15	0.05
P-DeepLog	0.91	0.80	0.64	**0.98**	**0.39**	0.74	0.24
LogAnomaly	0.14	0.22	0.10	0.10	0.17	0.15	0.05
NoTIL-LSTM	**0.96**	**0.93**	**0.77**	0.96	0.00	0.72	0.41
NoTIL-AE	0.93	0.89	0.70	0.97	0.33	**0.76**	0.26

Table 4.1 – Best F1-score results (over 20 attempts) for the 5 understudied methods, on the temporal anomaly scenarios, namely **periodic_t**, **sequence_t** and **periodic_sequence_t**. Each scenario is associated to 5 generated datasets.

4.2.3.1 Minimal scenarios

We first present the result of the anomaly detection on the temporal anomalies, in Table 4.1. As aforementioned, S-DeepLog and LogAnomaly are not designed for any of these types of anomalies. Specifically, in the case of a single event type, **periodic_t**, these algorithms are meaningless : they presumably learn to reproduce the input as the output, and the abnormal samples are represented exactly as the normal ones. On the contrary, P-DeepLog and the two versions of NoTIL, NoTIL-LSTM and NoTIL-AE, are suited for the detection of temporal anomalies. P-DeepLog presents excellent results on the scenario **periodic_t**, since it is particularly designed to detect single event type anomalies. It however shows more variability, and struggles to treat properly the datasets of the 2-event-type scenarios that have important variances ($D21\_at, D23\_at, D23\_t, D25\_t$). NoTIL-LSTM presents a slighlty better robustness to the difficult scenarios, while NoTIL-AE seems to be very robust to these difficult scenarios.

Table 4.2 presents the results for the anomalies that concern violations of the order of the sequence of logs. Both S-DeepLog and LogAnomaly are especially designed for these types of anomaly, while P-DeepLog can only detect them when they imply a temporal irregularity at the event-type level. It can be the case for the **periodic_sequence_o** scenario : since the abnormal B happens before A, it happens sooner than expected, regarding the previous B. Nonetheless, for datasets involving an important variance in the B temporal relationships, namely $D22$, $D24$ and

xor_o							
	D1	D2	D3	D4	D5	Mean	Std
S-DeepLog	0.72	0.46	0.63	0.76	0.91	0.70	0.17
P-DeepLog	0.08	0.07	0.11	0.02	0.09	0.07	0.03
LogAnomaly	0.71	0.45	0.72	0.76	0.93	0.71	0.17
NoTIL-LSTM	0.00	0.92	0.00	0.00	0.16	0.22	0.40
NoTIL-AE	**0.97**	**0.97**	**0.96**	**0.99**	**0.99**	**0.98**	0.01
periodic_sequence_o							
	D21	D22	D23	D24	D25	Mean	Std
S-DeepLog	0.87	0.95	**0.94**	**0.75**	0.38	0.78	0.24
P-DeepLog	0.82	0.56	0.83	0.59	0.48	0.66	0.16
LogAnomaly	**0.89**	0.95	**0.94**	**0.75**	**0.71**	**0.85**	0.11
NoTIL-LSTM	0.84	0.65	0.85	0.35	0.66	0.67	0.20
NoTIL-AE	0.84	**0.98**	0.90	0.74	0.67	0.83	0.12

Table 4.2 – Best F1-score results (over 20 attempts) for the 5 understudied methods, on sequential anomaly scenarios involving violations of the sequence order, namely **xor_o** and **periodic_sequence_o**. Each scenario is associated to 5 generated datasets.

$D25$, the anomaly becomes hardly detectable. It seems that NoTIL-LSTM suffers from the same difficulty of learning the nominal temporal relationships, while NoTIL-AE is less sensitive to noise, and manages to reach the levels of LogAnomaly and S-DeepLog.

Concerning, the **xor_o** scenario, the abnormal pattern is not detectable by P-DeepLog : the event types individually cannot be aware that both B and C happened. The simple neural architecture of NoTIL-LSTM also fails in learning the complex nominal pattern, with the exception of the dataset $D2$, where the **XOR** is almost equiprobable and the temporal relations have very low variances. While LogAnomaly and S-DeepLog are perfectly designed to detect this type of anomaly, they sometimes suffer from temporal imprecision in the detection. Indeed, in Figure 4.2b, these methods detect the anomaly when the second log occurs (C_3). Then, the log C_3 is passed as input in the following predictions, according to the look-back h. Yet, this look-back refers to a number of log, instead of a time interval. As a consequence, if the logs are temporally spread, the error detection is propagated for an important time duration after the actual anomaly, leading to numerous false positives. Hence, the results are directly correlated to the temporal interval t_{AA}, with lower scores when t_{AA} is high. While the two methods have similar results, we notice that LogAnomaly has a generally better accuracy, which we attribute to its attention mechanism, that reinforce the model learning. Finally, NoTIL-AE is the only method to present systematically excellent results in this scenario, and is not impacted by either the noise of temporal relations, or the unbalance in the probability distribution.

The results of the scenarios of missing logs are presented in Table 4.3. Missing log anomalies can generally be described as sequential anomalies, which LogAnomaly and S-DeepLog are specially designed to detect. Nonetheless, like **periodic_t**, the scenario **periodic_m** contains a single event type, and the two methods cannot learn any specific pattern on this perfectly uniform dataset. On the contrary, P-DeepLog catches temporal differences between two consecutive occurrences : if a log is missing, the temporal difference becomes twice as high. The method however

xor_m							
	D1	D2	D3	D4	D5	Mean	Std
S-DeepLog	0.58	0.48	0.61	0.83	0.88	0.68	0.17
P-DeepLog	0.00	0.04	0.07	0.05	0.15	0.06	0.06
LogAnomaly	0.61	0.67	0.67	0.70	0.88	0.71	0.10
NoTIL-LSTM	0.00	0.00	0.00	0.00	0.84	0.17	0.38
NoTIL-AE	**1.00**	**1.00**	**0.89**	**0.71**	**0.93**	**0.91**	0.12
periodic_sequence_m							
	D21	D22	D23	D24	D25	Mean	Std
S-DeepLog	0.84	0.62	0.77	0.56	0.65	0.69	0.11
P-DeepLog	0.67	0.41	0.80	0.40	0.37	0.53	0.19
LogAnomaly	0.84	0.62	0.67	0.56	0.58	0.65	0.11
NoTIL-LSTM	**0.96**	**0.74**	**1.00**	0.42	0.79	0.78	0.23
NoTIL-AE	**0.96**	**0.74**	**1.00**	**0.92**	**0.90**	**0.90**	0.10
periodic_m							
	D11	D12	D13	D14	D15	Mean	Std
S-DeepLog	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P-DeepLog	0.97	0.80	0.38	0.97	0.77	0.78	0.24
LogAnomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NoTIL-LSTM	**1.00**	**1.00**	0.37	**1.00**	0.64	0.80	0.29
NoTIL-AE	**1.00**	**1.00**	**0.91**	**1.00**	**0.86**	**0.95**	0.07

Table 4.3 – Best F1-score results (over 20 attempts) for the 5 understudied methods, on the missing log anomaly scenarios, namely **xor_m**, **periodic_sequence_m** and **periodic_m**. Each scenario is associated to 5 generated datasets.

shows a temporal imprecision, since the anomaly is only detected once the log finally arrives : in Figure 4.3a, the timestamp of A_5 is detected as abnormal, instead of the timestamp of the expected A_4. Moreover, as explained before, the abnormally long time interval A_3 - A_5 is propagated as long as this time interval is passed as input. Hence, the results of P-DeepLog are directly correlated to the time t_{AA}, with low results when t_{AA} is high. NoTIL-LSTM perfectly retrieves the error of 3 of the 5 datasets. However, $D15$ and even more $D13$ contain important variances, making the nominal behaviour too complex for NoTIL-LSTM to learn. Only NoTIL-AE presents impressively high results for this scenario.

On the contrary, LogAnomaly and S-DeepLog are designed to detect both **xor_m** and **periodic_sequence_m** scenario. Yet, as for **xor_o**, these methods suffer from temporal imprecision when the temporal intervals involved are too high (**xor_m** with $D1$, $D2$, $D3$, **periodic_sequence_m** with $D22$, $D24$, $D25$). NoTIL-LSTM generally fails to learn the complex behaviour of **xor_m**, except for $D5$ that presents a perfectly equiprobable schema. On the contrary, it presents highly satisfactory results for the scenarios **periodic_sequence_m** and **periodic_m**, except for the datasets with important temporal variances ($D24, D13$). NoTIL-AE systematically presents satisfactory results, with impressive improvements compared to the other methods, regardless of the complexity of the dataset.

