Abstraction and comparison of execution traces for analysis of embedded multimedia applications

Christiane Kamdem Kengne

UNIVERSITÉ DE YAOUNDÉ I

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

préparée dans le cadre d'une cotutelle entre l'*Université de Grenoble* et l' *Université de Yaoundé I*

Spécialité : **Informatique**

Arrêtés ministériels : 6 janvier 2005 - 7 août 2006

Présentée par

« Christiane KAMDEM KENGNE »

Thèse dirigée par **Marie-Christine Rousset**
et codirigée par **Maurice Tchuente**

préparée au sein des **Laboratoire d'informatique de Grenoble (LIG) & Laboratoire International de recherche en informatique et mathématiques appliquées (LIRIMA)**

dans **les Écoles Doctorales EDMSTII**

Abstraction et comparaison de traces d'exécutions pour l'analyse d'applications multimédias pour système embarqués

Thèse soutenue publiquement le **« 5 décembre 2014 »**,
devant le jury composé de :

M. Laks V.S. LAKSHMANAN
Professeur à l'Université de Colombie-Britannique, Rapporteur
M. Pascal PONCELET
Professeur à l'Université de Montpellier 2, Rapporteur
M. Alexandre TERMIER
Professeur à l'Université de Rennes 1, Examinateur
Mme Céline ROBARDET
Professeure à l'INSA Lyon, Université de Lyon, Examinatrice
M. Éric GAUSSIER
Professeur à l'Université de Grenoble, Examinateur
Mme Marie-Christine ROUSSET
Professeure à l'Université de Grenoble, Directrice de thèse
M. Maurice TCHUENTE
Professeur à l'Université de Yaoundé I, Co-directeur de thèse
Mme Noha IBRAHIM
Maître de conférence à l'Université de Grenoble, Co-encadrante de thèse

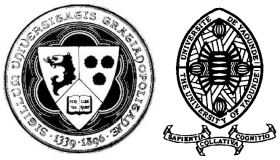
Université Joseph Fourier – Université de Yaoundé I

Abstraction and comparison of execution traces for analysis of embedded multimedia applications

présentée le 5 décembre 2014
Université de Grenoble Alpes, Université de Yaoundé I

pour l'obtention du grade de Docteur en Informatique
par

Christiane KAMDEM KENGNE



acceptée sur proposition du jury:

Pr Laks V.S. LAKSHMANAN, Rapporteur
Pr Pascal PONCELET, Rapporteur
Pr Alexandre TERMIER, Examinateur
Pr Céline ROBARDET, Examinatrice
Pr Éric GAUSSIER, Examinateur
Pr Marie-Christine ROUSSET, Directrice de thèse
Pr Maurice TCHUENTE, Co-directeur de thèse
Dr Noha IBRAHIM, Co-encadrante de thèse

Face à la roche, le ruisseau l'emporte toujours,
non pas par la force mais par la persévérance.
— H. Jackson Brown

À vous que j'aime si fort, mais qui êtes partis si tôt…
Maman, tata Appoline, tonton Pascal.

Remerciements

Je n'aurais jamais pensé qu'elles passeraient si vite ces années de thèse à Grenoble...C'est remplie d'émotions que je souhaite par ces quelques mots exprimer ma gratitude aux personnes et entités qui de près ou de loin, m'ont permise d'arriver jusqu'ici:

Mes directeurs de thèse Marie-christine Rousset et Maurice Tchuente. Vos questions et vos conseils n'ont cessé de me recadrer dans mon travail. Je ne saurais assez vous remercier de l'opportunité que vous m'avez donnée de travailler à vos côtés.

Noha, plus qu'une co-encadrante, tu as été une amie et une grande soeur. Tu m'as encouragée, m'a motivée tout au long de cette aventure en dents de scie. Tu as eu confiance en moi, encore plus que moi.
Merci Alex pour tout ce que tu m'as appris, grâce à toi et à tes blagues, j'ai réellement apprécié "fouiller des données".
Washio Sensei, thank you for all that I have learned and all that I am learning by collaborating with you.

Les Laboratoire d'informatique de Grenoble (LIG) et Laboratoire International de recherche en informatique et mathématiques appliquées (LIRIMA) qui m'ont accueillie en leur sein.

Je tiens à remercier les membres des équipes HADAS et SLIDE avec qui j'ai collaboré et échangé durant mes recherches.

L'administration de l'Université de Yaoundé 1 qui m'a autorisée à suspendre mes fonctions d'enseignement le temps de mes travaux de thèse.

Merci à la grande famille *Toko Wato* à laquelle je suis fière d'appartenir. Merci papa d'avoir toujours été présent pour nous. Merci aux filles Kamdem, Diane, Sandrine, Aurélie, Patricia et Audrey: mieux que mes soeurs vous êtes mes meilleures amies, et sans nos "divers" de tous les jours, la route aurait été bien laborieuse. Je te remercie Stéphane, de ta présence et de ton support qui restent sans pareil.

À vous Rodrigue et Simplice, Juliette, Léonie, Linda et Sandra, Blaise, Léon, Orléant et Serge, à mes amis d'enfance, à ceux rencontrés au long du chemin, à vous qui avez partagé mon quotidien et mes péripéties, à vous qui m'avez toujours soutenue, à vous qui avez une place si spéciale dans mon coeur, à tous: Merci!

Abstract

Nowadays, due to the increasing complexity in both the applications and the underlying hardware, it is difficult to understand what happens during the execution of these applications. Tracing techniques are commonly used to gather and provide information on application execution in the form of execution traces. The execution traces, which are sequences of events, can be very large (easily millions of events), hard to understand and thus require specific analysis tools. One critical case is the analysis of applications on embedded systems such as set-top boxes or smartphones, especially for understanding bugs of multimedia applications. In this thesis we propose two novel analysis techniques adapted to multimedia applications on embedded systems.

The first method reduces size of trace given to the analysts. This method needs to group sets of related events together. We propose an approach based on optimization and pattern mining techniques to automatically extract a set of subsequences from an execution trace. Our experiments showed that the method scales on large amounts of data and at the same time, highlighted the practical interest of this approach.

Our second contribution consists in proposing a diagnosis method based on the comparison of execution traces with reference traces. This approach is implemented in *TED*, our TracE Diagnosis tool. Experiments conducted on real-life use cases of multimedia application execution traces have validated that *TED* is scalable and brings added value to traces analysis. We also show that the tool can be applied on reduced size traces in order to further improve scalability.

Key words: execution trace, multimedia applications, sequence mining, optimisation, dissimilarity measures, anomalies detection.

Résumé

De nos jours, dû à la complexité croissante des applications et du matériel, il est difficile de comprendre ce qui se passe durant l'exécution de ces applications. Les techniques de traçage sont communément utilisées pour collecter et fournir les informations sur l'application sous forme de traces d'exécution. Les traces d'exécution, qui sont des séquences d'événements, peuvent être très volumineuses (elles atteignent facilement des millions d'événements), difficiles à comprendre et donc nécessitent des outils d'analyse spécifiques. Un cas critique est l'analyse d'applications pour systèmes embarqués tels les décodeurs ou les smartphones, en particulier pour comprendre les bugs d'applications multimédias. Dans cette thèse, nous proposons deux nouvelles techniques adaptées aux applications multimédia sur systèmes embarqués.

La première méthode réduit la taille de la trace donnée aux analystes. Cette méthode nécessite de regrouper un ensemble d'événements connexes. Nous proposons une approche basée sur des techniques d'optimisation et de fouille de motifs afin d'extraire automatiquement un ensemble de sous-séquences d'une trace. Nos expérimentations ont montré que cette méthode passe à l'échelle sur de gros volumes de données, et ont par la même occasion mis en évidence l'intérêt pratique de cette approche.

La seconde contribution consiste en la mise en place d'une méthode de diagnostic basée sur la comparaison de traces d'éxécution avec des traces de référence. Cette méthode est implémentée dans *TED*, notre outil de diagnostic de traces. Les expérimentations faites sur des cas d'utilisation concrets de traces d'exécution multimédia ont validé que *TED* passe à l'échelle et apporte une plus-value à l'analyse de traces. Nous montrons aussi que l'outil peut être appliqué sur des traces de taille réduite afin d'améliorer davantage le passage à l'échelle.

Mots clefs : trace d'exécution, applications multimedias, détection d'anomalies, fouille de séquences, techniques
d'optimisation, mesures de dissimilarité.

Contents

List of Figures

List of Tables

1 Introduction

Contents

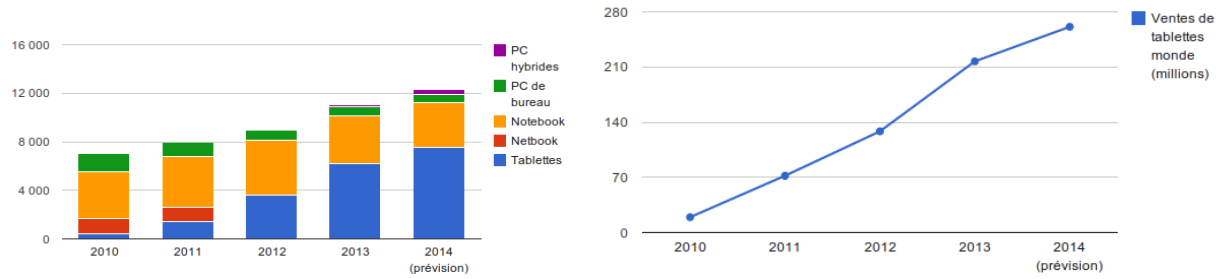
This thesis proposes solutions for analyzing embedded multimedia applications. The proliferation of embedded systems, from home boxes to tablets and smartphones, provides an everywhere access to multimedia contents. Developing multimedia applications is an area of high competition in which every second lost by a developer to debug the application amounts a financial loss for companies. The survival of the companies depends on the ability of the developers to quickly develop, debug, optimize software, and adapt to the constantly evolving platforms.

Embedded systems

Embedded systems can be defined as information processing systems embedded into enclosing products such as cars or telecommunication equipments. Such systems come with a large number of common characteristics, including efficiency requirements [VGW02].

As it has been the case for personal computers, the computational power provided by consumer electronics has not ceased to increase, motivated by ever increasing user demand. According to statistics presented by some websites ([Zdn14]), during 2013 in France, tablets were more widely sold than computers. Fig. 1.1(a) shows that more than six millions of tablets (blue color on figure) were sold versus four millions and eighty thousands of computers (all other colors).
In 2010, this part of the market was almost nonexistent. Fig. 1.1(b) presents the explosion of tablets sales over the world since 2010. Between 2010 and 2012, sales had increased by a factor of 6. This shows the interest of companies in quickly developing embedded applications, given

the amount of potential customers.



(a) Growth in sales of PCs and tablets in France

(b) tablets world sales(millions)

Figure 1.1 – Figure credits: [Zdn14]

People nowadays use their smartphones or tablets to watch video in many situations: during sport activities, while travelling, and so on. This situation increases the need to develop applications for these systems. One of the most used are multimedia applications in which video and audio decoding are the important tasks. That is why many companies are now launched in the race for easier and faster debugging techniques.

Multimedia applications

The most common definition of multimedia application is an application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video. In other words, multimedia applications carry out a series of transformations to a stream of data. These transformations (also called multimedia decoding) are not specific to a particular multimedia application (media players, video recorders and so on). This facilitates the reutilization of the decoding process.

A multimedia decoding is the process of rendering images and sounds on a screen, and the result must be of good quality, without interruption between images or any delay between picture and sound. This process deals with computations over frames. A frame is an image rendered during a known time interval.

The software infrastructure found on multimedia embedded systems consists of three layers: multimedia applications, the multimedia framework, and the operating system. Multimedia applications are generally platform independent since they sit on top of multimedia frameworks that isolate the application from the platform by providing the necessary services. Multimedia frameworks are in communication with the platform-dependent components of the operating system. Finally, the operating system is platform dependent since it has to communicate directly with the platform devices through drivers.
Multimedia frameworks, such as Gstreamer [Gst14] or VLC [Vid14] , offer a wide variety of processing elements that can be combined into a pipeline. The structure and the size of this

pipeline depend on the type of multimedia application. An example of a pipeline for a simple media player is shown in Fig. 1.2.

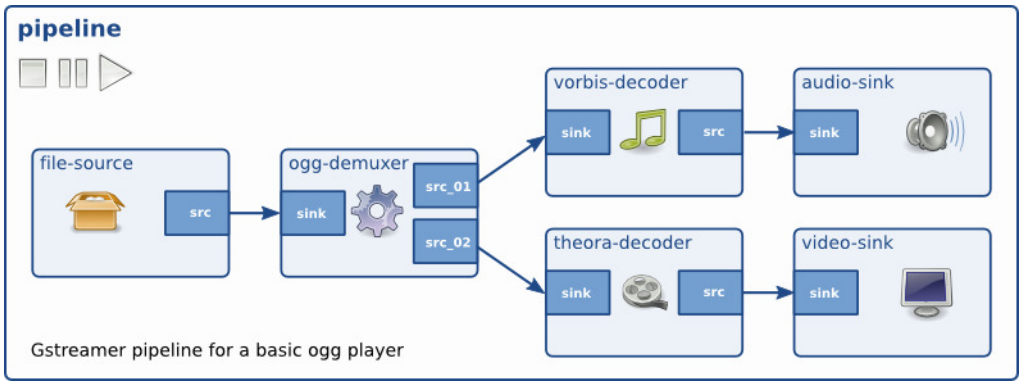


Figure 1.2 – Gstreamer pipeline for a simple media player - Figure credits: [Gst14]

The advantage of using such a framework to implement multimedia applications is that the developer can easily add, for example, support to new data formats or sources, by using plug-ins or components (piece of software that can be added to a bigger application and used transparently).
These components could be classified into: protocols handling, sources (for audio and video), formats (parsers, formaters, muxers, demuxers), codecs (coders and decoders), filters (converters, mixers, effects), sinks (for audio and video). For example, the *demuxer* is the component responsible for multiplexing the stream. It is responsible for extracting the contents of a given file/stream format, for instance AVI, OGG, MPEG2, WAV.
In Fig. 1.2, we can see a source component(file-source), a format component (demuxer), two codecs components (vorbis-decoder for audio and theora decoder for video) and two sink components (audio-sink and video-sink).