Finally, we present the result of the anomaly detection on the scenarios of multiple anoma-

periodic_t^n							
	D16	D17	D18	D19	D20	Average	Std
S-DeepLog	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P-DeepLog	**1.00**	0.45	0.96	**1.00**	0.93	0.87	0.24
LogAnomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NoTIL-LSTM	0.95	0.71	0.97	0.98	**0.95**	**0.91**	0.11
NoTIL-AE	0.98	**0.84**	**0.98**	0.78	**0.95**	**0.91**	0.09
periodic_t_n							
	D6_n	D7_n	D8_n	D9_n	D10_n	Average	Std
S-DeepLog	0.00	0.00	0.00	0.00	0.00	0.00	0.00
P-DeepLog	**1.00**	**1.00**	**1.00**	**1.00**	**0.88**	**0.98**	0.05
LogAnomaly	0.00	0.00	0.00	0.00	0.00	0.00	0.00
NoTIL-LSTM	0.97	0.88	**1.00**	0.88	0.87	0.92	0.06
NoTIL-AE	0.97	0.88	0.95	0.90	0.87	0.91	0.04

Table 4.4 – Best F1-score results (over 20 attempts) for the 5 understudied methods, on the multiple temporal anomaly scenarios, namely **periodic_t^n** (noted periodic_tn̂) and **periodic_t_n**. Each scenario generates 5 datasets.

lies in Table 4.4. As for the **periodic_t** scenario, these scenarios involve only one event type, and S-DeepLog and LogAnomaly are simply not designed for this detection. On the contrary, the two versions of NoTIL as well as P-DeepLog present globally and equivalently excellent results, generally with scores superior to 90%. On the one hand, P-DeepLog is perfectly designed for these cases, since it detects abnormal time interval separating the logs of the unique event type. The labelling strategy also benefits the method, since only the log presenting an abnormal time distance to their predecessor are labelled as abnormal. For the **periodic_t^n** scenario, NoTIL versions systematically reaches the performance of P-DeepLog, and even drastically surpasses it for the difficult dataset $D17$, where (i) the nominal behaviour is unsteady, with a quite important temporal variance, (ii) the difference between the nominal interval and the first occurrences of the anomaly (beginning of deceleration) is small. NoTIL-AE is especially capable of capturing the nuances of this complex scenario. The **periodic_t_n** scenario presents a slight degradation compared to the almost perfect results of P-DeepLog (up to 10%), yet still represents a strongly reliable detection, with results around 90%. Indeed, compared to the clear-cut between the anomalies and normal data in the acceleration scenarios, the difference is surely less blatant in this scenario.

In conclusion, NoTIL-AE is often the best solution, with some impressive improvements compared to the state-of-the-art propositions (+26% for **xor_o**, +21% for **periodic_sequence_m**, +20% for **xor_m**, + 17% for **periodic_m**). Moreover, when NoTIL-AE does not have the absolute best results, it is always extremely close to the best one, which shows its robustness across the scenarios. Especially, NoTIL-AE provides a unique answer to the missing log scenarios, with steady and high results, compared to the mitigated results of the state-of-the-art methods.

We generally observed that P-DeepLog manages to catch, apart from the temporal anomalies it is designed for, most of the sequential anomalies. Indeed, the modification of the sequential order (by inversion, addition or when logs are missing) often has a temporal signature in one of the event type. Yet, since this method is not designed for such a detection, the moment of detection is

	periodic_c		periodic_sequence_c		xor_c	
	simple	w_context	simple	w_context	simple	w_context
S-DeepLog	0.00	0.26	0.40	0.24	0.82	0.15
P-DeepLog	**0.95**	0.30	0.41	0.37	0.30	0.28
LogAnomaly	0.00	0.29	0.40	0.41	0.88	0.26
NoTIL-LSTM	0.82	**0.51**	0.63	0.19	0.08	0.24
NoTIL-AE	0.82	0.50	**0.91**	**0.77**	**0.95**	**0.48**

Table 4.5 – Best F1-score results (over 20 attempts) for the 5 understudied methods, on the composed scenario (simple columns) and with contextual logs added (w_context columns).

often inaccurate. Moreover, P-DeepLog is inefficient for modelling complex relationships between different event types, such as for the **XOR** scenario, resulting in extremely low detection results.

Generally speaking, LogAnomaly presents higher results than S-DeepLog on complex nominal behaviours, like the **XOR** scenario, since the attention mechanism is able to catch sophisticated sequential correlations between event types. Yet, on simpler scenarios, the model becomes more difficult to train, as it might be too sophisticated.

NoTIL-AE and NoTIL-LSTM present similar behaviours, with NoTIL-AE being capable of catching more complex anomalies. Nonetheless, the difference in the results is significant : in the **XOR** scenarios, NoTIL-LSTM completely plummets, since it did not manage to learn the nominal behaviour. We believe that its LSTM architecture is too simple to model the complex relationships involved in a **XOR**. NoTIL-AE instead is temporally accurate, and steady on the different datasets. We attribute this success to (i) the dimensionality modification process, that manages to extract the relevant features out of the set of event types, (ii) the multiple aggregation performed to calculate the final error on a single interval, which smooths the errors from multiple bigger temporal intervals, and provides a robust aggregation of the prediction error. We conclude from our experiments on the minimal scenarios that NoTIL-AE is the only solution to be highly recommendable to catch all the types of anomalies we have identified.

4.2.3.2 Composed scenarios

Table 4.5 presents the F1-scores of the methods for the aforementioned composed datasets. The simple scenarios exhibit results in line with the previous study. For the **periodic_c** scenario, which combines different types of anomalies on a single events, S-DeepLog and LogAnomaly are not designed for any of these anomalies, while P-DeepLog still presents high results. The two methods of NoTIL are slightly impacted by the complexity of the combined scenario.

Concerning the **periodic_sequence_c** scenario, in its basic version (column Simple), the three state-of-the-art methods present mitigated results, since the scenario gathers both sequential and temporal anomalies. Hence, S-DeepLog and LogAnomaly can detect both the order violation anomalies and the missing logs, yet, fail in detecting the periodic anomalies, resulting in disappointing results, around 40%. Similarly, P-DeepLog is suitable for detecting temporal anomalies, and can also catch sequential anomalies that present disruption in the nominal temporal behaviour. P-DeepLog can notably detect missing logs and orders violation, yet with a degradation of the temporal precision of the detection, resulting in overall mitigated results as well. The two ver-

sions of NoTIL present results that are coherent with those of the minimal scenarios, with overall high results. Especially, NoTIL-AE reaches the excellent score of 91% for this composed scenario, an improvement of 50% compared to the state-of-the-art results.

Finally, the **xor_c** scenario also illustrates the result of the individual scenarios : LogAnomaly, S-DeepLog and NoTIL-AE are especially suitable for the missing log and order violation detection, with a clear advantage for NoTIL-AE, thanks to a more precise temporal detection. On the contrary, NoTIL-LSTM fails in learning the complex model of the nominal beahviour. P-DeepLog also presents low results, since the XOR errors are not visible when event types are treated independently.

For each scenario, the column "w_context" outlines the results obtained, when logs with other event types are added, to create more realistic scenarios. Generally speaking, we observe that the addition of new event types greatly diminishes the results of the detection; indeed, the dimension of the nominal model to be learned is increased, and numerous new relationships need to be trained. It seems especially difficult for most of the methods to learn that the added event types are actually independent from the sequence of interest.

In **periodic_c** the three reliable methods, namely P-DeepLog and the NoTIL versions, are highly impacted by the presence of contextual logs. Nonetheless, only the versions of NoTIL still present reliable results, while P-DeepLog results plummet to 30%. Our analysis of the results showed that : (i) since the event types are processed independently, the nominal behaviour learning of the sequence of interest is not polluted by the new logs, (ii) however, the research of the detection threshold τ is common for all the event types, including the contextual ones, that do not burry any marker of the anomaly; as a result, applying the threshold detection to these event types triggers many false positives. This phenomenon is less visible on the other methods, where the error on each event type is averaged and compared to a global error threshold. Finally, it is worth noticing that both S-DeepLog and LogAnomaly benefit from the addition of new event types. Indeed, the contextual event types, when they are temporally regular, can act as temporal markers within the sequence of logs, and help to add the temporal dimension to the data. Yet, the results are still poorly reliable, since not all the contextual event types are temporally regular, which generates difficulties in learning the nominal pattern.

For both the **periodic_sequence_c** and **xor_c** scenario, the already-low scores are only slightly impacted by the addition of contextual event types. Yet, NoTIL-LSTM (for **periodic_sequence_c**), S-DeepLog and LogAnomaly (for **xor_c**), which present reliable results on the simple version, have their result plunged to the levels of the unsuited methods, due to the increase of complexity of the models. In both cases, NoTIL-AE establishes itself as the only reliable method, with impressive improvements compared to the other methods (+36% for **periodic_sequence_c**, and +20% for **xor_c**).

4.3 Real-world logs assessment

We now perform an evaluation of the methods on real-world data. In this section, we first describe the log datasets along with their associated anomalies. We explicit the adaptation of the protocol when required. Finally, we exhibit and comment the results of the anomaly detection task on these datasets for the different understudied methods.

4.3.1 Dataset presentation

We first present the datasets and the observable anomalies. We specify the few protocol adaptations required to perform the anomaly detection, due to the labelling, or to the huge size of data.