1.1 Challenges of trace analysis

Identifying the source and fixing the cause of unexpected or undesirable behavior in software can be a tedious, time-consuming and expensive task for developers. Even a code that is syntactically correct and functionally complete often leads to problems such as memory leaks or daemon tasks that can impact performance or lead to incorrect behavior. These oversights can be difficult to reproduce and even more difficult to locate, especially in large, complex applications. The analysis of multimedia application traces can reveal important information to enhance program execution comprehension. Many previous work [PR11],[Pou14],[Cue13] showed that tracing is the default debugging and validation technique when working on embedded systems. Tracing or trace recording implies detection and storage of relevant events during run-time, for later off-line analysis. Tracing is less intrusive than interactive debugging and cheaper than hardware solutions [KWK10]. However typical size of traces can

be in gigabytes, which hinders their effective exploitation by application developers.

1.1.1 Execution trace generation

An execution trace is defined as a sequence of events that represent the important moments in the execution of the program. Fig. 1.3 shows an example.

```
248657416;23033;0xe07a30;LOG;;ffmpeg;gstffmpegdec.c;948;alloc_output_buffer;ffdec_h2640;'calling
pad_alloc'
247972644;23033;0xe07ad0;LOG;;qtdemux;qtdemux.c;3761;gst_qtdemux_loop_state_movie;demuxer;'reading
109 bytes @ 138994'
248921074;23033;0xe07ad0;LOG;;filesrc;gstfilesrc.c;829;gst_file_src_create_read;filesrc0;'Reading 109 bytes at
offset 0x21ef2'
249040989;23033;0xe07a30;LOG;;ffmpeg;gstffmpegdec.c;1618;get_output_buffer;ffdec_h2640;'linsize 672 336
336'
249040959;23033;0xe07ad0;LOG;;qtdemux;qtdemux.c;3686;gst_qtdemux_decorate_and_push_buffer;demuxer;'
Pushing buffer with time 0:00:01.001000000, duration 0:00:00.033000000 on pad video_00'
249097587;23033;0xe07a30;LOG;;ffmpeg;gstffmpegdec.c;1621;get_output_buffer;ffdec_h2640;'data 0
3804454968 3804525272'
249151834;23033;0xe07ad0;LOG;;queue;gstqueue.c;656;apply_buffer;queue1;'last_stop updated to
0:00:01.034000000'
249259043;23033;0xe07ad0;LOG;;queue;gstqueue.c;588;update_time_level;queue1;'sink 0:00:01.034000000,
src 0:00:00.067000000'
```

Figure 1.3 – An example of execution trace

Traces are sequences of timestamped events produced by an application or a system. When detecting property violations, trace information can provide the path that led to this state, helping in discovering the cause of a disturbance. If an exhaustive search is not feasible, incomplete trace information may give clues to possible system behaviors. Different techniques exist to observe the execution of a software running on an embedded system. These techniques range from purely software-based to hardware-supported tracing techniques [KWK10]. Software-based tracing consists in instrumenting the code and inserting print statements in order to obtain a log of the execution. Hardware-based tracing consists in having dedicated hardware modules, where the components of the architecture can write their traces. For example, a bus profiler can collect tracing information and send it through a dedicated trace port.

1.1.2 Execution trace analysis

On-line execution traces are analysed *on the fly*, which means that they are analysed during the system execution. There is another alternative which is off-line, and where the sequence of events is stored in a file, that is used later for post analysis. In this work, we use off-line execution traces because this approach enables multiple analysis on the same execution. On-line analysis avoids to store the trace in a file, but gets potentially slower if several correctness properties must be checked on the same trace. In this case, it might be faster to generate the trace and perform all verifications off-line [GM04]. Fig. 1.4 presents three stages of application debugging techniques which depend on the moment when the verification is done. *Pre-execution analysis* uses the source code, *live debugging* considers on-line execution traces and *post-mortem debugging* works with off-line execution traces.

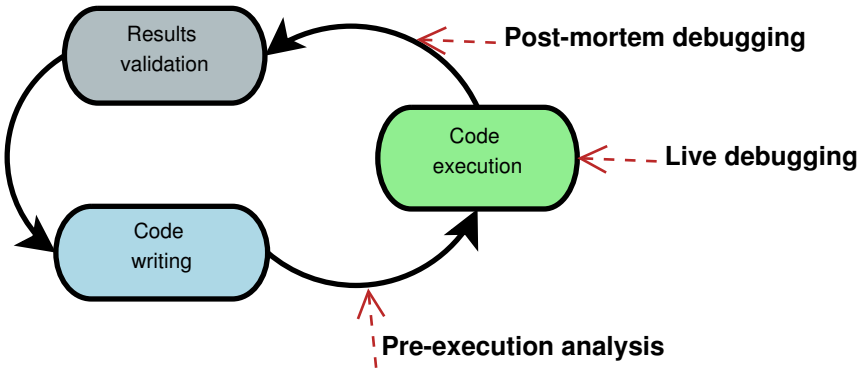


Figure 1.4 – Three possibilities for application analysis - Figure credits: [Pou14]

Generally, visualization tools help in the analysis of execution traces. They offer a graphical representation of the trace and several analysis functionalities. The latter help in analyzing the CPU time of each process, the memory used, etc. Different techniques have been proposed to visualize execution traces [CZvD11] and Fig. 1.5 shows an example of Trace viewer Pajè [CdKSB00].

The fact is that, because of the huge amount of information available, it is very difficult to

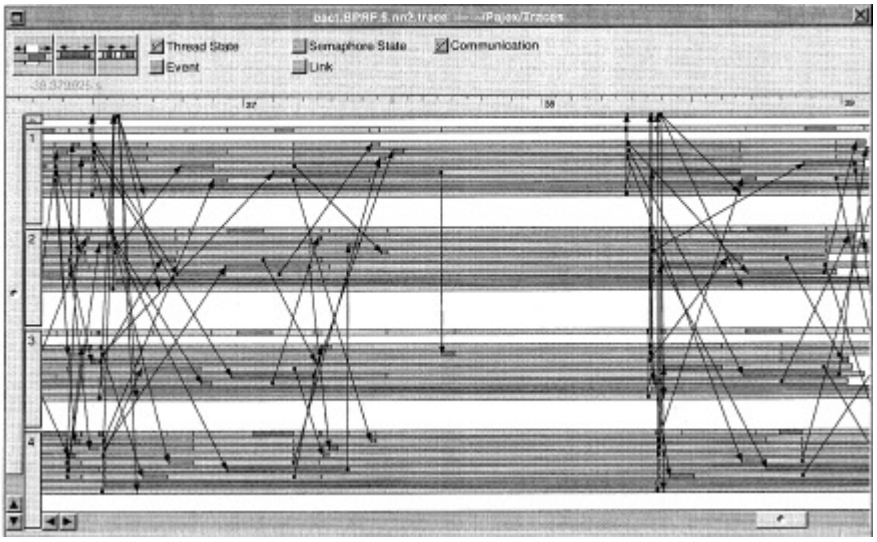


Figure 1.5 – Trace viewer Pajè - Figure credits: [CdKSB00]

analyze execution traces manually. For instance, the tool Parallel MJPEG [gue10] can produce a trace file of 7 Gigabytes for less than 5 minutes of video decoding. Another example is the STMicroelectronics video decoding application DVBTest which can produce a trace file of 1 Gigabyte for less than 10 minutes of playback. Viewers often face to thousand of pages representing events. It is then essential to set up improved analysis techniques which deal with data amount.

Therefore, our approach is to reduce the trace size in order to allow a better interpretation by the developers. Obviously, this size reduction should not lead to a loss of information in the trace. It must guide the developer in his analysis by presenting a trace more accessible in terms of events to explore.

In the rest of this chapter, we present the contributions of this thesis as well as the context in which this thesis was carried out. Finally, we present the organization of this document.

1.2 Contributions of this thesis

In this thesis, we propose two trace processing techniques that can be very useful when debugging embedded multimedia applications: The first method aims to abstract the execution traces in order to reduce its size and allows a better exploration. The idea behind this abstraction is to group sequences of events, and to replace these groups by meaningful blocks. In this manner, the execution trace initially seen as a sequence of events becomes a sequence of blocks, and is significantly reduced. The second method consists in detecting errors in a trace (where the trace is a sequence of events or blocks). The error detection is done by comparing the trace with a reference trace. This comparison aims to extract anomalies, i.e., patterns contained in the trace and absent in the reference trace.

Our contributions can be summarized as follows:

- **Abstraction of traces.** The abstraction is done by using sequences of events called *blocks*. We automatically extract these blocks from the trace by exploiting sequence mining techniques. The application of a classical mining process provides a certain quantity of block candidates. We only keep the most promising candidate blocks, i.e., blocks that ensure the best coverage of the original trace. We also propose an original method that combines into a single step the block discovering and the trace covering phases.

- **Anomaly detection by comparison of traces.** We propose to automatically provide a diagnosis by comparing two execution traces. The former is a reference trace corresponding to a correct behaviour, and the later the execution trace to analyse. We first identify a family of anomalies that are likely to occur in multimedia applications. We choose the most common types of anomalies and design a specific dissimilarity score for each of them. These scores help the developer to know how far the execution trace to analyse is from the correct behaviour. This highlights the anomalies contained in the execution trace. We propose a version of comparison on reduced size traces.

1.3 Outline

This thesis is organized as follows:

- Chapter 2 presents the **state of the art** in sequence abstraction and the most used methods in anomaly detection by comparison of sequences.
- Chapter 3 provides a **method to abstract execution trace** with a set of discovered event sequences.
- Chapter 4 proposes a **dissimilarity-based comparison method to analyse event sequences** applicable to original traces and reduced traces.
- **Conclusions** and some perspectives are given in Chapter 5.

In the rest of the document, the terms *trace*, *events sequence* and *execution trace* are used as synonyms.

List of publications

Journal :

1 - Sihem Amer-Yahia, Noha Ibrahim, Christiane Kamdem Kengne, Federico Ulliana, Marie-Christine Rousset, *SOCLE: Towards a Framework for Data Preparation in Social Applications*, Ingénierie des Systèmes d'Information 19(3): 49-72 (2014).

International conferences & workshops :

2 - Christiane Kamdem Kengne, Léon C. Fopa, Alexandre Termier, Noha Ibrahim, Marie-Christine Rousset, Takashi Washio, Miguel Santana, *Efficiently Rewriting Large Multimedia Application Execution Traces with few Event Sequences*, In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1348-1356), 2013.

3 - Christiane Kamdem Kengne, Noha Ibrahim, Marie-Christine Rousset, Maurice Tchuente, *Distance-based Trace Diagnosis for Multimedia Applications: Help me TED!*, Seventh IEEE International Conference on Semantic Computing, ICSC (acceptance rate 30%), (pp. 306-309), 2013.

4 - Christiane Kamdem Kengne, Léon C. Fopa, Noha Ibrahim, Alexandre Termier, Marie-Christine Rousset, Takashi Washio, *Enhancing the Analysis of Large Multimedia Applications Execution Traces with FrameMiner*,In Data Mining Workshops (ICDMW), IEEE 12th International Conference on (pp. 595-602), 2012.

Research report :

5 - Christiane Kamdem Kengne, Noha Ibrahim, Marie-Christine Rousset, Maurice Tchuente, *Distance-based Trace Diagnosis for Multimedia Applications: Help me TED!*, Research Report, (RR-LIG-045), LIG, Grenoble, France, 2013.

This thesis is conjointly done between two institutions: Grenoble University (SLIDE team), in France and University of Yaounde I (IDASCO team) in Cameroon. This research is supported by the FUI project Soc-Trace [ST11]

2 Related work

In this chapter we review the state of the art regarding the main topics concerning this thesis. We present in Section 2.1 some recent work related to abstraction. Then in Section 2.2 we study previous work on sequence-based anomaly detection.

Contents

2.1 Abstraction methods

This thesis is concerned with the study of execution traces, which are usually large sequences of low level events. Such traces have an extremely fine level of granularity, which makes them difficult to manipulate and understand for analysts. Our goal is thus to provide meaningful abstractions for rewriting traces with a coarser level of granularity. Data mining methods [PNSK+06], which are designed to find information in large volumes of data, are well adapted to discover such abstractions. One informal definition of abstraction is the process of summarizing in order to have a global (or general) view of the object to abstract, instead of details [ld14]. In the following we will survey recent work for discovering *patterns* in sequential data, which will be the basis of our approach for abstracting traces.

2.1.1 Abstraction techniques in data mining

According to [HKP12], mining "interesting" patterns is one of the core data mining tasks. A well studied measure of interest is the frequency of the pattern in the data. Frequent pattern mining problem is a combinatorial problem and existing algorithms generally output a huge number of discovered patterns. Having too many results makes the work of analysts difficult. Using only a minimum support threshold to control the number of patterns found has a limited effect. Recent mining methodologies tend to reduce the huge set of frequent patterns generated in mining by focusing on reduced sets of patterns that best describe the data. Such patterns are often qualified of *summarizing, representative* or *utility* patterns [LZW12, SV12, TV12, KIA+11]. The first approaches for computing such set of patterns are "two-step" approaches: first the complete set of frequent patterns is computed, then this set is postprocessed to compute a small set of useful patterns. More recent approaches focus on more efficient “one step” approaches which directly mine the small set of useful patterns.

► One trend is to mine ***summarizing*** patterns which are generally small sets of patterns, containing no redundancy and which provide the optimal lossless compression of the data. Many work[KT09, SV12, LMFC14, TV12] are based on the minimum descriptive length principle(MDL) in order to compress data. For explaining the MDL principle, assume that two parties P_1 and P_2 want to communicate, P_1 wants to send event sequence S to P_2 using as few bits as possible. In order to achieve this minimization of communication cost, P_1 has to select a model M from a class of models $\mathscr{M}$ to describe the sequence to send. The question is then: *How to choose M?* A brief formal description of MDL principle is the following:
Given a set of models $\mathscr{M}$, the best model $M \in \mathscr{M}$ is the one that minimises

$$L(M) + L(D|M)$$

where $L(M)$ is the length (in bits) of the description of M and $L(D|M)$ is the length of the description of the sequence when encoded with model M.

Krimp algorithm [VVLS11] is the pioneer algorithm in term of using MDL for identifying good pattern sets. It uses *code tables* as model. A code table is a dictionary between patterns and associated codes.

Example: (code table)
A code table has four columns. The first columns contains patterns. The second column contains code for identifying these patterns. The third column contains codes for identifying gaps and the last column contains codes specifying absence of gaps (See Fig. 2.1). Gaps and non-gaps of a pattern X indicate in the final encoding of a sequence, whether or not the symbol after X is part of a gap in the usage of X. There is no need of gaps and non-gaps for singletons events. a, b and c do not have codes for gap.

patterns	code	gap	non-gap
abc	*p*	*?*	*!*
ca	q	?	!
a	a		
b	b		
c	c		

Figure 2.1 – An example of code table

Given a code table, there are many ways of encoding a sequence. For more details, please refer to [VVLS11, TV12, SV12].

Knowing a decoding scheme of the database, Vreeken *et al.* [VVLS11] calculate the length of the code table and the length of the sequence S. Then, they propose an iterative strategy for discovering good code tables directly from the data. Smets and Vreeken with their algorithm *SLIM*[SV12] and Nikolaj *et al.* in [TV12] use also code tables. Both have similar strategies for discovering code tables, they estimate the gain of adding a pattern in the code tables and proceed iteratively.

In [KT09], the MDL principle is used to search the best balance between the short length of the summary and the accuracy of the data description. In this case, the selected model M is the segmental model that splits the sequence S into segments $S_1, \ldots, S_k$. They propose many greedy algorithms to compute a summary of a sequence S.

The key issue in designing an MDL-based algorithm for sequences is the encoding scheme that determines how a sequence is compressed, given some patterns. Authors in [LMFC14] use a dictionary-based encoding scheme, similar to code tables. They do not follow the traditional manner of MDL principle for calculating the description length as the number of bits. They assume that any number or character in data has a fixed length bit representation, which requires a unit memory cell. They propose a two-step candidate-based algorithm for mining compressing patterns, and *GoKrimp*, a one-step algorithm that avoids the expensive candidate generation step.

► Another trend is to mine ***representative*** patterns to best approximate or explain other patterns. Kim et al. studied in [KIA+11] the problem of finding a minimum set of signature patterns. Given a collection of objects where each element of the collection has an itemset and a label, a pattern is a signature pattern if it appears with a single label or in a single class. Their objective is to find a minimum set of signature patterns that make some properties easier to find. This set is called *signature pattern cover*. The signature patterns, categorized as discriminative patterns, can be mainly used in hardware design as a verification tool. The signature pattern cover problem can be viewed as a special case of the set covering problem. In the set covering problem, the goal is to cover all elements in a universe of elements U, using the smallest number of sets in a collection of subsets of elements S. In the signature pattern cover problem, let $U = o_1, o_2, \ldots, o_n$ be the collection of all objects and let $S = \{sup(P) \mid P$ is a signature pattern $\}$. Finding a minimum signature patterns cover is equivalent to finding a minimum set cover. However, the main difference between the two problems is that signature patterns are not given a priori, in contrast to set covering problem. Kim et al. propose a two-step approach and a one-step approach (or direct mining) for signature pattern mining.

Guimei et al. propose in [LZW12] three requirements that should be satisfied by ideal approaches in order to output a set of representative patterns: (i) produce a minimum number of representative patterns, (ii) have a good efficiency, (iii) restore the support of all patterns with error guarantee. They define a distance $D(X_1, X_2)$ between two patterns X_1, X_2 based on their supporting transaction sets and given a real number ϵ, X_1 is ϵ-$covered$ by X_2 if $X_1 \subseteq X_2$ and $D(X_1, X_2) \leq \epsilon$. With these two main definitions, the goal is then to select a minimum set of patterns (called representative patterns) that can ϵ-$cover$ all the frequent patterns. They assimilate the problem of finding a minimum representative pattern set to a set cover problem. The authors first present a greedy algorithm ($MinRPset$) and claimed that it gives the best possible polynomial time approximation algorithm for the set cover problem. They improved the efficiency of $MinRPset$ by applying three techniques: (a)Use closed frequent patterns (frequent patterns that are not included in another patterns having exactly the same support) only instead of frequent patterns. The motivation is that $MinRPset$ can be very slow when the number of frequent patterns is large; (b) Given a pattern X, use a particular structure called CFP-$tree$ to find the set ($C(X)$) of frequent closed patterns that can be ϵ-$covered$ by X, the hope being that the greedy algorithm can still find a near optimal solution; (c) Finally, apply a compression technique to compress $C(X)$. Intuitively, the fewer the number of patterns in $C(X)$ is, the more efficient the algorithm is. These strategies conduct to their third algorithm called $FlexRPset$.

► The last trend consists in ***high utility pattern*** mining. Previous work are interested in properties of patterns sets. High utility patterns mining rather focuses on individual pattern properties. However in both cases, the goal is to reduce the amount of output patterns, in order to facilitate the work of the analyst for instance. While previous approaches focused on the "representativity" of the patterns compared to data, work on high utility pattern mining provide an objective evaluation method (utility) of patterns. This evaluation method is used

to output high quality patterns. High utility patterns mining refers to the discovery of itemsets with "utilities" higher than a user-specific minimum utility threshold, where utility is a numerical value associated to items in input data. Liu et al. in [LWF12] illustrated utility pattern mining as follows: consider a supermarket manager who wants to identify every combination of products (itemset) with high sales revenue. An itemset has high utility if the revenue is no less than an expected level. In a supermarket database, each item has a price and each item in a transaction is associated with a distinct count which means the quantity of this item bought by someone. There are seven items in the utility table (Tab. 2.1), and seven transactions in the transaction table (Tab. 2.2).

item	a	b	c	d	e	f	g
utility	1	2	1	5	4	3	1

Table 2.1 – Utility table (Items Price)

	a	b	c	d	e	f	g
T_1		1	2	1			1
T_2	4	1	3	1	1		
T_3	4		2	1			
T_4			2		1	1	
T_5	5	2		1	2		
T_6	3	4	1		2		
T_7				1			5

Table 2.2 – Transaction table (Shopping transaction)

Tab. 2.3 gives utilities of some itemsets. For instance, $\{a,b,c\}$ appears in transactions T_2 and T_6; the utility of this itemset is then $4\times1+1\times2+3\times1+3\times1+4\times2+1\times1=21$.

Itemset	Utility
$\{a\}$	16
$\{a,b\}$	26
$\{a,b,c\}$	21
$\{a,b,c,d\}$	14

Table 2.3 – Itemsets utilities

The revenue (utility) of $\{a,b\}$ is 26 as customers who buy $\{a,b\}$ spend a total of 26 on $\{a,b\}$. Assume that the expected revenue is 25; $\{a,b\}$ is a high utility itemset but the others are not. The anti-monotone property does not hold with the utilities of itemsets. Indeed we can see in Tab. 2.3 that $\{a,b\}$ extends $\{a\}$ and has an utility higher than utility of $\{a\}$; on the other hand $\{a,b,c\}$ extends $\{a,b\}$ but has lower utility.

Many studies have been done for mining high utility pattern sets (HUI) in two step as [LYC08, TWSY10, WSTY12]. Recently some work were proposed to discover HUI without candidate generation. For instance, J. Liu et. al in [LWF12] proposed an efficient pruning of the search

space based on estimated utility values for itemsets. Some studies have been done to integrate utility into sequential pattern mining, and the most known is $USpan$ [YZC12] which defines the problem of mining high utility sequential patterns, but the approach used is a two-step approach, which may have difficulties to scale on very large datasets. [WLYT13] extend $USpan$ to episodes by proposing $UP\text{-}Span$ for mining high utility episodes. An episode is a collection of events, that occur relatively close to each other, in a given partial order [MTV97].

2.1.2 Discussion

We have presented approaches in the literature. In this section we discuss on some points of these approaches, for explaining why they are not suitable for execution traces analysis.

Some previous approaches such as [KIA+11] need external information (labels) to proceed to trace size reduction or pattern extraction. For a first processing of unknown traces where no information is available, an approach which does not need such information is better adapted.

Tab. 2.4 compares different work in terms of input, output and goal of the main algorithms.

A relevant point of this comparison is that, the user interest claimed for each method varies depending on the objective. Sometimes the aim is to: (i) explain data, i.e., find patterns which categorize data; (ii)represent data, i.e. find a reduced set of patterns that restore the support of all patterns; (iii) describe or compress data, i.e., use another coded representation of data which allow a gain of size. (iv) obtain value from data, i.e., find patterns which maximize the profit in a database.

Many approaches presented beforehand (section 2.1.1) focus on frequent itemsets as patterns. In multimedia applications where a strict sequencing of processing steps has to be enforced, an approach based on frequent sequences is better adapted.
By considering characteristics of execution traces (e.g., sequentiality) and the size of these traces (very large); taking into account the behaviour of embedded multimedia applications (regular steps in the process) and the need for the developer to master the size of the results that he wants to analyse, we think that a good method must: optimize sequence mining techniques, output a fixed number (specified by the user) of results, in order to simplify trace exploration.

Among the existing methods and to the best of our knowledge, only half of the methods involve sequences of events. Moreover, they do not allow to decide on the number of patterns found. Finally no one has an objective of improving sequence analysis by compressing data.

	Input	Output	Goal	output size parameter
[KT09]	Sequences	set of sequential patterns	compress data	parameter-free
[LZW12]	Itemsets	Min. set of patterns	approximate all patterns	parameter k
[KIA+11]	Itemsets with labels	Min. set of patterns	explain data	parameter-free
[SV12]	Itemsets	set of patterns	describe data	parameter-free
[TV12]	Sequences	Min. set of sequential patterns	describe data	parameter-free
[YZC12]	Sequences	set of sequential patterns	obtain value on data	parameter-free
[LQ12]	Itemsets	set of patterns	obtain value on data	parameter-free
[LWF12]	Itemsets	set of patterns	obtain value on data	parameter-free
[WSTY12]	Itemsets	set of patterns	obtain value on data	parameter k
[LMFC14]	Sequences	set of sequential patterns	describe data	parameter-free
[WLYT13]	Sequences	set of episodes	obtain value on data	parameter-free
[YZC+13]	Sequences	set of sequential patterns	obtain value on data	parameter k

Table 2.4 – Comparison of existing approaches.
The parameter k is the number of found patterns.

2.2 Sequence-based anomaly detection

Anomaly detection is an important topic which has considerable interest in many domains such as system failure, fraud detection, software debugging, or education. Another term referring to is outlier detection [GGAH14]. There is an extensive work on anomaly detection techniques in sequences. These techniques are grouped in two categories: semi-supervised anomaly detection and unsupervised anomaly detection. For the first category, one (or more) *normal* (or reference) sequence is assumed and used to compare with test sequence. In the second category, the task is to detect anomalous sequences from a database without knowing which are *normal* sequences.

For multimedia applications traces, it is possible to obtain a reference trace. That is why in this second section, we present an overview of semi-supervised anomaly detection techniques. We go further by presenting popular techniques of distances applied on sequences. Finally, we discuss distance-based techniques, tackled by this work.

2.2.1 Overview of semi-supervised anomaly detection techniques

[CBK12] classify these techniques into four categories: window-based techniques, Markovian techniques, Hidden Markov-based Model techniques and similarity-based techniques.

a- Window-based techniques

For these techniques, at a specific time, a short window of symbols within the test sequence is analyzed. Then another step is required to detect the anomaly type in the entire test sequence, based on the analysis of the short subsequence. By analyzing a short window at a time, these techniques try to detect the cause of possible anomaly type within one or a few windows. Fig. 2.2 presents different steps followed by these techniques.

Step 1-Extract fixed-length overlapping windows: The standard technique to obtain short windows from a sequence is to slide a fixed-length window, one symbol at a time, along the sequence. To explain the intuition for using techniques based on windows, let us assume that an anomalous test sequence t contains a subsequence t', which is the actual cause of anomaly. In a sliding window-based technique, if the length of the window is l, the anomalous subsequence t' will occur (partly or entirely) in $|t'|+l-1$ windows. Thus, the anomalous sequence can be potentially detected by detecting at least one of such windows.

Step 2-Assign anomaly scores to windows: many techniques have been proposed to assign a score $A(w_i)$ to a window w_i. One could (i) consider the inverse of the occurrence frequency of window; (ii) use a sequence of symbols $\langle \alpha, \beta \rangle_j$ called *lookahead pair*, such that the symbol β occurs in the j^{th} location after the symbol α in at least one of the windows in the reference sequence; (iii) use a classifier to assign an anomaly label to each window.