The OpenStack dataset Openstack logs monitors the creation and use of virtual machine (VM) instances. The authors of DeepLog generated and made available their own **Openstack** dataset with injected performance anomalies. The dataset contains 207 820 logs, and 47 event types, among which 23 are implied in a sequence (a VM instance). 4 sequences contain temporal anomalies, in the same transition time, for a global rate of 0.7% of abnormal sequences. These anomalies can be described as performance anomalies : a slow down in temporal transition from one event type to another. This case perfectly matches our minimal scenario **periodic_sequence_t**. As presented in Appendix B.1, the conversion between sequence-level labels and temporal detection can be performed in two ways : either we are manually able to identify the timestamp of the abnormal moments in the sequence (and the problem becomes that of equation B.3), or we need to split the logs by sequence, perform the temporal anomaly detection on each sample individually, and aggregate the temporal detection to the full sequence (as described by equation B.4). For the OpenStack dataset, due to the very straightforward description of the anomalies (exactly similar and few), we are able to detect the two logs involved in the performance anomaly. We then applied our labelling strategy, similarly to that of **periodic_sequence_t**.

The BGL dataset The **BGL** dataset contains logs from a super computer, monitoring the execution of the system. The dataset is composed of more than 4.7 million logs, with around 500 different event types. In this case, no sequence identifier is defined. The logs are rather labelled individually as failure, with a provided type of failure indicator, or as normal. This study is challenging, since some log messages contain, in their content part, terms such as "Error" or "Failure", and are however labelled as normal. In this case, we easily understand the need for a sophisticated deep learning method. While S-DeepLog and LogAnomaly are reported to treat efficiently this dataset (Meng et al., 2019), we are interesting in analyzing the types of anomalies they are able to detect, and the ones they fail to alert on.

A first challenge of applying anomaly detection to this unique set of logs consists in extracting the training, validation and testing subsets. Indeed, we need to be able to retrieve a training set with only nominal examples. Cutting the time into three consecutive temporal frames, while insuring the training set is anomaly-free, can only result in a too small training sample. Hence we rather propose to cut the dataset into a multitude of temporal segments. To do so, we define a maximum length between two consecutive logs of the same segment, t_{max}. In other words, if the temporal interval separating two consecutive logs is greater than t_{max}, we split the logs into two segments. The segments that do not contain any abnormal logs are eligible to be added to the training set.

On each segment, the general data construction for the prediction is built independently. We illustrate this principle with the log sequence L_1, L_2, L_3, L_4, L_5, L_6, L_7. For simplicity purpose, we take the case of DeepLog, that models the sequence of event types, yet, this idea is the same for all the methods. In this case, we assume that all the temporal intervals separating consecutive logs are inferior to t_{max}, except for L_4 - L_5. Hence, we cut the sequence into the subsequences

$s_1 = (L_1, L_2, L_3, L_4)$ and $s_2 = (L_5, L_6, L_7)$. If the look-back $h = 2$, then the sequence s_1 generates the input (L_1, L_2), associated to the output L_3, and the input (L_2, L_3), associated to the output L_4, and s_2 generates the input (L_5, L_6) associated to the output L_7. None of the intermediate input - output relation will be generated (e.g., the input (L_3, L_4) with output L_5). Once the sliding window mechanism is settled, we can concatenate different samples that are not necessarily consecutive into the training algorithm.

Besides, due to the important number of log messages, we observed that both DeepLog and LogAnomaly, which treat the dataset log by log, could not scale to this huge size dataset. The two versions of NoTIL do not encounter such an issue, since their event count representation can aggregate the data. Actually, the size of the window w (and of the decay δ) can be used as a lever to balance between the temporal accuracy requirements and the scalability to huge datasets. Its competitors do not propose such a flexible adaptation of the data representation. Moreover, our experiments showed that the segment cutting makes it difficult to ensure that the anomalies are represented evenly in the validation and the testing set. Yet, an even representation is desirable to learn a detection threshold, during the validation phase, that is applicable to the testing set. In this case, we rather directly perform the optimization of the threshold on the testing set.

To cope with these issues and still provide a fair evaluation, we slightly modify the general evaluation protocol. We propose the following step :

1. Cut the logs into temporal segments, separating normal and abnormal samples, with the associated type of anomaly;
2. Train on a subset of the available normal segments;
3. For each type of anomalies :
   (a) Optimize the threshold on a testing set, composed of a subset of segments that contain the anomaly;
   (b) Retrieve the best anomaly score obtained on this subset.

This way, we accelerate the training process, and evaluate only on the anomalies of interest. Moreover, distinguishing the types of anomaly upstream enables to evaluate the performance of the anomaly detectors on each type of anomaly.

Anomaly on the OBIS We study an anomaly detected on an application of the OBIS. A specific event type is associated to this anomaly, and can therefore be used as a temporal marker of the anomaly. The detection of this easily identifiable anomalies can be used to assess the ability of the models to recognize the pattern that generate the anomalies.

4.3.2 Detection results

We present the results obtained on the real-world datasets. The results of the OpenStack dataset are presented in Table 4.6, line “Raw”. The dataset only contains temporal anomalies. For this reason neither S-DeepLog nor LogAnomaly are designed for this use case. Moreover, P-DeepLog presents very disappointing results for this dataset. Indeed, during the validation phase, P-DeepLog tries to learn a common prediction error threshold, whatever the event type is. This common threshold is similarly used for the anomaly detection on each of the event types. Yet,

OpenStack	S-DeepLog	P-DeepLog	LogAnomaly	NoTIL-LSTM	NoTIL-AE
Raw	0.075	0.078	0.076	**0.756**	0.646
No_context	0.075	0.230	0.100	0.606	**0.861**

Table 4.6 – F1-scores obtained on the detection of the four temporal anomalies of OpenStack, on raw data (line Raw) and after removing the contextual logs (line No_context), e.g. logs that are not associated to a sequence identifier.

since more than half of the event types are not implied inside a sequence execution (they do not contain a sequence identifier) and do not bear any signature of the anomalies, applying a threshold on these event types leads to numerous false positives. On the contrary, the two versions of NoTIL present already satisfactory results, despite the complexity of the nominal behaviour to be trained.

To confirm the hypothesis that the low results of P-DeepLog are linked to the contextual logs, we propose an alternative version of the dataset (line "No_context"), which only contains the 23 event types involved in a sequence execution, by removing all the logs that do not contain a sequence identifier. This process is however not innocuous : it requires domain knowledge to identify the sequence identifier in the logs, and it eliminates 61% of the logs. Yet, it is not sufficient to improve the results of P-DeepLog : one of the event type occurs twice in each instance, hence why there are two nominal durations between consecutive occurrences : (a) a short one, between occurrences of the same instance, (b) a much longer one between distinct instances. PDL fails in learning this nominal behaviour and triggers many false positives. On the contrary, NoTIL-AE results are enhanced by this simplification process. The removal however impacts slightly the performance of NoTIL-LSTM, which now lacks some robustness in the learning of the nominal behaviour. Our analysis of the two versions of NoTIL results highlighted some false positive cases, that seem reasonable. For instance, a rare event type appears at an unusually high frequency in the testing dataset which is legitimately detected as abnormal by NoTIL.

We present the anomaly results obtained, for each of the 10 main types of anomalies of the BGL dataset (Oliner and Stearley, 2007). Firstly, we observed that all the anomalies could be mainly identified by the fact that they correspond to specific event types. As a result, the data creation process excludes these event types from the training set, and therefore, no individual training can be performed on it. For this reason, P-DeepLog, which models each event type individually, is not suitable for this detection. On the contrary, the other methods have a specific feature entry that designates the unseen logs. Hence, new event types are all gathered in this representation feature.

While the anomalies of these datasets can be identified through the presence of a new event type, this rule only is not sufficient to obtain a reliable anomaly detection solution. Indeed, due to process of separating normal and abnormal data to generate an anomaly-free training set, we excluded whole segments of the training phase, which contain event types that end up to be absent from the training set. These missing event types are not only the ones matching the anomalies. As a result, the simple detection that consists in alerting when a new event type occurs generates many false positives. Hence, the model should be able to extract meaningful patterns that represent the anomalies from the rest of the observed event types.