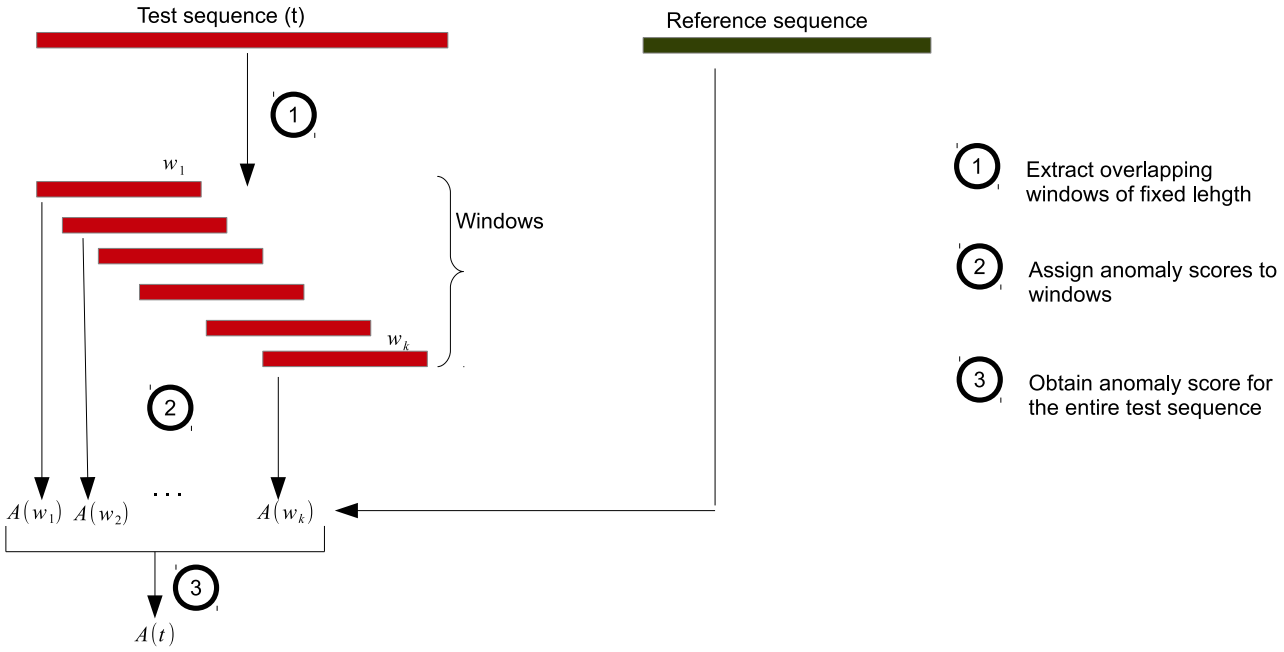


Figure 2.2 – Steps for window-based techniques

Step 3-Obtain anomaly score for the entire sequence: this global score is proportional to the number of anomalous windows in the test sequences; [WFP99], and [HFS98] propose many other methods to compute the overall anomaly score.

As disadvantage of these techniques, we notice the large amount of memory which could be necessary to store all windows. Another point is that, window-based techniques are highly dependent of the length l of the window. Setting an optimal value for l is challenging. If l is chosen to be very small, most windows will have a high probability of occurrence in the training sequences, while if l is chosen to be very large, most windows will have a low probability of occurrence in the reference sequence.
We refer to a non exhaustive list of work ([GGAH14],[WFP99],[HFS98]) for more information about window-based techniques.

b- Markovian techniques

These techniques learn a model from the reference sequences. The model is used as an approximation of the "true" distribution that generated the normal data.
As seen in Fig. 2.3, Markovian techniques operate in two phases: training and testing.
Training involves learning the parameters of a probabilistic model of the training sequences and **testing** involves computing the likelihood of the test sequence given the parameters.
Markovian techniques are usually divided into three categories ([Agg13], [GGAH14],[CBK12]):

(i) *Fixed Markovian techniques*: use a fixed history of length k to estimate the conditional

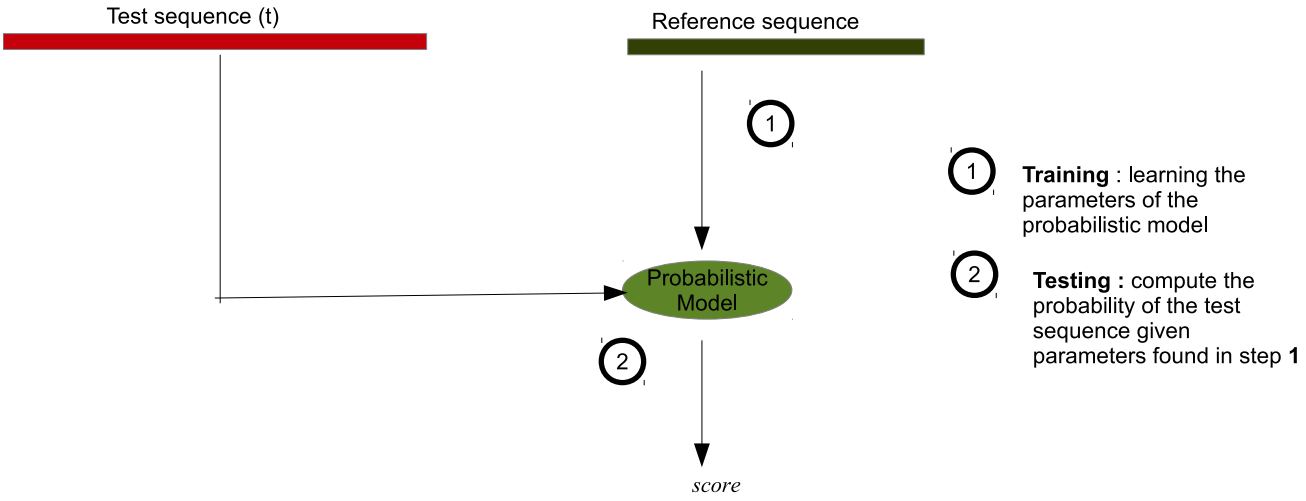


Figure 2.3 – Steps for Markovian techniques

probability of a symbol in the test sequence.

(ii) *Variable Markovian techniques:* an issue with fixed Markovian techniques is that they force each symbol of a test sequence to be conditioned on the previous k symbols of the sequence. Often, the frequency of a k-length substring, i.e., $(t_{i-k} \dots t_{i-1})$ may not be sufficiently large to provide a reliable estimate of the conditional probability of a symbol that follows this substring. Variable Markovian techniques try to address this issue by allowing symbols to be conditioned on a variable length history.

(iii) *Sparse markovian techniques:* variable Markovian techniques allow a symbol t_i to be analyzed with respect to a history that could be of different lengths for different symbols; but they still choose contiguous and immediately preceding symbols to $t_i \in t$. Sparse Markovian techniques are more flexible in the sense that they estimate the conditional probability of t_i based on symbols within the previous k symbols. These symbols are not necessarily contiguous or immediately preceding to t_i. In other words, the symbols are conditioned on a sparse history.

An issue with the basic fixed Markovian technique could be the huge amount of space needed to store frequencies used to compute symbols probabilities. For variable and sparse Markovian techniques techniques, the probability of a "truly" anomalous symbol will be boosted since it will be conditioned on a shorter history, whereas the fixed Markovian technique will still assign a low probability to such a symbol. Thus, the variable and sparse techniques might suffer with higher false negative rate.
We refer to a non exhaustive list of work ([YL00],[SCA06],[LSC97], [IRBT14]) for more information about Markovian techniques.

c- Hidden Markov-based Model (HMM) techniques

Hidden Markov-based models are finite state machines widely used for sequence modeling [CBK12]. These models use a sequence of transitions between states in a Markov chain to generate sequences. An HMM is parameterized by a hidden state transition matrix and an observation matrix. The three steps to obtain a HMM are: 1) for a given set of observation sequences, learn the most likely HMM parameters which result in maximum probability for the observation sequences, 2) for a given HMM, compute the hidden state sequence that is most likely to have generated a given test sequence, and 3) for a given HMM, with given state transition and observation matrices, compute the probability of a given test sequence.
One approach to use HMM techniques for anomaly detection is to (1) learn an HMM that best describes the normal training sequences, and then (2) compute the probability of the test sequence using the learned HMM. The negative log of the probability can be used as the anomaly score for the test sequence. This approach is summarized in Fig. 2.4.
Hidden markov models are different from the Markovian techniques by the fact that: each

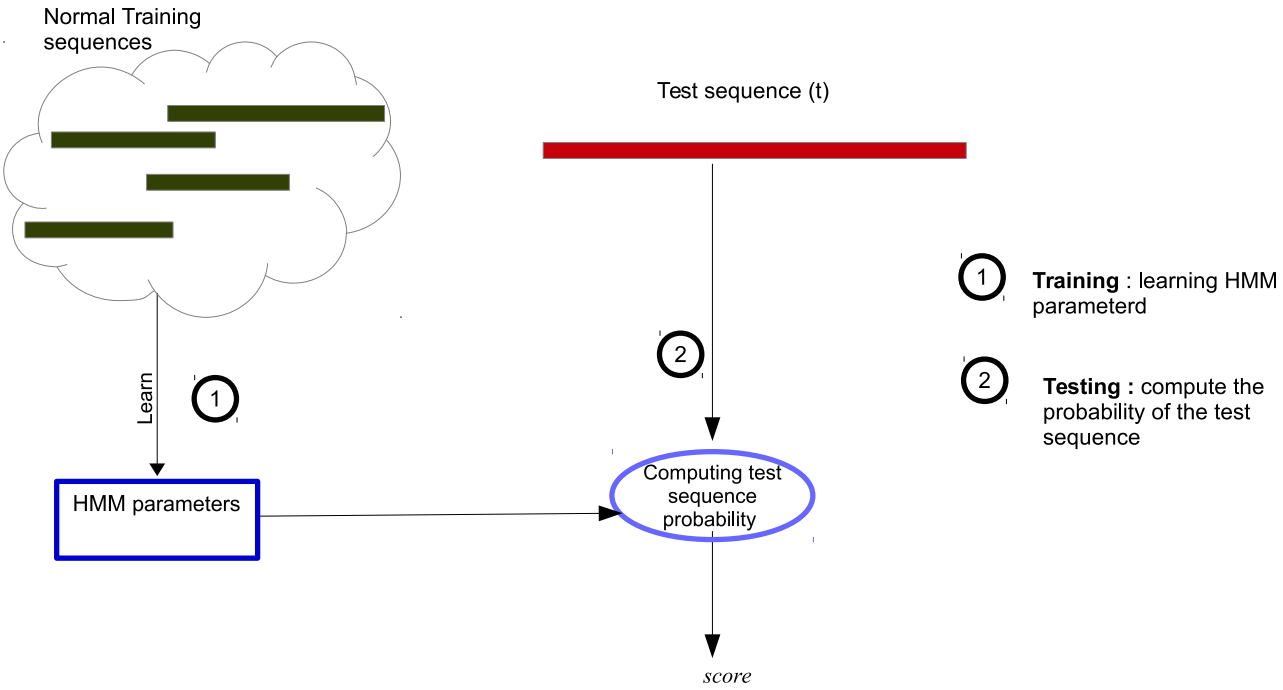


Figure 2.4 – Steps for HMM techniques

state in the Markovian techniques is well defined and is based on the last k positions of the sequence. This state is also directly visible to the user in terms of the precise order of transitions for a particular reference or test sequence [Agg13]. The generative behavior of the Markovian model is always known completely. In a Hidden Markov model, the states of the system are hidden, and not directly visible to the user.
The main assumption for HMM-based techniques is that the normal sequences are generated from a probabilistic model (the HMM). If this assumption does not hold or the parameters are

not estimated accurately, the HMM-based technique will not be able to effectively distinguish between normal and anomalous sequences.
We refer to a non exhaustive list of work ([Mör06],[CBK12]) for more information about HMM techniques.

d- Similarity-based techniques

According to [MRS08], sequence comparison has become a very essential tool in modern molecular biology and similarity measures need to be able to capture the rearrangements involving segments contained in the sequences. Several techniques have been proposed, which use different similarity (or dissimilarity) measures to compare a pair of sequences.

Common (dis)similarity measures on sequences
Over the years, many measures on sequences were developed and some of them are frequently used for purpose of anomaly detection. Hamming distance ([CCT10]), that simply counts the number of positions where the sequences differ. The family of edit distances([DZPM09]) is motivated by string matching. The similarity between two sequences is measured by determining the cost of transforming one into the other. The edit operations are insertion, deletion, and substitution of a symbol and can have different costs. With unit costs for all three operations, the Levenshtein distance ([Lev66]) is obtained. The Jaccard similarity ([CRF03]) is calculated by dividing the size of the intersection by the size of the union of the two multisets. Euclidian distance is used for clustering in information retrieval, but also in sequence comparison, in particular biological sequences [VA03]. DTW similarity ([ZHT06]) is especially used for time series to find an optimal alignment between two given sequences, under certain restrictions.
The advantage of similarity-based techniques is that one can use any existing or design new similarity measure and hence can devise a unique anomaly detection which is best suited for a given problem. A disadvantage of similarity-based techniques is that their performance is highly dependent on the choice of the similarity measure.

2.2.2 Discussion

In the context trace analysis of multimedia applications, it is indispensable to quickly figure out the specific anomaly. Similarity-based techniques are more suitable for this purpose. Given a non exhaustive overview of the literature on the subject of (dis)similarity measures on sequences, very few distances take into account the temporal aspect. In the field of multimedia execution traces, and more generally execution trace, each event is associated to a timestamp. This feature is essential to analyse these sequences.
Another important point is that the type of anomaly can differs across domains, and it is necessary to obtain the most accurate method for a data domain.
Finally, it seems that for a debugging setting, a dissimilarity measure would be more interesting if it is able to reveal not only that there is an anomaly by the value of distance, but also a

diagnosis for a family of anomalies in the domain. To the best of our knowledge, existing work on dissimilarity measures do not offer this option.

Our ambition is to propose a temporal distance that is adapted for trace comparison. We want an approach which returns a diagnosis to the user, added to the effective value of distance.

Fig. 2.5 shows the existing sequence-based anomaly detection techniques and highlights dissimilarity measures, which is the category corresponding to our contribution.

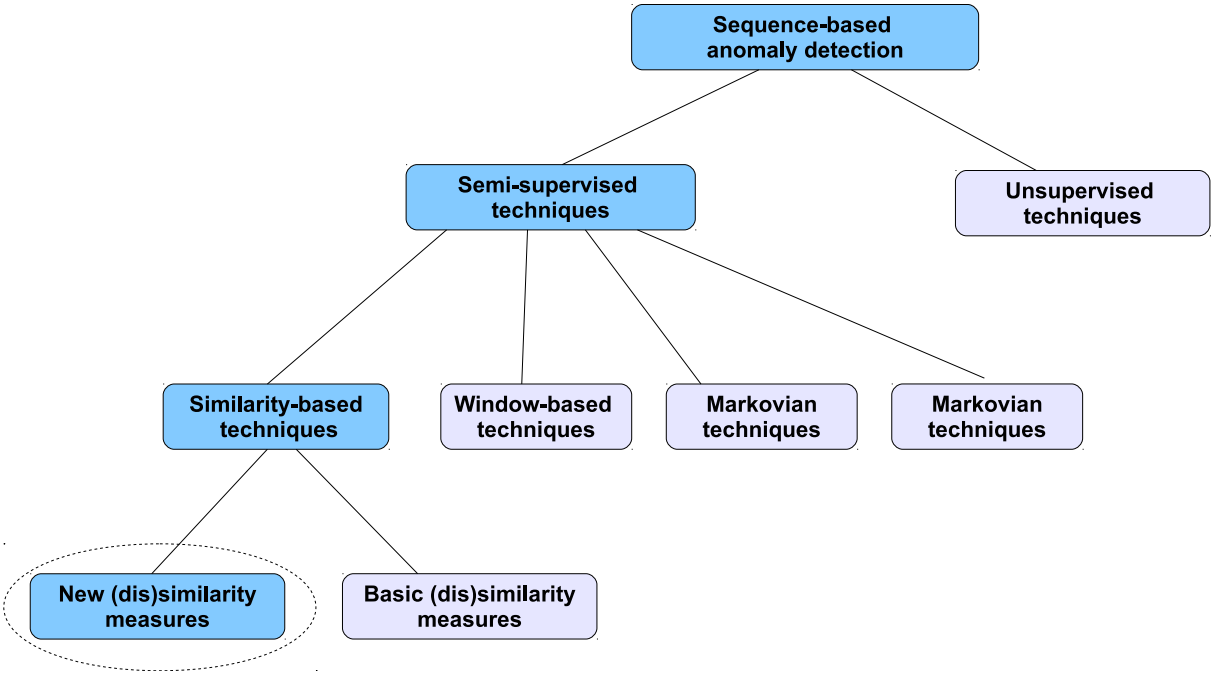


Figure 2.5 – Classification of sequence-based techniques

3 A method to abstract event sequences

In this chapter, we study the problem of finding a set of event subsequences that allows a rewriting of the original trace. Covering the trace with such subsequences of events, called blocks *hereafter, can help the developers to better understand the trace. The problem of computing such set of blocks, for rewriting a given sequence into a minimum length is NP-hard. Naive approaches lead to prohibitive running times that prevent the analysis of real world traces. We propose a practically efficient approach for mining blocks. Experiments show that our algorithm can analyse traces of up to two hours of video in practical applications. We also show experimentally the quality of the mined blocks, and the effectiveness to understand the structure of practical and massive trace data.*

Contents

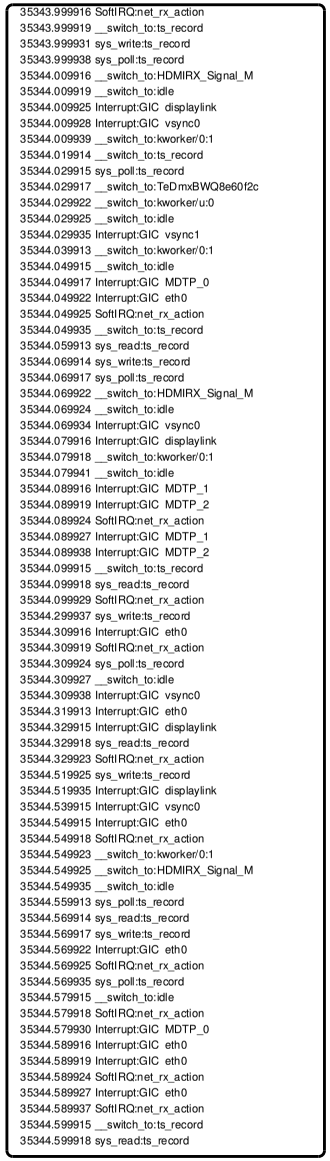
Context

We recall that the challenge in using multimedia execution traces is that their size can easily reach gigabytes for only few minutes of Audio/Video decoding.

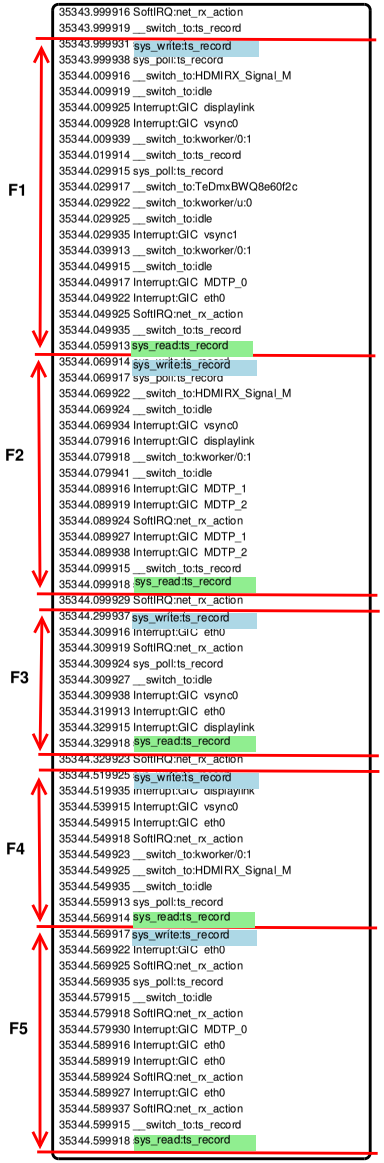
Various studies have proposed techniques to reduce the volume of traces ([Ste03],[WH]) with sampling methods. Indeed, sampling is a commonly used approach for selecting a subset of data to be analyzed.

These techniques can obtain a reduced execution trace that is not always representative of the entire trace [PSHLM11]. [PHL11] and [HEJ09] state that the general consensus in the trace analysis community is to provide more effective trace abstraction techniques. Our approach to provide an abstraction of event sequences is to exploit the following limited domain knowledge. We are dealing with traces of Audio/Video applications. In such applications, the decoding process follows a regular stream based on frames.

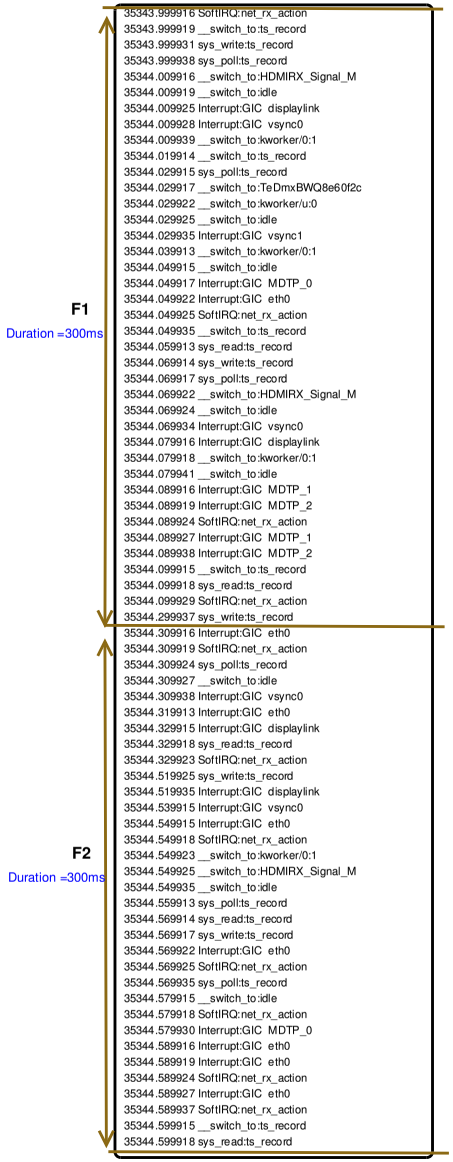
A **frame** is a semantic indicator on a regular unit of treatment, which could be easily given by the domain expert [VN12].
A frame is sometimes the main loop of the process. In this case, it is delimited by a start event and an end event (see Fig. 3.1(b)).
A frame could also follow a temporal delimitation (see Fig. 3.1(c)) or a more complex delimitation (see Fig. 3.1(d)).



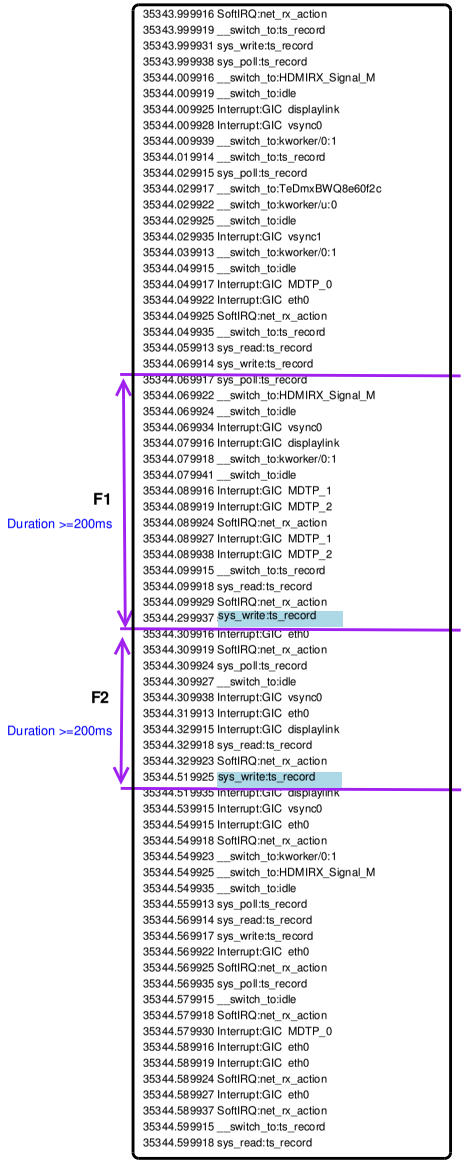
(a) A raw trace, timestamps are given in seconds



(b) Six frames delimited by a start event=*sys_write:ts_record* and an end event=*sys_read:ts_record*



(c) Two frames which are sequence of events occurring every 300 ms



(d) Two frames which are sequence of events occurring between two consecutive *sys_write:ts_record* events, when their time gap $\geq 200\,ms$

Contributions

In this chapter, an execution trace is split into frames. We investigate an approach for trace rewriting. This rewriting aims to simplify its exploration. The approach is based on covering frames by blocks that are subsequences of low-level events. More precisely, given a set of frames, the problem is to discover a given (small) number of blocks that cover (without overlaps) as much as possible each frame of the input set, thus making possible to rewrite them using blocks. Fig. 3.1 illustrates a trace with frames and blocks.

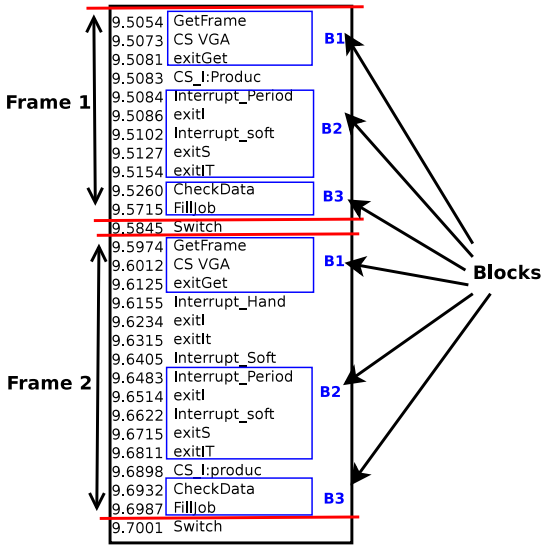


Figure 3.1 – A trace with blocks

The main contribution of this chapter consists in several efficient algorithms to discover blocks. To the best of our knowledge, these algorithms are the first that combine optimization techniques and pattern mining techniques in order to find blocks that provide the best coverage of the set of frames. These algorithms are mutually different in the sense they discover candidate blocks, either as a preliminary step independent of the coverage test, or combined with the coverage test. We propose some greedy approaches for scalability and validate them on gigabyte-sized traces.

RoadMap

This chapter is organized as follows: Section 3.1 states the problem and briefly gives some notations and important definitions. In Sections 3.2, we present our approaches based on greedy algorithms. Section 3.3 reports on experiments done on real traces of multimedia applications. We conclude in Section 3.4.

3.1 Preliminaries and problem statement

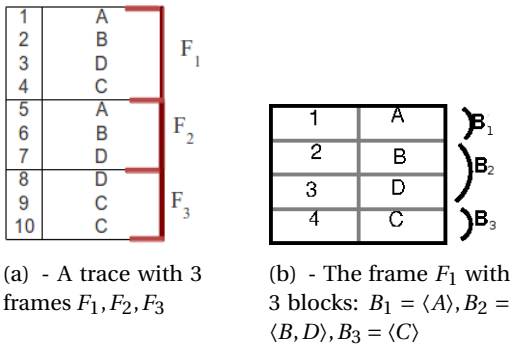
In this section we give the notations and definitions necessary to model our problem. As our domain is strongly related to frames, we will consider our granularity level to be a frame. Each block has to be related to a frame and meaningful in the frame decoding process.

Remark: As we have seen before, frames in a trace can be separated by out-of-frames events. *9.5845 Switch* in Fig 3.1 is an example of out-of-frame event. This splitting into frames is the first preprocessing task, and our interest is focused on the set of the obtained frames and not on the out-of-frame events. In practice, the events between two consecutive frames generally represent system events, which are out of the scope of this study.

3.1.1 Notations

Let Σ be a set of events. A *block* is a non empty sequence of events. A *timestamped event* is a pair (t,e) where $t \in \mathbb{N}$ is a timestamp and e is an event. *Frames* are sequences of timestamped events and a *trace* is a sequence of frames ordered by timestamps. The size of a sequence Q, denoted by $\|Q\|$, is the total number of events that it contains.

Example 1: In Fig. 3.2(a), the trace has three frames F_1,F_2,F_3. $\Sigma = \{A,B,C,D\}$. F_1 consists of four events: $\|F_1\| = 4$. In the same manner, $\|F_2\| = 3$, and $\|F_3\| = 3$.
For the three blocks $B_1 = \langle A \rangle$, $B_2 = \langle B,D \rangle$ and $B_3 = \langle C \rangle$, in Fig. 3.2(b):
$\|B_1\| = 1$, $\|B_2\| = 2$ and $\|B_3\| = 1$. F_1 can be rewritten as the sequence $\langle B_1,B_2,B_3 \rangle$.