Table 4.7 presents the F1 scores, the precisions and the recalls of each method, performed on the selected anomalies. We generally observe overall satisfactory results. Especially, S-DeepLog

Anomaly	S-Deeplog			LogAnomaly			NoTIL-LSTM			NoTIL-AE		
	F1	Pr	Re	F1	Pr	Re	F1	Pr	Re	F1	Pr	Re
KERNDTLB	**0.81**	**0.68**	**1.00**	0.81	0.68	**1.00**	0.74	0.62	0.94	0.79	0.65	**1.00**
KERNSTOR	0.56	0.33	**1.00**	0.56	0.33	**1.00**	**0.68**	**0.55**	0.88	0.68	0.52	0.88
APPSEV	0.88	0.83	0.96	0.90	0.88	0.96	0.94	0.90	0.99	**1.00**	**0.99**	**1.00**
KERNMNTF	0.52	0.52	0.52	0.52	0.52	0.52	0.63	0.48	1.00	**0.99**	**1.00**	**0.99**
KERNTERM	0.48	0.44	0.97	0.50	0.46	**1.00**	**0.66**	**0.54**	**1.00**	**0.66**	**0.54**	**1.00**
KERNREC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.99	0.99	**1.00**	**1.00**	**1.00**	**1.00**
APPREAD	0.73	0.73	0.73	0.73	0.73	0.73	0.78	0.61	**1.00**	**1.00**	**1.00**	**1.00**
KERNTSP	0.55	0.48	0.68	0.60	0.51	0.78	0.29	0.17	**1.00**	**0.86**	**0.80**	0.98
APPRES	**1.00**	**0.99**	**1.00**	**1.00**	**0.99**	**1.00**	0.94	0.89	0.99	0.95	0.92	0.99
APPUNAV	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Mean_w	0.74	0.62	0.94	0.74	0.63	0.94	0.75	0.64	0.95	**0.83**	**0.73**	**0.97**

Table 4.7 – Results of the detection of the 10 most frequent anomalies in BGL for the understudied methods, with the F1-scores (F1), the precisions (Pr) and the recalls (Re).

and LogAnomaly systematically present very similar results, while NoTIL-LSTM and NoTIL-AE are also very similar. Some type of anomalies, like APPUNAV and KERNREC are easily detectable, because they do not enter in conflict with other new event types when they occur. Other anomalies commonly present much lower detection result, like KERNTERM. As a general observation, it seems that, in most of the cases, the recall is favoured compared to the precision score. This characteristics is highly desirable in the context of critical anomalies, where the user would rather detect too much anomalies than missing a single one. However, this assumption is only verified if the precision is still high, like for NoTIL-AE with the anomalies APPRES and APPSEV. Otherwise, the user might loose confidence in the prediction, and the too important number of false positives might make the analysis of anomalies difficult for the user.

In most of the cases, the two versions of NoTIL reach the level of the state-of-the-art propositions. Yet, for some anomalies that are not properly treated by neither S-DeepLog nor LogAnomaly, the versions of NoTIL sometimes show impressive improvements of the F1-measure : +48% for KERNMNTF, +31% for KERNTSP, +11% for KERNSTOR and KERNTERM. Hence, more than the slight improvement in the overall score, our methods are able to catch some anomalies that were not treated by the sequential approach of the two understudied state-of-the-art reference.

Table **??** exhibits the results of the anomaly detection on the OBIS study. Except for P-DeepLog, all the methods present satisfactory results, with an advantage for NoTIL-LSTM and NoTIL-AE regarding the F1-score, while LogAnomaly is slightly better than S-DeepLog. Especially, LogAnomaly, has a satisfactory precision score, which means that the methods generate few false positives, while the other methods generate a few more. On the other hand, the two versions of NoTIL present extremely high recall scores, meaning that they catch almost all the anomalies, compared to LogAnomaly, and especially S-DeepLog. Since errors might be fundamental to detect, the recall is generally more valued than the precision : we would rather detect false alarms than miss anomalies. Of course, too many false positives generate a lack of confidence on the anomaly detected. Yet, it is not the case of our two NoTIL versions, which still present high precision scores, and should therefore be favoured.

We also take interest in the execution times, which measure the operational efficiency of the anomaly detectors. For all the methods, the training phase is between 20 to 30 times as long as the testing and validation phase. While the size of sets can partially justifies this gap, the main explanation might be that the validation and testing phases are linear in complexity : they only consist in applying the weighted learned model, while the training phase contains an optimization involving back propagation on large and high dimensional matrices. This is a classical remark for the deep learning method, which is acceptable for the industrial use, since the model is generally trained once, and offline, and can be efficiently applied, once trained, in online fashion.

More interestingly, we observe a huge difference between our methods and the three other in terms of execution times, over the whole dataset. Our methods are impressively more efficient, because they represent the logs in an aggregated way, which summarizes many logs into a unique event count vector. This highlights the important modulation power of NoTIL, which can adapt to the temporal concentration of logs by adapting both the window size w and the window decay δ. We notice that P-DeepLog trains a different model for each of the event types, which is poorly efficient, even though each model is of dimension 1, and can be trained quickly. S-DeepLog suffers from the high dimension of the data, added to the high input dimension (the number of logs). LogAnomaly has the same constraints, yet, it also needs to train the attention weights. The higher complexity of the network explains the higher execution time — the same remark is applicable between NoTIL-LSTM and NoTIL-AE.

4.4 Main conclusions on the evaluation study

In this chapter, we performed a comprehensive assessment of our method and compared it to the state-of-the-art reference S-DeepLog, P-DeepLog and LogAnomaly. Firstly, we proposed to generate artificial scenarios with (i) minimal examples, (ii) composed datasets, (iii) datasets with contextual logs. We then experimented on real-world log datasets, containing both sequential and temporal anomalies.

For the minimal examples, the general theoretical deductions are generally observable : S-DeepLog and LogAnomaly are able to detect sequential anomalies, and fail in retrieving temporal ones; P-DeepLog can detect temporal anomalies, and sequential anomalies that impact the temporal behaviour of the event types individually; the two versions of NoTIL are designed to detect all types of temporal and sequential anomalies.

More specifically, S-DeepLog and LogAnomaly can detect order violation anomalies, and missing log anomalies, when several event types are involved. In the case of a single event type, the data representation becomes meaningless, and the methods retrieve a null score. Moreover, the absence of consideration for the time elapsed between logs in the data representation makes the methods unsuitable for detecting purely temporal anomalies. Adding contextual logs can slightly improve this issue : the contextual logs constitute temporal markers and can introduce the temporal dimension to the representation. Yet, our experiments on both the simulated data and the OpenStack dataset showed that this is not sufficient for S-DeepLog and LogAnomaly to be able to properly model the temporal relations between event types, and therefore, the methods cannot catch temporal anomalies. Nonetheless, some reference log datasets only contain sequential anomalies, such as BGL, which S-DeepLog and LogAnomaly are designed to catch. Their efficiency

on these specific scenarios, which are however common, justifies the popularity of these methods.

On the contrary, P-DeepLog is designed to model temporal relations for each event type individually. Consequently, the method is particularly efficient for detecting temporal anomalies on unique event types. When several types are involved, the accuracy of P-DeepLog depends on the visibility of the temporal anomaly in each event type separately. This remark can be extended to sequential anomalies, which often also have a temporal signature. Yet, for sequential anomalies with complex patterns involving several event types (like the **XOR** situation), P-DeepLog fails in modelling the nominal behaviour, which explains its limited results in the version of OpenStack without contextual logs. Besides, P-DeepLog is impacted by the presence of contextual logs, since it tries to learn a common detection threshold, even for the event types not involved in the sequence of interest, as for the originial OpenStack dataset.

Finally, we showed that the two versions of NoTIL were able to catch both sequential and temporal anomalies. Especially, NoTIL-AE seems to be able to catch more complex patterns, and systematically reaches the level of the state-of-the-art methods on their minimal scenarios of predilection, while enabling the discovery of new types of anomaly. The real-world datasets confirmed the universality of our solution, since the two versions of NoTIL perform the best detection results on all the understudied scenario, with some impressive improvements.

5 Part conclusion

In this part, we presented NoTIL (**?**), a new anomaly detection method to alert on anomalies in system logs. NoTIL is based on the detection of novelties, which alleviates the imbalance of data and the scarcity of labelled data, and enables the detection of new anomalies, which are not necessarily outliers. Moreover, NoTIL adopts a data representation that takes into account the time elapsed between logs, by counting the number of event types in temporal windows. Finally, our method learns the nominal behaviour of logs through the training of a prediction task, which we choose to rely on deep learning. This way, NoTIL is able to models complex relationships between the event types. We proposed two different versions of NoTIL, NoTIL-LSTM and NoTIL-AE, based on two different prediction tasks : (i) forecasting with an LSTM, (ii) reconstructing with an auto-encoder.

We compare NoTIL to the state-of-the-art references in novelty detection, DeepLog and LogAnomaly. We proposed a new typology to define the anomalies in accordance with the anomalies observed in the real datasets. Our evaluation work compares the existing methods to the two versions of NoTIL on both simulated data, based on the topology, and real-world data. Our experiments show that the data representation chosen for NoTIL is the only one able to detect all of the types of anomalies we identified, including sequential anomalies, with order violation or missing logs, temporal anomalies, and distribution anomalies. NoTIL-AE tends to provide more robust results than NoTIL-LSTM. Our study of the real-world datasets confirmed the robustness of NoTIL. The two versions of NoTIL provides important result improvement on the OpenStack dataset, and on some anomalies of BGL that were not properly treated by the state-of-the-art methods.

General conclusion

To conclude this manuscript, we first summarize the different contributions of our work, and describe the main conclusions of our experimental evaluations. We then provide an insight on the field of possible applications for our method. Finally, we describe the perspective of improvement of our work, both in terms of contribution and evaluation.

Summary of the contributions

In this thesis, we propose an end-to-end solution to detect information system anomalies based on the detection of anomalies in the execution logs. Our proposition includes (i) the structure inference of logs in order to infer their structure, retrieve exploitable information, and extract a data representation for the anomaly detection task, (ii) a novelty detection method that is adapted to capture a wide variety of types of anomalies, described by a new topology.