(a) - A trace with 3 frames F_1,F_2,F_3

(b) - The frame F_1 with 3 blocks: $B_1 = \langle A \rangle, B_2 = \langle B,D \rangle, B_3 = \langle C \rangle$

Figure 3.2 – Example of trace, frames and blocks

3.1.2 Definitions

A block can occur at several places in a frame. To distinguish them, we first introduce the *occurrence time* of a block in a frame.

Definition 1. *Let* $B = \langle e_B^1,\ldots,e_B^v \rangle$ *be a block and let* $F = \langle (t_1,e_F^1),\ldots,(t_n,e_F^n) \rangle$ *be a frame.* B

occurs in F (denoted $B \sqsubseteq F$) between timestamps i and $i+v$ iff:

$$\forall j \in [i, i+v], \quad e_F^j = e_B^{j-i+1}.$$

i is then called an occurrence time of B in F.
Example 2: In Fig. 3.2(a), $B_1 = \langle B, D \rangle$ occurs in F_1 between timestamps 2 and 3 (occurrence time 2); it occurs in F_2 between timestamps 6 and 7 (occurrence time 6).

We focus on frames to cover event sequences within each frame by blocks; so we forbid: 1) to have several consecutive frames covered by a big block ; 2) to have a block that covers the end of a frame and the beginning of the next frame. In this setting, blocks of the covering can only occur inside individual frames. The *global coverage* of the set of frames can thus be expressed by a series of *local coverages* of each of the frames. A local coverage is a sequence of blocks taken from a given set of blocks, which satisfies the constraints stated below.

Definition 2. *Given a frame F and a set of blocks S, a sequence of blocks $C = \langle B_1, \ldots, B_m \rangle$ is a local coverage of F, with $\forall i \in [1, m]$ $B_i \in S$, if and only if all blocks in C occur in F in a non overlapping manner, and in order given by C.*
More formally, for each $B_i \in C$, let ϕ_i be the occurrence time of B_i in F, the following relation holds:

$$\forall i \in [1, m-1], \quad \phi_i + \|B_i\| \leq \phi_{i+1}$$

Note that the B_i are not necessarily distinct blocks: the same block can appear several times in a *local coverage*. Moreover, for a given F and S, there may be many local coverages satisfying the definition.

Example 3: In Fig. 3.2(b), $C = \langle B_1, B_2 \rangle$ occurs in F_1, and so is a local coverage of F_1 when considering $S = \{B_1, B_2, B_3\}$.

With the above definition, a *coverage* of a set of frames is dependant of locale coverage of each frame of the set. We define a *coverage* over $\mathscr{F} = \{F_1, \ldots, F_l\}$ using a set of candidate blocks S as a set of the local coverages of the frames.

Definition 3. *Let S be a set of candidate blocks $\{B_1, \ldots, B_n\}$ and $\mathscr{F} = \{F_1, \ldots, F_l\}$ be a set of frames. A coverage of $\mathscr{F}$ using S is a set $\{C_1, \ldots, C_l\}$ such that $\forall i \in [1, l]$, C_i is a local coverage of F_i using blocks in S.*

Based on the above definition, there may exist frames F_i such as their local coverage C_i is the empty sequence. Such frames contain no blocks of S.

The *covering rate* of a coverage is the proportion of events in the frames of a trace file that are covered by the blocks in the coverage.

Definition 4. *Let $\mathscr{C} = \{C_1, \ldots, C_l\}$ be a coverage of a set of frames $\mathscr{F} = \{F_1, \ldots, F_l\}$. The covering rate of $\mathscr{C}$ over $\mathscr{F}$ is defined as follows:*

$$coverRate(\mathscr{C}, \mathscr{F}) = \frac{\sum_{i=1}^{l} \sum_{j=1}^{v_i} \|B_j^i\|}{\sum_{i=1}^{l} \|F_i\|}$$

where B_j^i is the j-th block of C_i and v_i is the number of blocks of C_i.

Example 4: In Fig. 3.3, the set of frames is $\mathscr{F} = \{F_1, F_2, F_3\}$.
For the set of candidate blocks $S = \{\langle A, B\rangle, \langle B, D\rangle, \langle D, C\rangle\}$, a coverage of $\mathscr{F}$ is $\mathscr{C} = \{C_1, C_2, C_3\}$, with $C_1 = \langle\langle B, D\rangle\rangle$, $C_2 = \langle\langle B, D\rangle\rangle$, $C_3 = \langle\langle D, C\rangle\rangle$
$coverRate(\mathscr{C}, \mathscr{F})$ is $\frac{2+2+2}{10} = 0.6$

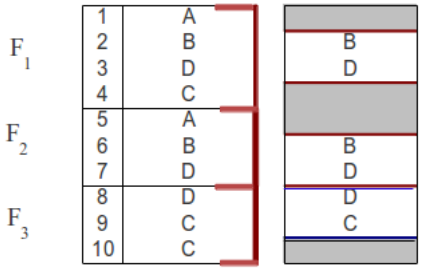


Figure 3.3 – A set of frames with a coverage:$\{\langle\langle B, D\rangle\rangle, \langle\langle B, D\rangle\rangle, \langle\langle D, C\rangle\rangle\}$

Because a set of candidate blocks S may lead to many local coverages of the same frame (Def. 2), it may also lead to many coverages for a set of frames. We define the *coverage rank* of S on F as the maximum rate of all the coverages that can be built from the set S.

Definition 5. *Let S be a set of blocks, $\mathscr{F}$ be a set of frames and let $\mathscr{C} = \{\mathscr{C}_1, \ldots \mathscr{C}_p\}$ be the set of all coverages of $\mathscr{F}$ using blocks in S. The coverage rank of S on $\mathscr{F}$ is defined as follows:*

$$coverRank(S, \mathscr{F}) = \max_{\mathscr{C}} \quad coverRate(\mathscr{C}, \mathscr{F})$$

Example 5: The *coverage rank* of S on the set of frames of Fig. 3.3 is 0.8 with the coverage $\{\langle\langle A, B\rangle, \langle D, C\rangle\rangle, \langle\langle B, D\rangle\rangle, \langle\langle D, C\rangle\rangle\}$

Remark: $\forall\ S, \mathscr{F},\ 0 \leq coverRank(S, \mathscr{F}) \leq 1$

Given a set of frames $\mathscr{F}$, we can compare the coverage ranks of different candidate blocks S having a fixed size k, and choose the candidate block S that maximizes the coverage rank. Such a set of blocks, with size k, is called *k-most representative block set* (denoted k-MRBS), and its elements, the *most representative blocks* (noted MR-blocks). The most representative blocks in a *k-most representative block set* provide the maximum power of coverage on the set of frames for any combination of k blocks.

Definition 6. *In a family $\{S_1, \ldots, S_q\}$ of sets of blocks where all sets have an identical size k, a k-most representative block set is a set S_i, satisfying:*

$$i = \underset{j \in [1,q]}{argmax} \quad coverRank(S_j, \mathscr{F})$$

Example 6: Let us consider Fig. 3.3. Assuming that $\langle C \rangle$, $\langle A, B \rangle$, $\langle B, D \rangle$, and $\langle D, C \rangle$ are frequent subsequences for the set of frames; let us consider the following sets consisting of 3 blocks:
$S_1 = \{\langle C \rangle, \langle B, D \rangle, \langle D, C \rangle\}$, $S_2 = \{\langle C \rangle, \langle A, B \rangle, \langle D, C \rangle\}$, $S_3 = \{\langle C \rangle, \langle A, B \rangle \langle B, D \rangle\}$,
$S_4 = \{\langle D, C \rangle, \langle A, B \rangle \langle B, D \rangle\}$;
The coverage rank of these sets are: $coverRank(S_1, \mathscr{F}) = 0.8$, $coverRank(S_2, \mathscr{F}) = 0.9$, $coverRank(S_3, \mathscr{F}) = 0.7$, $coverRank(S_4, \mathscr{F}) = 0.8$;
S_2 is then the *3-most representative block set* (3-MRBS).

3.1.3 Problem statement

The problem that we consider is the following:
Given as input a set of frames $\mathscr{F}$ and a number k, our goal is to output a k-most representative blocks set S that maximizes the coverage rate of $\mathscr{F}$.

The problem is then to rewrite each frame into a short description with a set of k blocks. These blocks should represent some main regular sub-parts (like the initialization step or the audio decoding step) of frames. Such sub-parts are likely to be frequent. They can thus be discovered with frequent sequence mining algorithms [ZXHW10].

This problem is a variant of the *packing problem* [Ege08] where one wishes to find a placement of items within one or several larger objects. One common packing problem is the one-dimensional knapsack problem. For this basic version, we are given a knapsack with a weight capacity W, a set of items I, each item from I having a weight and a profit-value assigned to it. The objective is to determine the subset of items which can be packed in the knapsack without violating the weight capacity limit, such that the sum of the profits of the items from the subset is maximal. The capacity constraint ensures that all items can fit inside the knapsack without "overlap". The problem is illustrated on Fig. 3.4. The multiple knapsack problem, is a variation of the knapsack problem where there are several knapsacks to fill.

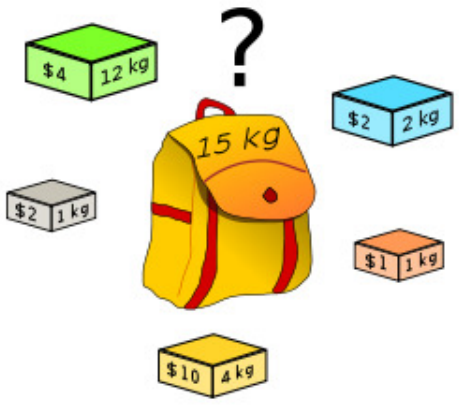


Figure 3.4 – A knapsack with weight limit must be filled with the most profitable set of items up to limit.- credits picture [wik14]

In our case the objects are frames, and the items are blocks. An additional constraint in our setting is that the location of each type of item is constrained: a block can only cover specific places of the frames. An additional difficulty of our case is that the items, i.e. the blocks, are not given as input, but must be computed from the data.
The packing problem is a NP-hard problem. There is no known generic algorithm for global optimization of this problem [Ege08].

In the next section, we propose several greedy approximation algorithms for the frames coverage problem.

3.2 Finding maximum covering of frames

The problem of finding a limited set of blocks allowing to maximally cover a set of frames can be decomposed into two subproblems:

- Find a large set S_0 of "candidate" blocks that are subsequences of frames
- Find $S \subset S_0$ such that $|S| = k$ and the coverage of the frames is maximized by the blocks of S

These two subproblems could be solved separately, or solved simultaneously in order to decrease execution time.

3.2.1 Sequential pattern mining

In the context of multimedia application debugging, particularly video decoding, a frame decoding generally follows the same procedure. A good heuristic for the discovery of the

blocks is then to assume that the blocks are *frequent subsequences* in the set of frames.
We apply frequent pattern mining algorithms in order to tackle the first subproblem of finding event subsequences in frames. A frequent pattern is a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set. Frequent pattern analysis was motivated by the objective of finding inherent regularities in data. Given a set of sequences and given a minimum support threshold, the sequential pattern mining consists in finding the complete set of frequent subsequences.
We are interested in strict sequences where it means that no gap is allowed. The patterns are made up of consecutive events, without possibility for relaxing the order constraint. We do not consider episodes, which are collection of events, that occur relatively close to each other, in a given partial order [MTV97]. In practice, we will use $ProfSpan$ algorithm [ZXHW10]. Other algorithms could be used as the adapted versions for strict sequences of GSP([SA96]) and $PrefixSpan$([PPC+01]) algorithms.

3.2.2 Approaches

a- Two step approaches

A naive approach consists in solving these two subproblems separately. The candidate blocks are first obtained as the result of a sequence mining algorithm returning the set of consecutive events that occur frequently in the set of frames in a trace file. In this section we explain how to obtain the blocks that maximally cover the frames.

(i)- CompleteBaseline Algorithm

A naive idea to obtain the k-most representative blocks set is to first generate all sets consisting

Algorithm 1 CompleteBaseline

Input: A set of frames $\mathscr{F}$, an integer k, a frequency threshold ε, minimum block size m
Output: The complete set S of k-most representative blocks sets

1: $S_0 \leftarrow computeFrequentSequences(\mathscr{F}, \varepsilon, m)$
2: $SS \leftarrow \{S_i | S_i \subseteq S_0, |S_i| = k\}$ {where $|S_i|$ means the number of blocks in S_i}
3: $max \leftarrow 0$; $S \leftarrow \{\emptyset\}$
4: **for** each $S_i \in SS$ **do**
5: $d \leftarrow coverRank(S_i, \mathscr{F})$
6: **if** $d > max$ **then**
7: $S \leftarrow \{S_i\}$; $max \leftarrow d$
8: **else if** $d = max$ **then**
9: $S \leftarrow S \cup \{S_i\}$
10: **end if**
11: **end for**
12: **return** S

of k blocks respectively, and then find among them the set that maximizes the $coverRank$. The algorithm for this simple method, termed *CompleteBaseline*, is presented above (Algorithm 1).

Although $CompleteBaseline$ Algorithm is simple and ensures that we obtain all exact solutions, it has an exponential time complexity : the number of subsets in SS in line 2 is C_n^k where $|S_0| = n$. Even for a reasonable dataset size, this method is not practically applicable. Assume $k = 5$ and $n = 100$ for instance, there is 776,160 subsets in SS. Therefore, we introduce a greedy algorithm to avoid this costly enumeration of all subset candidates.

(ii)- NaiveBaseline Algorithm

A simple greedy algorithm can be used to choose the k frequent sequences of S_0 that maximize coverage. We call this approach `NaiveBaseline` and it is depicted in Algorithm 2.

Algorithm 2 `NaiveBaseline`

Input: A set of frames $\mathscr{F}$, an integer k, a frequency threshold ε, minimum block size m
Output: A set S of frequent sequences giving a local optimum of coverage over $\mathscr{F}$, with $|S| = k$

1: $S_0 \leftarrow computeFrequentSequences(\mathscr{F}, \varepsilon, m)$
2: $S \leftarrow greedyChooseBlocks(S_0, \mathscr{F}, k)$
3: **return** S

S_0 contains all frequent sequences from the set of frames. This set is fed into the greedy algorithm, which produces the solution in line 2.