Our first contribution consists in providing a new and complete method to pre-process the logs, from raw unstructured data, to structured and exploitable features. We proposed METI*N*G, a log parsing method based on the frequent pattern mining assumption. With this flexible syntactic representation of logs, METI*N*G does not introduce any strong assumption. METI*N*G is also parametric, which enables the method to modulate, and to adapt to the important heterogeneity of the reference datasets. Our experimental study shows that METI*N*G has overall more accurate results in the parsing of the reference datasets than all the state-of-the-art methods. Especially, it performs quite impressive improvements on some datasets that were not properly treated by the existing methods. We also studied the sensitivity of our method regarding its hyper-parameters. We evaluated that METI*N*G is highly sensitive to the variations of its hyper-parameters, which has two main consequences : (i) it can propose a multitude of different variants, and can therefore adapt to a large variety of datasets, compared to the existing methods, and (ii) the optimization of the hyper-parameter is a crucial question. As such, we detailed an algorithmic solution to label a small portion of logs. We evaluated the algorithmic efficiency of METI*N*G and found out that it is faster than the most popular state-of-the-art methods. We experimentally demonstrated the feasibility of the online version of METI*N*G. Moreover, we described a template extraction method through a detailed example, illustrating the challenges of this task. Our proposition not only separates the fix and variable parts in the logs, but also identifies the types of variables, and provides some general statistics on their values. We provide an example of the application of this method on a reference dataset, which demonstrates the feasibility and exhibits the format of the obtained result.

Finally, we extend our complete method to the task of stemming, in the context of text mining. We presented RFreeStem, a multilanguage stemmer based on the mining of a corpus and detailed the modifications necessary to adapt to the paradigms of this new context. We then evaluated our proposition in terms of improvement of the accuracy of a text mining task, and compared to the state-of-the-art reference, the Porter stemmer. Our evaluation shows that RFreeStem is systematically better than Porter, and systematically improves the result of the text mining task, with some important improvements on the flexional languages (French and German). RFreeStem also proposes a stemming solution for the urdu language, which does not benefit from a version of Porter stemmer.

Our second contribution concerns the detection of anomalies on a structured dataset. We presented NoTIL, a novelty detection method based on the learning on an intermediate task to detect anomalies as violations of the nominal behaviour. NoTIL opts for a temporal event count representation which is able to represent the temporal aspect of log dataset, and to catch temporal irregularities in logs. For the intermediate prediction task, we propose two different deep learning models : an LSTM to predict the next event count, and an auto-encoder to reconstruct a matrix of event counts. To evaluate NoTIL, we defined a new topology of the anomalies observable in logs, and implemented a simulator to generate fictive datasets. We also evaluated our method on real-world datasets, and compare it to the most up-to-date novelty detection approaches, namely DeepLog and LogAnomaly. We observed that the two versions of NoTIL are the only methods able to catch all the types of anomalies, due to the flexible temporal representation we opted for. Especially, the auto-encoder version presents highly accurate and steady results.

Application domain of our approach

The main context of this manuscript concerns the detection of anomalies on information system logs, with both a log parsing method, and a detection algorithm applied on the event types. We applied these methods to real-world datasets : publicly available datasets as well as the logs of the information system of aircrafts. Yet, we only studied Information System logs. Other types of logs could enter our field of application, with only some slight modifications (e.g., connection logs, web logs... (Masseglia et al., 2000)).

Moreover, we identified an opportunity to extend our log parsing approach to the context of stemming : instead of finding the structure of logs, by identifying the fix words, we study the structure of words, and identify the fix letters. Even though our approach was not designed for this context, and with only some slight adaptation efforts, we show that RFreeStem is a valid solution and perform an accurate stemming; indeed, mining the n-grams of textual data is an efficient way to retrieve their structure. We could also imagine applying this method to other context, such as the structure inference of semi-structured data or text compression (Nestorov et al., 1997; Laur and Baril, 2004; Witten, 2004).

Finally, our method for the log anomaly detection can easily be generalized. Indeed, the input information consists in a couple (timestamp, event type). Actually, not only the logs can be represented by this couple, but any event-based context. For instance, our method could be used to detect anomalies in connected vehicles (Bridgelall and Tolliver, 2020) or medical data events (Ukil et al., 2016).

Perspectives and future work

Firstly, for the log parsing task, we will study an automated way to set the hyper-parameters, as part of the mining method, instead of requiring a semi-manual labelling task. We also wish to generalize the evaluation framework, especially by (i) comparing the results of the online functioning to other methods, and on more datasets, or (ii) comparing the outputs of the template extraction to both the labels of LogPAI, and the results of the other methods. Moreover, we could enrich our assessment with an evaluation of the effect of the log parsing as a preprocessing step of a log mining task, such as the anomaly detection. It would also be interesting to study the impact of the log parsing on extracted features for the anomaly detection task : knowing which parsing outputs are relevant for the anomaly detection, in terms of groups and variable parts extracted, remains an open question.

Concerning our proposition for the extension of METI*N*G to the context of stemming in text mining, we aim at studying different scoring functions for the n-grams : we could adapt the scoring function to the syntactic specificites of languages. We could also introduce an asymmetry between the first and last n-grams, since the prefixes are more likely bear meaningful information than the suffixes (Majumder et al., 2007). Our evaluation framework could be enhanced by comparing to other stemming or lemmatization methods. We would also like to experiment on different text mining tasks; especially, we want to evaluate our method on information retrieval tasks, for which Porter is especially designed. We want to assess the ability of our method to adapt to the requirements of different tasks. Moreover, since RFreeStem is multi-language, we could apply our stemmer to the task of cross-language information retrieval.

As far as the anomaly detection is concerned, we aim at experimenting a wider set of neural architecture for the prediction task, like a CNN or a GAN architecture, with different associated prediction tasks. We are especially interested in inserting an attention mechanism to our framework. Indeed, this could improve the interpretability of the prediction models, which is an acknowledged drawback of black-box algorithms like neural networks. Moreover, our evaluation framework could be improved by adding other deep learning competitors, and studying other real-world datasets.

Besides, we identified an important challenge that we aim at studying : the integration of the parameter values to the anomaly detection solution. Once extracted, the parameter values form time series that can be exploited for traditional anomaly detection in time series (Ben Kraiem et al., 2019; Clément et al., 2020; Archimbaud et al., 2021). Deeplog includes the parameter values in its anomaly detection, however, in an independent way. Since each event type has a different set of parameter values, the alignment of the parameter spaces is challenging. Instead, we are looking for a solution to incorporate the parameters to our existing anomaly detection method.

Beyond anomaly detection, critical information systems require explanation for the detected anomalies. If the output of an anomaly detection method can be interpreted, this method can provide more detailed guidance for the root cause analysis of investigated failures. The interpretation also provides a way to improve the transparency of the trained model. This transparency reinforces the trust of the users in the results, which can facilitate the industrialization of machine learning tools. As a result, in the longer term, we are interested in investigating the interpretability of novelty detection methods. This study enters the field of artificial intelligence explainability, which

aims at providing automated machine learning solutions which behaviour and decision making are intelligible for the user.

Bibliography

Adamson, G. W. and Boreham, J. (1974). The use of an association measure based on character structure to identify semantically related pairs of words and document titles. Information storage and retrieval, 10(7-8):253–260.

Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In International conference on database theory, pages 420–434. Springer.

Archimbaud, A., Boulfani, F., Gendre, X., Nordhausen, K., Ruiz-Gazen, A., and Virta, J. (2021). Ics for multivariate functional anomaly detection with applications to predictive maintenance and quality control.

ARINC 664P5 (2005). Aircraft data network part 5 network domain characteristics and interconnection. Standard.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

Ben Kraiem, I., Ghozzi, F., Péninou, A., and Teste, O. (2019). Pattern-based method for anomaly detection in sensor networks.

Bonnevay, S., Cugliari, J., and Granger, V. (2019). Predictive maintenance from event logs using wavelet-based features: An industrial application. In 14th Inter. Conf. on Soft Computing Models in Industrial and Environmental Applications (SOCO), volume 950 of Advances in Intelligent Systems and Computing, pages 132–141.

Borghesi, A., Libri, A., Benini, L., and Bartolini, A. (2019). Online anomaly detection in HPC systems. In IEEE Inter. Conf. on Artificial Intelligence Circuits and Systems, AICAS, pages 229–233.

Braschler, M. and Ripplinger, B. (2004). How effective is stemming and decompounding for german text retrieval? Information Retrieval, 7(3-4):291–316.

Bridgelall, R. and Tolliver, D. (2020). Accuracy enhancement of anomaly localization with participatory sensing vehicles. Sensors, 20(2):409.

Brown, A., Tuor, A., Hutchinson, B., and Nichols, N. (2018). Recurrent neural network attention mechanisms for interpretable system log anomaly detection. CoRR, abs/1803.04967.

Burns, L., Hellerstein, J., Ma, S., Perng, C., Rabenhorst, D., and Taylor, D. (2001). A systematic approach to discovering correlation rules for event management. In Proceedings of the 7th IFIP/IEEE International Symposium on Integrated Network Management, pages 345–359.

Cancedda, N., Gaussier, E., Goutte, C., and Renders, J. M. (2003). Word sequence kernels. The Journal of Machine Learning Research, 3:1059–1082.

Chalapathy, R. and Chawla, S. (2019). Deep learning for anomaly detection: A survey. arXiv preprint arXiv:1901.03407.

Chen, B. and Jiang, Z. M. J. (2017). Characterizing logging practices in java-based open source software projects - a replication study in apache software foundation. Empirical Software Engineering, 22(1):330–374.

Chen, B., Song, J., Xu, P., Hu, X., and Jiang, Z. M. (2018). An automated approach to estimating code coverage measures via execution logs. In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pages 305–316.

Chen, M. Y., Zheng, A. X., Lloyd, J., Jordan, M. I., and Brewer, E. A. (2004). Failure diagnosis using decision trees. In 1st Inter. Conf. on Autonomic Computing (ICAC), pages 36–43.

Chow, M., Meisner, D., Flinn, J., Peek, D., and Wenisch, T. F. (2014). The mystery machine: End-to-end performance analysis of large-scale internet services. In 11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14), pages 217–231.

Chuvakin, A. (2019). Public security log sharing site project.

Clément, L., Josiane, M., and Olivier, T. (2020). Outlier detection in multivariate functional data based on a geometric aggregation. In EDBT, pages 383–386.

Dai, H., Li, H., Shang, W., Chen, T., and Chen, C. (2020). Logram: Efficient log parsing using n-gram dictionaries. CoRR, abs/2001.03038.

Dawson, J. (1974). Suffix removal and word conflation. ALLC bulletin, 2(3):33–46.

Ding, R., Fu, Q., Lou, J. G., Lin, Q., Zhang, D., and Xie, T. (2014). Mining historical issue repositories to heal large-scale online service systems. In 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 311–322. IEEE.

Du, M. and Li, F. (2016). Spell: Streaming parsing of system event logs. In Bonchi, F., Domingo-Ferrer, J., Baeza-Yates, R., Zhou, Z., and Wu, X., editors, IEEE 16th International Conference on Data Mining, ICDM, pages 859–864. IEEE Computer Society.

Du, M., Li, F., Zheng, G., and Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In Thuraisingham, B. M., Evans, D., Malkin, T., and Xu, D., editors, ACM SIGSAC Conf. on Computer and Communications Security, pages 1285–1298.

Farshchi, M., Schneider, J.-G., Weber, I., and Grundy, J. (2015). Anomaly detection of cloud application operations using log and cloud metric correlation analysis. ISSRE.

Fisher, K., Walker, D., and Zhu, K. Q. (2008). Learnpads: automatic tool generation from ad hoc data. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pages 1299–1302.

Frakes, W. B. and Fox, C. J. (2003). Strength and similarity of affix removal stemming algorithms. In ACM SIGIR Forum, volume 37, pages 26–30. ACM New York, NY, USA.

Fu, Q., Lou, J., Wang, Y., and Li, J. (2009a). Execution anomaly detection in distributed systems through unstructured log analysis. In Wang, W., Kargupta, H., Ranka, S., Yu, P. S., and Wu, X., editors, ICDM 2009, The Ninth IEEE International Conference on Data Mining, Miami, Florida, USA, 6-9 December 2009, pages 149–158. IEEE Computer Society.

Fu, Q., Lou, J.-G., Wang, Y., and Li, J. (2009b). Execution anomaly detection in distributed systems through unstructured log analysis. In 9th IEEE Inter. Conf. on data mining, pages 149–158.

Fu, Q., Zhu, J., Hu, W., Lou, J.-G., Ding, R., Lin, Q., Zhang, D., and Xie, T. (2014). Where do developers log? an empirical study on logging practices in industry. In Companion Proceedings of the 36th International Conference on Software Engineering, pages 24–33.

Galgani, F., Compton, P., and Hoffmann, A. (2012). Knowledge acquisition for categorization of legal case reports. In Pacific Rim Knowledge Acquisition Workshop, pages 118–132. Springer.

Gartner (2014). Siem magic quadrant leadership report.

Gaussier, E., Goutte, C., Popat, K., and Chen, F. (2002). A hierarchical model for clustering and categorising documents. In European Conference on Information Retrieval, pages 229–247. Springer.

Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680.

Hafer, M. A. and Weiss, S. F. (1974). Word segmentation by letter successor varieties. Information storage and retrieval, 10(11-12):371–385.

Hamooni, H., Debnath, B., Xu, J., Zhang, H., Jiang, G., and Mueen, A. (2016). Logmine: Fast pattern recognition for log analytics. In Mukhopadhyay, S., Zhai, C., Bertino, E., Crestani, F., Mostafa, J., Tang, J., Si, L., Zhou, X., Chang, Y., Li, Y., and Sondhi, P., editors, ACM International Conference on Information and Knowledge Management, CIKM, pages 1573–1582.

Harman, D. (1991). How effective is suffixing? Journal of the american society for information science, 42(1):7–15.

He, P., Zhu, J., He, S., Li, J., and Lyu, M. R. (2016a). An evaluation study on log parsing and its use in log mining. In 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, pages 654–661. IEEE Computer Society.

He, P., Zhu, J., Xu, P., Zheng, Z., and Lyu, M. R. (2018a). A directed acyclic graph approach to online log parsing. arXiv preprint arXiv:1806.04356.

He, P., Zhu, J., Zheng, Z., and Lyu, M. R. (2017). Drain: An online log parsing approach with fixed depth tree. In Altintas, I. and Chen, S., editors, IEEE International Conference on Web Services, ICWS, pages 33–40. IEEE.

He, S., He, P., Chen, Z., Yang, T., Su, Y., and Lyu, M. R. (2020a). A survey on automated log analysis for reliability engineering. arXiv preprint arXiv:2009.07237.

He, S., Lin, Q., Lou, J.-G., Zhang, H., Lyu, M. R., and Zhang, D. (2018b). Identifying impactful service system problems via log analysis. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 60–70.

He, S., Zhu, J., He, P., and Lyu, M. R. (2016b). Experience report: System log analysis for anomaly detection. In 27th IEEE International Symposium on Software Reliability Engineering, ISSRE, pages 207–218. IEEE Computer Society.

He, S., Zhu, J., He, P., and Lyu, M. R. (2020b). Loghub: A large collection of system log datasets towards automated log analytics. arXiv preprint arXiv:2008.06448.

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013). Applied logistic regression, volume 398. John Wiley & Sons.

Hull, D. A. (1996). Stemming algorithms: A case study for detailed evaluation. Journal of the American Society for Information Science, 47(1):70–84.

Jiang, Z. M., Hassan, A. E., Hamann, G., and Flora, P. (2008). Automatic identification of load testing problems. In 2008 IEEE International Conference on Software Maintenance, pages 307–316. IEEE.

Jiang, Z. M., Hassan, A. E., Hamann, G., and Flora, P. (2009). Automated performance analysis of load tests. In 2009 IEEE International Conference on Software Maintenance, pages 125–134. IEEE.

Jivani, A. G. et al. (2011). A comparative study of stemming algorithms. Int. J. Comp. Tech. Appl, 2(6):1930–1938.

Johnson, S. C. (1967). Hierarchical clustering schemes. Psychometrika, 32(3):241–254.

Kimura, T., Watanabe, A., Toyono, T., and Ishibashi, K. (2018). Proactive failure detection learning generation patterns of large-scale network logs. IEICE Transactions on Communications.

Kondrak, G. (2005). N-gram similarity and distance. In International symposium on string processing and information retrieval, pages 115–126. Springer.

Kosala, R. and Blockeel, H. (2000). Web mining research: A survey. ACM Sigkdd Explorations Newsletter, 2(1):1–15.

Kraaij, W. and Pohlmann, R. (1994). Porter's stemming algorithm for dutch. Informatiewetenschap, pages 167–180.

Kraaij, W. and Pohlmann, R. (1996). Viewing stemming as recall enhancement. In SIGIR, volume 96, pages 40–48.

Krovetz, R. (1993). Viewing morphology as an inference process. In Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval, pages 191–202. ACM.

Laur, P. and Baril, X. (2004). Découverte de régularités pour l'intégration de données semi-structurées. In Hébrail, G., Lebart, L., and Petit, J., editors, Extraction et gestion des connaissances (EGC'2004), Actes des quatrièmes journées Extraction et Gestion des Connaissances, Clermont Ferrand, France, 20-23 janvier 2004, 2 Volumes, volume RNTI-E-2 of Revue des Nouvelles Technologies de l'Information, pages 537–542. Cépaduès-Éditions.

Lee, G., Lin, J., Liu, C., Lorek, A., and Ryaboy, D. (2012a). The unified logging infrastructure for data analytics at twitter. arXiv preprint arXiv:1208.4171.

Lee, G., Lin, J., Liu, C., Lorek, A., and Ryaboy, D. (2012b). The unified logging infrastructure for data analytics at twitter. arXiv preprint arXiv:1208.4171.

Liang, Y., Zhang, Y., Xiong, H., and Sahoo, R. K. (2007). Failure prediction in IBM bluegene/l event logs. In Proc. of the 7th IEEE Inter. Conf. on Data Mining (ICDM 2007), pages 583–588.