Algorithm 3 greedyChooseBlocks

Input: A set of frequent sequences $PatPool$, a set of frames $\mathscr{F}$, a maximal number of blocks k
Output: A set $S_{new} \subseteq PatPool$ of frequent sequences that gives a local optimum of coverage of $\mathscr{F}'$ (parts of $\mathscr{F}$ not already covered by the blocks of S), with $|S_{new}| \leq k$

1: $\mathscr{F}' \leftarrow \mathscr{F}$
2: $S_{new} \leftarrow \emptyset$
3: **while** $PatPool \neq \emptyset$ and $|S_{new}| < k$ **do**
4: $\quad B' \leftarrow argmax_{B \in PatPool}(coverRank(\{B\}, \mathscr{F}'))$
5: $\quad$ *// by definition of coverRank, B' is non-overlapping with all blocks of S_{new}*
6: $\quad S_{new} \leftarrow S_{new} \cup \{B'\}$
7: $\quad OB \leftarrow \{P \mid P \in PatPool, overlap(P, B')\}$
8: $\quad PatPool \leftarrow PatPool \setminus OB$
9: $\quad$ Remove from $\mathscr{F}'$ all instances of B'
10: **end while**
11: **return** S_{new}

We briefly review the function $greedyChooseBlocks$, presented in Algorithm 3. It is a standard greedy algorithm: the algorithm is given a target number of blocks k, and iterates as long as its solutions has less than k blocks and has not exhausted patterns of $PatPool$. At each iteration it chooses the block B' that gives best coverage in line 4 and adds it to the solution S_{new}. The algorithm then marks all blocks of $PatPool$ overlapping B' (they cannot be part of the solution any longer), and all instances of B' from a projection of the frames, in order to avoid doing computations for already covered parts.

The disadvantage of this approach is that it has a prohibitive computation time. Computing the frequent sequences (S_0) has an exponential time complexity in the number of events in the frames, and can output thousands, even millions of frequent sequences. The greedy algorithm is then confronted with a very large combinatorial search space, thus requiring high computation time. In our experiments, finding blocks for rewriting a small set of 200 frames (less than 10 seconds of video) took more than 10 hours on a standard computer. This simple approach do not scale to real world multimedia traces having tens of thousands of frames, and cannot be exploited to help multimedia application developers.

To address this limitation, we propose several approaches, which are based on the following ideas: **1)** the greedy algorithm should have a considerably smaller search space, i.e. receive several orders of magnitude less frequent sequences to choose from ; **2)** the number of frequent sequences should be reduced by considering coverage constraints.

(iii)- RandomBaseline Algorithm

An aggressive reduction in the number of input frequent sequences given to the greedy algorithm may prevent to find a solution with k elements. All the approaches that we propose are based on an iterative process, where in each iteration a set of frequent sequences is generated, then passed to the greedy algorithm. If the solution found has k blocks the algorithm stops, else it continues to further add extra blocks having large coverages until the number of blocks reaches k.

In order to illustrate this iterative process, consider the pseudo code of Algorithm 4 below.

Algorithm 4 `RandomBaseline`

Input: A set of frames $\mathscr{F}$, an integer k, a frequency threshold ε, minimum block size m, size of greedy algorithm input ℓ
Output: A set S of frequent sequences giving a local optimum of coverage over $\mathscr{F}$, with $|S| = k$

1: $S \leftarrow \emptyset$
2: $AllFrqSeq \leftarrow computeFrequentSequences(\mathscr{F}, \varepsilon, m)$
3: **while** $|S| < k$ and $AllFrqSeq \neq \emptyset$ **do**
4: $\quad PatPool \leftarrow$ randomly get ℓ frequent sequences from $AllFrqSeq$
5: $\quad S_{new} \leftarrow greedyChooseBlocks(PatPool, \mathscr{F}, k - |S|)$
6: $\quad S \leftarrow S \cup S_{new}$
7: $\quad \mathscr{F} \leftarrow$ Remove all blocks of S_{new} from $\mathscr{F}$
8: $\quad AllFrqSeq \leftarrow AllFrqSeq \setminus PatPool$
9: **end while**
10: **return** S

This algorithm is still a baseline because it only exploits intuition **1)** above when the size of the greedy algorithm input ℓ is much less than the size of all frequent sequences from frames: $\ell << |AllFreqSeq|$. As in the `NaiveBaseline` approach, the complete set of frequent sequences $AllFrqSeq$ is computed beforehand in line 2. Then in the iteration of lines 3-9, a

set $PatPool$ of fixed size ℓ (user given) is taken from $AllFrqSeq$ (line 4). This set is fed into the greedy algorithm, which produces (a part of) the solution in line 5. Blocks in the solution are removed from the frames (line 7), and if the solution does not have k blocks the algorithm continues. Note that in some rare cases (for example when $|AllFrqSeq|$ is small, or when k is set too large), the algorithm may not find a solution. Although such cases are unlikely to happen on real data, should they happen, the user would have to decrease k and/or decrease the support threshold ε.

This $greedyChooseBlocks$ algorithm guarantees the non-overlapping of the blocks in S_{new}: they will make a proper coverage of $\mathscr{F}$ according to Def. 3. However, it is not guaranteed that this coverage will have the highest $coverRank$ value, as the coverage is only estimated on the new block being added at each iteration, and not globally on the set of blocks. In line 4 of $greedyChooseBlocks$ algorithm (Algorithm 3), we use the heuristic of adding first blocks of highest coverage. This heuristic has very good practical results, as experimentally shown in Section 3.3. Moreover, it avoids the huge computational price of an exhaustive computation of the best $coverRank$.

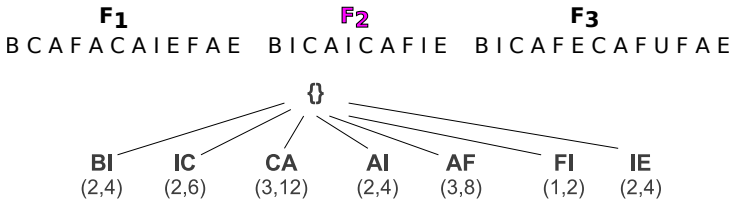
b- One step approaches

We now have the necessary material to present an improvement of baseline approaches. First, recall that the main difference between $NaiveBaseline$ and $RandomBaseline$ is that *RandomBaseline* does not consider all possible frequent patterns at once in the greedy algorithm: it proceeds iteratively, considering at each iteration a small random set $PatPool \subset AllFreqSet$. This should improve the computation time of the greedy algorithm, but because patterns of $PatPool$ are choosen at random, the coverage of the solution output may be far from optimal.

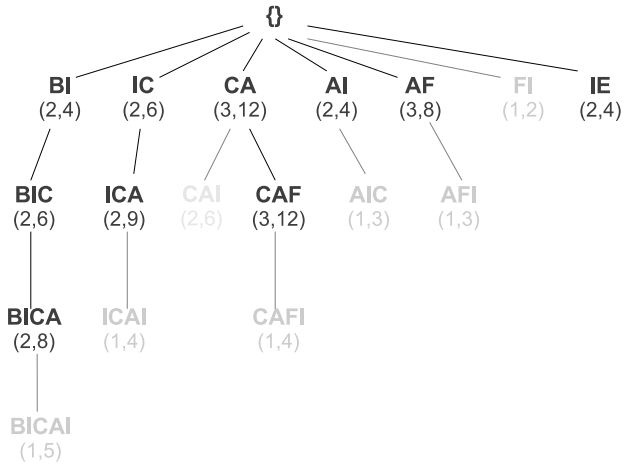
Our contribution thus consists in two approaches, coined `OneStepMultSon` and `OneStepOneSon`, which follow an iterative structure similar to $RandomBaseline$, but where, by exploiting intuition **2)** on previous page (reduce the number of frequent sequences), the choice of $PatPool$ is improved. In these approaches, $PatPool$ is guaranteed to contain blocks that all have high coverage, and that are already known to participate together in at least one local coverage.

(i)- OneStepMultSon Algorithm

The intuition is to avoid to output all the frequent patterns. The search for frequent patterns is restricted to a frame and only patterns which "promise" good coverage are kept. Fig. 3.5 illustrates the pattern growth of this algorithm. A frame $F2$ is randomly chosen from the set of 3 frames. The patterns of size two at least, occurring in two frames at least are found. However, the output is only frequent patterns that provide better coverage than the patterns they extend.



(a) values ($\{v_1, v_2\}$) for each pattern is a couple of v_1= frequency and v_2=number of events covered by the pattern. CA occurs in all frames and covers 12 events



(b) Output patterns in bold

Figure 3.5 – Pattern growth in `OneStepMultSon` algorithm

The pseudo-code for `OneStepMultSon` is given in Algorithm 5.

We first present the approach used in `OneStepMultSon`, by explaining function $getFramePatternsMS$, whose pseudo-code is given in Algorithm 6. This function is very similar to a classical pattern growth algorithm. However, there are two key differences with traditional pattern growth:

- all the patterns found are necessarily rooted in a random frame f, which severely restricts the search space
- for a given "seed" pattern of pattern growth (see below), the output is not all the frequent patterns extending this seed pattern, but only those extensions that provide better coverage than the patterns they extend.

The first step, shown in line 2 of Algorithm 6, is to find pattern growth "seeds". It is done by computing all subsequences of length m in the frame f. For each of these seeds, its extension is computed by the procedure $pattGrowth$ called in line 4. This procedure is shown in lines 7-21. It takes as input a pattern P, and modifies the final output $PatPool$. First the frequency of the P is tested (line 9). If P is frequent, its extensions in f are computed (line 11), i.e. all

Algorithm 5 `OneStepMultSon`

Input: A set of frames $\mathscr{F}$, an integer k, a frequency threshold ε, minimum block size m
Output: A set S of frequent sequences optimizing coverage over $\mathscr{F}$, with $|S| = k$

$S \leftarrow \emptyset$
$\forall f \in \mathscr{F}\ \ f.mark = false$
while $|S| < k$ and $\exists f \in \mathscr{F}$ s.t. $f.mark = false$ **do**
    $f \leftarrow random(\{f \in \mathscr{F} \mid f.mark = false\})$
    $PatPool \leftarrow getFramePatternsMS(f, \mathscr{F}, \varepsilon, m)$
    $S_{new} \leftarrow greedyChooseBlocks(PatPool, \mathscr{F}, k - |S|)$
    $S \leftarrow S \cup S_{new}$
    $\mathscr{F} \leftarrow$ Remove all blocks of S_{new} from $\mathscr{F}$
    $f.mark \leftarrow true$
end while
return S

Algorithm 6 getFramePatternsMS

Input: A frame $f \in \mathscr{F}$, a set of frames $\mathscr{F}$, a frequency threshold ε, minimum block size m
Output: A set $PatPool$ of coverage-maximal frequent sequences (each of length $\geq m$) occurring in f and frequent in $\mathscr{F}$

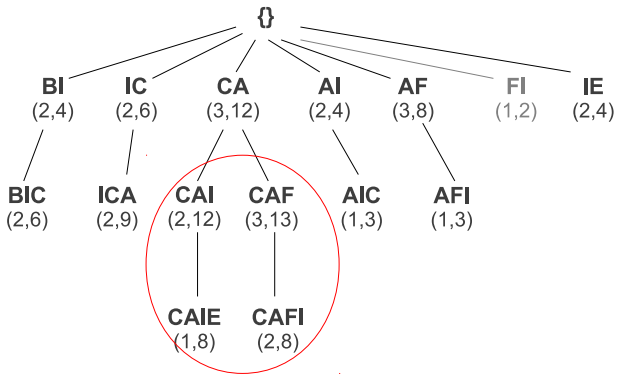
$PatPool \leftarrow \emptyset$
$Pool_m \leftarrow$ set of all sequences of consecutive events of f of length m
for all $P \in Pool_m$ **do**
    pattGrowthMS($P, \varepsilon, \mathscr{F}, PatPool$)
end for
return $PatPool$

procedure pattGrowthMS(**in** $P, \varepsilon, \mathscr{F}$, **in/out** $PatPool$)
begin
if $isFrequent(P, \varepsilon, \mathscr{F}) = true$ **then**
    $c_p \leftarrow coverRank(\{P\}, \mathscr{F})$
    $Ext_P \leftarrow \{e \in f \mid P + e$ is a sequence in $f\}$
    $Child_P \leftarrow \{P + e \mid e \in Ext_p$ s.t. $coverRank(\{P + e\}, \mathscr{F}) \geq c_p\}$
    if $Child_P \neq \emptyset$ **then**
        for all $P' \in Child_P$ **do**
            pattGrowthMS($P', \varepsilon, \mathscr{F}, PatPool$)
        end for
    else
        $PatPool \leftarrow PatPool \cup \{P\}$
    end if
end if
end // *procedure pattGrowthMS*

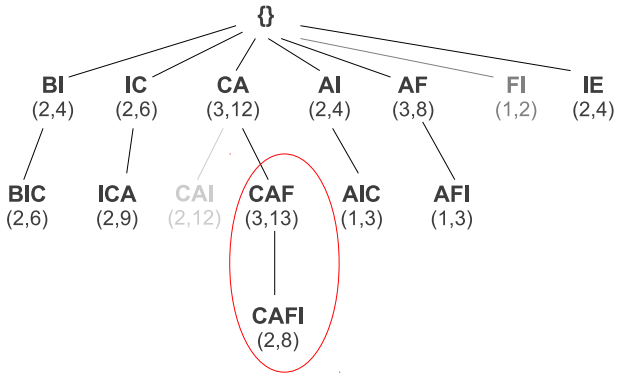
occurrences of P plus one event e are computed in f. The extensions that have a higher or equal coverage than P (line 12) are explored recursively in line 15. If none exist, P is added to the final result $PatPool$. In this way, $pattGrowth$ guarantees its maximality condition.