Lim, M.-H., Lou, J.-G., Zhang, H., Fu, Q., Teoh, A. B. J., Lin, Q., Ding, R., and Zhang, D. (2014). Identifying recurrent and unknown performance issues. In 2014 IEEE International Conference on Data Mining, pages 320–329. IEEE.

Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., and Chen, X. (2016). Log clustering based problem identification for online service systems. In 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pages 102–111. IEEE.

Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., and Meng, D. (2019a). Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pages 1777–1794.

Liu, J., Zhu, J., He, S., He, P., Zheng, Z., and Lyu, M. R. (2019b). Logzip: extracting hidden structures via iterative clustering for log compression. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 863–873. IEEE.

Lou, J.-G., Fu, Q., Yang, S., Xu, Y., and Li, J. (2010). Mining invariants from console logs for system problem detection. In USENIX Annual Technical Conference, pages 1–14.

Lovins, J. B. (1968). Development of a stemming algorithm. Mech. Translat. & Comp. Linguistics, 11(1-2):22–31.

Lu, J., Li, F., Li, L., and Feng, X. (2018a). Cloudraid: hunting concurrency bugs in the cloud via log-mining. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 3–14.

Lu, S., Wei, X., Li, Y., and Wang, L. (2018b). Detecting anomaly in big data system logs using convolutional neural network. In 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pages 151–158. IEEE.

Majumder, P., Mitra, M., Parui, S. K., Kole, G., Mitra, P., and Datta, K. (2007). Yass: Yet another suffix stripper. ACM transactions on information systems (TOIS), 25(4):18.

Makanju, A., Zincir-Heywood, A. N., and Milios, E. E. (2012). A lightweight algorithm for message type extraction in system application logs. IEEE Trans. Knowl. Data Eng., 24(11):1921–1936.

Masseglia, F., Poncelet, P., and Cicchetti, R. (2000). An efficient algorithm for web usage mining. Networking and Information Systems Journal, 2(5/6):571–604.

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., Chen, Y., Zhang, R., Tao, S., Sun, P., and Zhou, R. (2019). Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In Kraus, S., editor, International Joint Conference on Artificial Intelligence, IJCAI, pages 4739–4745. ijcai.org.

Messaoudi, S., Panichella, A., Bianculli, D., Briand, L. C., and Sasnauskas, R. (2018). A search-based approach for accurate identification of log message formats. In Khomh, F., Roy, C. K., and Siegmund, J., editors, Conference on Program Comprehension, ICPC, pages 167–177. ACM.

Mi, H., Wang, H., Zhou, Y., Lyu, M. R., and Cai, H. (2013). Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems. IEEE Trans. Parallel Distrib. Syst., 24(6):1245–1255.

Mizutani, M. (2013). Incremental mining of system log format. In 2013 IEEE International Conference on Services Computing, Santa Clara, CA, USA, June 28 - July 3, 2013, pages 595–602. IEEE Computer Society.

Montanari, M., Huh, J. H., Dagit, D., Bobba, R. B., and Campbell, R. H. (2012). Evidence of log integrity in policy-based security monitoring. In IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012), pages 1–6. IEEE.

Moral, C., de Antonio, A., Imbert, R., and Ramírez, J. (2014). A survey of stemming algorithms in information retrieval. Information Research: An International Electronic Journal, 19(1):n1.

Nagappan, M. and Vouk, M. A. (2010a). Abstracting log lines to log event types for mining software system logs. In Whitehead, J. and Zimmermann, T., editors, IEEE International Working Conference on Mining Software Repositories, MSR, pages 114–117.

Nagappan, M. and Vouk, M. A. (2010b). Abstracting log lines to log event types for mining software system logs. In Whitehead, J. and Zimmermann, T., editors, International Working Conference on Mining Software Repositories, MSR, pages 114–117. IEEE Computer Society.

Nagappan, M., Wu, K., and Vouk, M. A. (2009). Efficiently extracting operational profiles from execution logs using suffix arrays. In 2009 20th International Symposium on Software Reliability Engineering, pages 41–50. IEEE.

Nagaraj, K., Killian, C., and Neville, J. (2012). Structured comparative analysis of systems logs to diagnose performance problems. In Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12), pages 353–366.

Nandi, A., Mandal, A., Atreja, S., Dasgupta, G. B., and Bhattacharya, S. (2016). Anomaly detection using program control flow graph mining from execution logs. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 215–224.

Nestorov, S., Abiteboul, S., and Motwani, R. (1997). Inferring structure in semistructured data. ACM SIGMOD Record, 26(4):39–43.

Oberkampf, W. L. and Trucano, T. G. (2008). Verification and validation benchmarks. Nuclear engineering and Design, 238(3):716–743.

Oliner, A. J. and Stearley, J. (2007). What supercomputers say: A study of five system logs. In The 37th Annual IEEE/IFIP Inter. Conf. on Dependable Systems and Networks, DSN, pages 575–584.

Oprea, A., Li, Z., Yen, T.-F., Chin, S. H., and Alrwais, S. (2015). Detection of early-stage enterprise infection by mining large-scale log data. In 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 45–56. IEEE.

Paice, C. D. (1990). Another stemmer. SIGIR Forum, 24(3):56–61.

Paice, C. D. (1996). Method for evaluation of stemming algorithms based on error counting. Journal of the American Society for Information Science, 47(8):632–649.

Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In Proceedings of the 42nd annual meeting on Association for Computational Linguistics, page 271. Association for Computational Linguistics.

Poggi, N., Muthusamy, V., Carrera, D., and Khalaf, R. (2013). Business process mining from e-commerce web logs. In Business process management, pages 65–80. Springer.

Porter, M. F. (1980). An algorithm for suffix stripping. Program, 14(3):130–137.

Porter, M. F. (2001). Snowball: A language for stemming algorithms.

Savoy, J. (2006). Light stemming approaches for the french, portuguese, german and hungarian languages. In Proceedings of the 2006 ACM symposium on Applied computing, pages 1031–1035.

Shang, W., Jiang, Z. M., Hemmati, H., Adams, B., Hassan, A. E., and Martin, P. (2013). Assisting developers of big data analytics applications when deploying on hadoop clouds. In 2013 35th International Conference on Software Engineering (ICSE), pages 402–411. IEEE.

Sharf, Z. and Rahman, S. U. (2018). Performing natural language processing on roman urdu datasets. INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY, 18(1):141–148.

Shen, T., Zhou, T., Long, G., Jiang, J., Pan, S., and Zhang, C. (2018). Disan: Directional self-attention network for rnn/cnn-free language understanding. In Thirty-Second AAAI Conference on Artificial Intelligence.

Shima, K. (2016). Length matters: Clustering system log messages using length of words. CoRR, abs/1611.03213.

Soricut, R. and Och, F. (2015). Unsupervised morphology induction using word embeddings. In Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 1627–1637.

Tala, F. Z. (2003). A study of stemming effects on information retrieval in bahasa indonesia. Institute for Logic, Language and Computation, Universiteit van Amsterdam, The Netherlands.

Tan, Y., Nguyen, H., Shen, Z., Gu, X., Venkatramani, C., and Rajan, D. (2012). PREPARE: predictive performance anomaly prevention for virtualized cloud systems. In IEEE 32nd Inter. Conf. on Distributed Computing Systems, pages 285–294.

Tang, L., Li, T., and Perng, C. (2011). Logsig: generating system events from raw textual logs. In Macdonald, C., Ounis, I., and Ruthven, I., editors, ACM Conference on Information and Knowledge Management, CIKM, pages 785–794. ACM.

Ukil, A., Bandyoapdhyay, S., Puri, C., and Pal, A. (2016). Iot healthcare analytics: The importance of anomaly detection. In 2016 IEEE 30th international conference on advanced information networking and applications (AINA), pages 994–997. IEEE.

UpGuard (2019). The cost of downtime at the world's biggest online retailer. UpGuard.

Vaarandi, R. (2003). A data clustering algorithm for mining patterns from event logs. In IEEE Workhop on IP Operations & Management (IPOM), pages 119–126. IEEE.

Vaarandi, R. and Pihelgas, M. (2015). Logcluster - A data clustering and pattern mining algorithm for event logs. In Tortonesi, M., Schönwälder, J., Madeira, E. R. M., Schmitt, C., and Serrat, J., editors, International Conference on Network and Service Management, CNSM, pages 1–7. IEEE Computer Society.

Willett, P. (2006). The porter stemming algorithm: then and now. Program, 40(3):219–223.

Witten, I. H. (2004). Adaptive text mining: inferring structure from sequences. Journal of Discrete Algorithms, 2(2):137–159.

Wu, Y. (2018). Deeplog. `https://github.com/wuyifan18/DeepLog`.

Xia, B., Bai, Y., Yin, J., Li, Y., and Xu, J. (2020). Loggan: a log-level generative adversarial network for anomaly detection using permutation event modeling. Information Systems Frontiers, pages 1–14.

Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. (2009a). Largescale system problem detection by mining console logs.

Xu, W., Huang, L., Fox, A., Patterson, D. A., and Jordan, M. I. (2009b). Online system problem detection by mining patterns of console logs. In Wang, W., Kargupta, H., Ranka, S., Yu, P. S., and Wu, X., editors, ICDM, pages 588–597. IEEE Computer Society.

Xu, W., Huang, L., Fox, A., Patterson, D. A., and Jordan, M. I. (2010). Detecting large-scale system problems by mining console logs. In Fürnkranz, J. and Joachims, T., editors, Proc. of the 27th Inter. Conf. on Machine Learning (ICML), pages 37–46.

Xu, X., Zhu, L., Weber, I., Bass, L., and Sun, D. (2014). Pod-diagnosis: Error diagnosis of sporadic operations on cloud applications. In 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pages 252–263. IEEE.

Yamanishi, K. and Maruyama, Y. (2005). Dynamic syslog mining for network failure monitoring. In Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining, pages 499–508.

Yu, X., Li, M., Paik, I., and Ryu, K. H. (2012). Prediction of web user behavior by discovering temporal relational rules from web log data. In Int. Conf. on Database and Expert Systems Applications, pages 31–38. Springer.

Yuan, D., Mai, H., Xiong, W., Tan, L., Zhou, Y., and Pasupathy, S. (2010). Sherlog: error diagnosis by connecting clues from run-time logs. In Proceedings of the fifteenth International Conference on Architectural support for programming languages and operating systems, pages 143–154.

Yuan, D., Park, S., and Zhou, Y. (2012). Characterizing logging practices in open-source software. In 2012 34th International Conference on Software Engineering (ICSE), pages 102–112. IEEE.

Zhang, K., Xu, J., Min, M. R., Jiang, G., Pelechrinis, K., and Zhang, H. (2016). Automated IT system failure prediction: A deep learning approach. In IEEE Inter. Conf. on Big Data,, pages 1291–1300.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., Xie, C., Yang, X., Cheng, Q., Li, Z., et al. (2019). Robust log-based anomaly detection on unstable log data. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 807–817.

Zhao, X., Rodrigues, K., Luo, Y., Stumm, M., Yuan, D., and Zhou, Y. (2017). Log20: Fully automated optimal placement of log printing statements under specified overhead threshold. In Proceedings of the 26th Symposium on Operating Systems Principles, pages 565–581.

Zhu, J., He, P., He, S., and Liu, J. (2016). Logpai. `https://github.com/logpai`.

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., and Lyu, M. R. (2019). Tools and benchmarks for automated log parsing. In Sharp, H. and Whalen, M., editors, International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP), pages 121–130. IEEE/ACM.

A Log parsing evaluation

A.1 Best hyper-parameter configuration for the log parsing evaluation

This section presents the best hyper-parameters of log parsers on the reference datasets, obtained by a grid-search optimization with 120 attempted configurations. In Table A.1, we census the discrepancies between the default parameters proposed in LogPAI, and our optimization process. We also detail the values used for METI*N*G.

Method	METI*N*G		Drain		IPLoM		Spell
	n	ht	`st`	`depth`	`CT`	`lower_b`	τ
Proxifier	2	0.54			0.80	0.20	
OpenStack	2	0,4	0.60	4			0.88
HDFS	4	0.64					
Zookeeper	2	0.6	0.65	4	0.70	0.70	
Thunderbird	2	0.72	0.55	5			0.58
Linux	2	0.48					0.40
BGL	1	0.46	0.3	6	0.70	0.30	
Healthapp	3	0.64			0.20	0.20	
OpenSSH	2	0.4					
Mac	3	0.58	0.6	2	0.35	0.60	
Windows	1	0.5			0.60	0.60	
Android	2	0.4	0.55	6			
Hadoop	1	0.62			0.50	0.20	
HPC	3	0.42	0.5	4			
Spark	2	0.46					0.20

Table A.1 – Modifications of parameter setting (compared to `LogPAI`) and parameter setting propositions for METI*N*G.

B Anomaly detection

B.1 Conversion between the different types of labelling

In this section, we provide a formalized solution to convert the problem of detecting sequence or log anomalies to the problem of detecting temporal anomalies.

Abnormal sequences. In some reference datasets, where the sequences are defined and retrievable, the labelling concern whole sequences (namely OpenStack, HDFS). In this case, let S be the set of sequences present in the logs : $S = \{s_\ell, \ell \in L\}$. The labelling provides a subset of S of abnormal sequences, noted S_{ano}. The anomaly detection task therefore consists in retrieving these anomalies. Formally, let α be an anomaly detection alogrithm. α_s generates a subset $\alpha_s(L) \subset S$, that contains the detected anomalies. The objective of the anomaly detection problem is to maximize the similarity between S_{ano} and $\alpha_s(L)$. If Θ_s is a function that measures the similarity between two sets of sequences, the problem can be summed up as :

$$\underset{\alpha_s}{\text{maximize}} \;\; \Theta_s\big(S_{ano}, \alpha_s(L)\big) \tag{B.1}$$

Abnormal logs. In other reference datasets, the logs are individually labelled as normal or abnormal (e.g. BGL, Thunderbird). In this case, $L_{ano} \subset L$ is the subset of logs that are labelled as abnormal. If α is an anomaly detection algorithm in this context, $\alpha_l(L) \subset L$ is the subset of logs that are detected as abnormal by the algorithm. If Θ_l is a function that measures the similarity between two sets of logs, we obtain the following problem statement :

$$\underset{\alpha_l}{\text{maximize}} \;\; \Theta_l\big(L_{ano}, \alpha_l(L)\big) \tag{B.2}$$

Abnormal instants. The algorithms that work with a temporal log partition rather aim at detecting time windows that are abnormal. We can obtain the corresponding ground truth labelling thanks to simple rules. If the original labelling concerns logs, we can label the timestamps of these logs. Specifically, $\forall \ell \in L_{ano}$, t_{ell} is labelled as abnormal, meaning that a timestamp is abnormal if at least one of the log occurring during this timestamp is abnormal. Formally, we apply the following rule that builds $T_{ano} \subset [\![1..T]\!]$, the set of abnormal instants. Formally, the timestamp of these logs can be labelled with the rule :

$$\forall t \in [\![1..T]\!],\; t \in T_{ano} \iff \exists \ell \in L_{ano} \text{ s.t. } \ell_t = t$$

Common - Neural network architecture	
Number of hidden layers (in LSTM)	$[2:4]$
Number of neurons (in each hidden layer)	$2^i, i \leftarrow [6:10]$
Drop out	$0.1 \times i, i \leftarrow [0:5]$
Look-back h	$\left[\frac{\texttt{len}(seq)}{\delta} : 5 \times \texttt{len}(seq)\right]$
Common - Learning	
Optimizer	[Adam; RMSprop; SGD]
Learning rate	$10^i, i \leftarrow [-4:-1]$
Number of epochs	$[30:200]$
Batch size	$2^i, i \leftarrow [5:9]$
Weight initizialition	[kaimining normal; xavier normal; zeros; default]
Hidden layer intizialiation	[kaimining normal; xavier normal; zeros; default]
Temporal windows - NoTIL-LSTM and NoTIL-AE	
Window size w	$i \times \delta, i \leftarrow [1:5]$
Window shift δ	$[1 : \texttt{len}(seq)]$

Table B.1 – The hyper-parameters of the novelty detection models and the values tested. The initialization methods proposed are those implemented in the Pytorch framework. $\texttt{len}(seq)$ is the temporal length of a sequence, when it is defined and known. Otherwise, for BGL for instance, it is the length of the smallest temporal block (sequence of logs without interruption).

As a result, if α_t is an anomaly detection algorithm that detects abnormal instants, $\alpha_t(L) \subset [\![1..T]\!]$ is the subset of instants that are detected as abnormal by the algorithm. Let Θ_t be a function that measures the similarity between two sets of instants. The problem statement becomes :

$$\underset{\alpha_t}{\text{maximize}} \ \Theta_t\big(T_{ano},\ \alpha_t(L)\big) \tag{B.3}$$

The transformation is less obvious for sequence labelling. In the reference dataset, we observe that, sometimes, a manual analysis can improve the sequence labelling by precisely detecting the abnormal instants or logs (it is the case for OpenStack). If so, the problem comes down to the previous case (Equation B.3). Otherwise, a double partition can solve the problem : we first partition logs according to their sequence identifier, then for each sequence individually, we detect the abnormal instants. Finally, the sequences that contain abnormal instants are detected as abnormal sequences. Formally, the algorithm α_t is used individually in sequence to retrieve the anomalies : $\forall s \in S$, the subset of abnormal instants detected for the sequence is $\alpha_t(L_s)$. If this subset is not empty, then the sequence contains abnormal instants, and is alerted as abnormal. Hence, we obtain the following problem statement :

$$\underset{\alpha_t}{\text{maximize}} \ \Theta\big(S_{ano}, S_\alpha\big), \text{ with } \ S_\alpha = \big\{s \in S,\ \alpha_t(L_s) \neq \emptyset\big\} \tag{B.4}$$

B.2 Hyper-parameters of novelty detection methods

This section presents the hyper-parameters that are optimized during the validation process for novelty detection methods. In Table B.1, we census the hyper-parameters linked the neural

architecture, especially linked to the LSTM functioning. We also present the learning parameters, that modulate the training process. Finally, we present the two hyper-parameters used to build the temporal windows in the two versions of NoTIL.

List of Figures

List of Tables