(ii)- OneStepOneSon Algorithm

The method used in $getFramePatternsMS$ is close to a full fledged pattern mining algorithm. Especially, it may explore and even return a number of frequent sequences exponential with the size of input frame f, due to the way it explores most subsequences of f.
In OneStepOneSon method, for a given seed pattern of pattern growth, the output is neither all the frequent patterns extending this seed pattern, nor extensions Ext that provide better coverage than the patterns they extend. It is only the extension that provides the maximum coverage, among patterns in Ext. The difference between `OneStepOneSon` and `OneStepMultSon` is illustrated in Fig. 3.6



(a) In `oneStepMultSon`, each extension of pattern CA, which covers 12 events is extended because each of them is frequent and covers at least 12 events



(b) In `oneStepOneSon`, only the extension CAF will be extended because it covers the maximum number of events.

Figure 3.6 – Pattern growth in `OneStepOneSon` algorithm

The pseudo-code for `OneStepOneSon` is identical to `OneStepMultSon`, except for line 5 where

the call to $getFramePatternsMS$ is replaced by a call to $getFramePatternsOS$. Algorithm 7 is the equivalent pseudo-code.

Algorithm 7 `OneStepOneSon`

Input: A set of frames $\mathscr{F}$, an integer k, a frequency threshold ε, minimum block size m
Output: A set S of frequent sequences optimizing coverage over $\mathscr{F}$, with $|S| = k$

1: $S \leftarrow \emptyset$
2: $\forall f \in \mathscr{F}\ \ f.mark = false$
3: **while** $|S| < k$ and $\exists f \in F$ s.t. $f.mark = false$ **do**
4: $\quad f \leftarrow random(\{f \in \mathscr{F} \mid f.mark = false\})$
5: $\quad PatPool \leftarrow getFramePatternsOS(f, \mathscr{F}, \varepsilon, m)$
6: $\quad S_{new} \leftarrow greedyChooseBlocks(PatPool, \mathscr{F}, k - |S|)$
7: $\quad S \leftarrow S \cup S_{new}$
8: $\quad \mathscr{F} \leftarrow$ Remove all blocks of S_{new} from $\mathscr{F}$
9: $\quad f.mark \leftarrow true$
10: **end while**
11: **return** S

The approach used in function $getFramePatternsOS$, presented in Algorithm 8, is a slight variation, which relaxes the exhaustiveness in search of traditional pattern mining algorithms.

Here, instead of choosing a set of possible extensions in line 12, only the extension $BestExt$ leading to the best coverage is retained. If it leads to a better coverage than original pattern P, then a single recursive call is performed on $\{P + BestExt\}$.

We now focus on the common parts of `OneStepMultSon` and `OneStepOneSon`, and position them w.r.t. `RandomBaseline`. The non-baseline algorithm are "one step", in the sense that they don't need to compute the whole set of frequent sequences beforehand. A reduced set of frequent sequences is computed at each iteration by $getFramePatternsMS$ / $getFramePatternsOS$ and feeds the greedy algorithm. The reduction comes from two points: first, the coverage constraint is taken into account during frequent sequence generation. Second, at each iteration of the algorithm, only sequences belonging to a selected random frame can be generated. This last point means that our approach is a based on a *sampling* of frames: the blocks output by `OneStepMultSon`/`OneStepOneSon` will be blocks appearing in a small set of randomly chosen frames (one random frame per iteration of the algorithm). This comes from the observation that usually multimedia application have a very regular execution, thus the sequences of events of the frames will be quite similar. When mining frequent sequences that should occur in most of the trace (support threshold > 50 %), taking a few sample frames is likely to quickly give enough blocks to get a good coverage of the whole trace.

In the algorithm, this is realized by first setting all frames as "unmarked" in line 2. In each iteration, a random sample frame f in selected in line 4, which is then passed as input to the frequent sequence mining algorithm. At the end of an iteration, frame f is marked in order to

Algorithm 8 getFramePatternsOS

Input: A frame $f \in \mathscr{F}$, a set of frames $\mathscr{F}$, a frequency threshold ε, minimum block size m
Output: A set $PatPool$ of cover- maximal frequent sequences (each of length $\geq m$) occurring in f and frequent in $\mathscr{F}$

```
1:  PatPool ← ∅
2:  Pool_m ← set of all sequences of consecutive events of f of length m
3:  for all P ∈ Pool_m do
4:      pattGrowthOS(P, ε, F, PatPool)
5:  end for
6:  return PatPool
7:  procedure pattGrowthOS(in P, ε, F, in/out PatPool)
8:  begin
9:  if isFrequent(P, ε, F) = true then
10:     c_p ← coverRank({P}, F)
11:     Ext_P ← {e ∈ f | P + e is a sequence in f}
12:     BestExt ← argmax_{e∈Ext_p}(coverRank({P + e}, F)) s.t. coverRank({P +
        BestExt}, F) ≥ c_p
13:     if BestExt exists then
14:         pattGrowthOS({P + BestExt}, ε, F, PatPool)
15:     else
16:         PatPool ← PatPool ∪ {P}
17:     end if
18: end if
19: end // procedure pattGrowthOS
```

avoid selecting it again.

Note that this is different from the $RandomBaseline$ approach, where at each iteration a random sample of blocks is selected, but there are no constraints on where do this blocks come from: they may all come from different frames, possibly never appearing together in local coverages.

(iii)- Computational complexity of one step approaches

In $getFramePatternsOS$, in the worst case the number of frequent sequences examined is $O(|f|^3)$: $Pool_m$ has less than $|f|$ elements, for each of these elements there can't be more than $|f|$ extensions to check in line 12, and the number of recursive calls to $pattGrowthOS$ it generates is bounded by $|f|$. $getFramePatternsOS$ is thus polynomial in the size of the input frame. This was not the case in $getFramePatternsMS$, where it was possible to have one recursive call to $pattGrowthMS$ per extension, leading to a worst case complexity of $O(2^{|f|})$. Thus, `OneStepOneSon` should exhibit better execution times than `OneStepMultSon`, possibly with a minor degradation of coverage value of the solution.

Comparing the performances of `NaiveBaseline`, `RandomBaseline`, `OneStepMultSon` and

`OneStepOneSon` in terms of computation time and of coverage value is one of the objective of the next section. We will also show the interest of the k-most representative block sets obtained on real execution traces.

3.3 Experiments

In this experimental section, our goal is first to evaluate the scalability on large real world traces of the four greedy approaches presented above. For each approach, we will also evaluate the average coverage given by a solution, in order to evaluate the quality of the solution found. We will also show how real traces can be rewritten as sequences of most representative blocks, and show how helpful it can be for application developers.

3.3.1 Experimental settings

We implemented the greedy algorithms of Section 3.2 in Python 3. The frequent sequence mining algorithm used is an implementation of ProfScan [ZXHW10], realized in our research group by PhD student Leon Fopa. The experiments were run on an Intel Xeon E5-2650 at 2.0GHz with 32 Gigabytes of RAM with Linux. The parameters of the algorithms are fixed to $k = 10$, $\varepsilon = 75\%$, $m = 2$ and $\ell = 300$.

▷ $k = 10$: a number of patterns easy to annotate by a developer

▷ $\varepsilon = 75\%$: we are interested by very regular subsequences and this value is a good compromise.

▷ $m = 2$: a block of size one is less likely to represent a sub-part of video decoding process than a block of size ≥ 2

▷ $\ell = 300$: with the previous parameters ε and m, we computed all the frequent sequences per dataset. We then computed the average number of frequent sequences and ℓ is the one-tenth of this value.

The datasets used are traces from two real applications, described below.

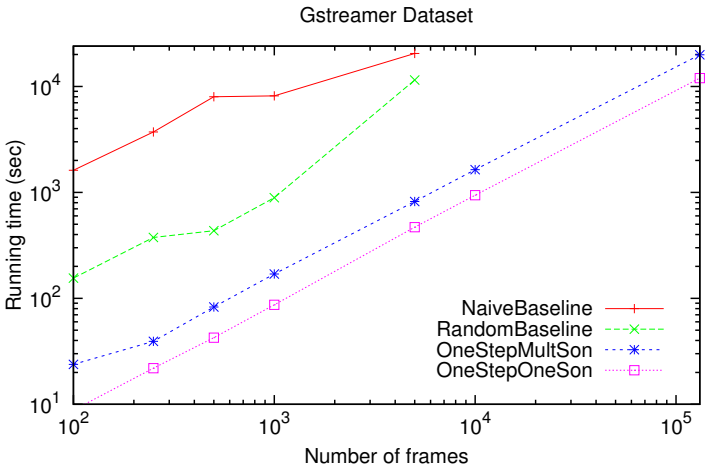
* **Gstreamer application**: Gstreamer [Gst14] is a powerful open source multimedia framework for creating streaming applications, used by several corporations as Intel, Nokia, STMicroelectronics and many others. It is modular, pipeline-based and open source. For our experiments we decoded a movie of 2 hours using Gstreamer on a Linux platform, with the $ffmpeg$ plugin for video decoding. The execution trace obtained has a size of 1 Gigabyte. This trace comprises 131,340 frames, for a total of 5,120,973 events.

* **DVBTest application**: It is a test video decoding application for STMicroelectronics development boards. This application is widely used by STMicroelectronics developers. In our trace, the application is run on a STi7208 SoC, which is used in high definition set-top boxes produced by STMicroelectronics. The execution trace contains both application events and system-level events. It is generated from a $ST40$ core of the SoC, which is dedicated to appli-

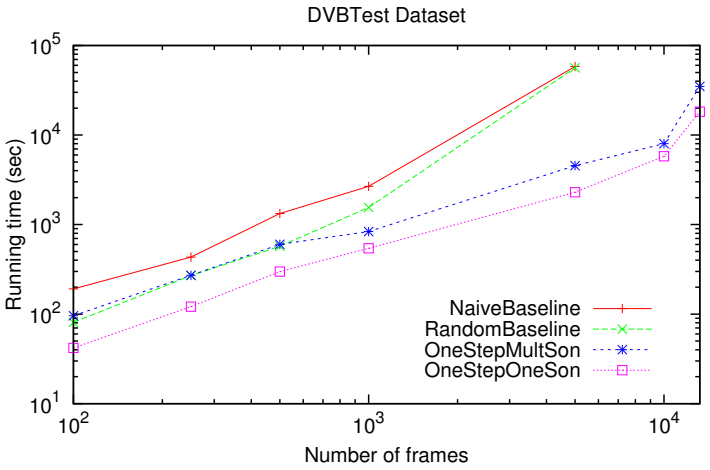
cation execution and device control. This trace has a size of 1.2 Gigabytes, contains 13,224 frames for a total of 18,208,938 events.

3.3.2 Comparison of scalability

Fig. 3.7 reports the wall clock time of the four algorithms presented in Section 3.2, when varying the number of frames given as input. Each point represents the average of 10 executions.



(a) Gstreamer trace



(b) DVBTest trace

Figure 3.7 – Running Time

One can notice that both `OneStepMultSon` and `OneStepOneSon` are always faster than `NaiveBaseline` and `RandomBaseline`. For the GStreamer dataset, both `OneStep` approaches are one order of magnitude faster than `RandomBaseline` and two orders of magnitude faster than

`NaiveBaseline`. For the DVBTest dataset, the difference is less important for small number of frames, but quickly jumps to more than one order of magnitude for 5,000 frames. Note that in both datasets, the baseline approaches could not output results for more than 5,000 frames even after more than 10 hours of computation. This comes from the much bigger search space that they have to explore. On the other hand both `OneStepMultSon` and `OneStepOneSon` can output results even for the 131,340 frames of the GStreamer dataset within 3 hours. This makes them more suitable for analysis of real traces.

3.3.3 Comparison of coverage

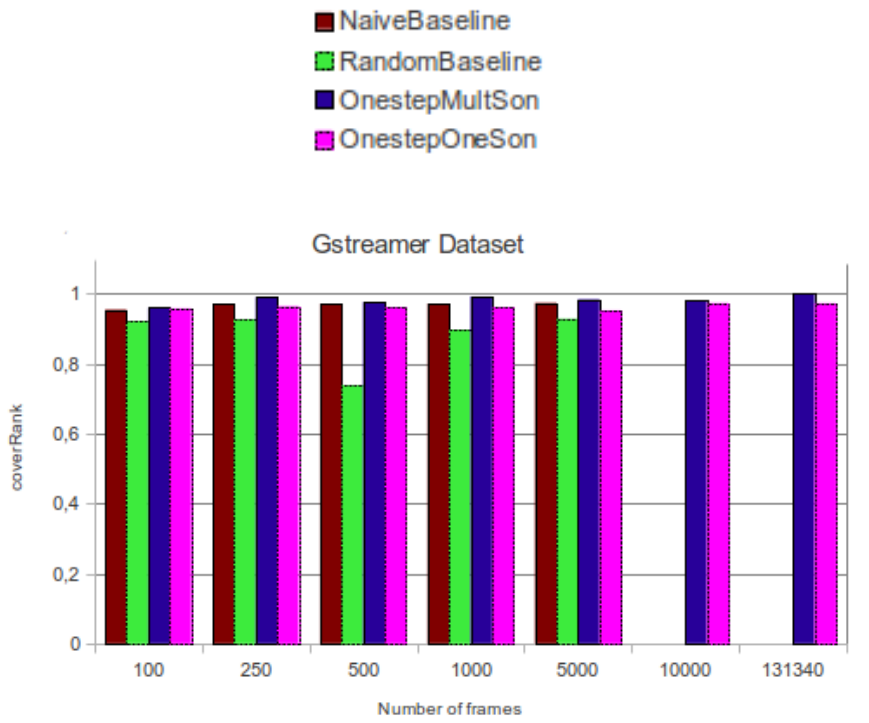
Fig. 3.8 shows the coverage of the set of blocks obtained, w.r.t. the number of frames given as input.

The first observation from the DVBTest dataset is that the coverage value of the solutions given by all approaches decreases with the number of frames given as input. The reason is that we fixed $k = 10$, which is small and thus prefers blocks that appear in a many frames, i.e. with large support. The frames in this dataset tend to have many events with some variety between the frames, especially because of system-level events. For small number of frames, interesting frequent blocks with good coverage can be found. However with more frames, blocks with very high support tend to have small size and thus small coverage.

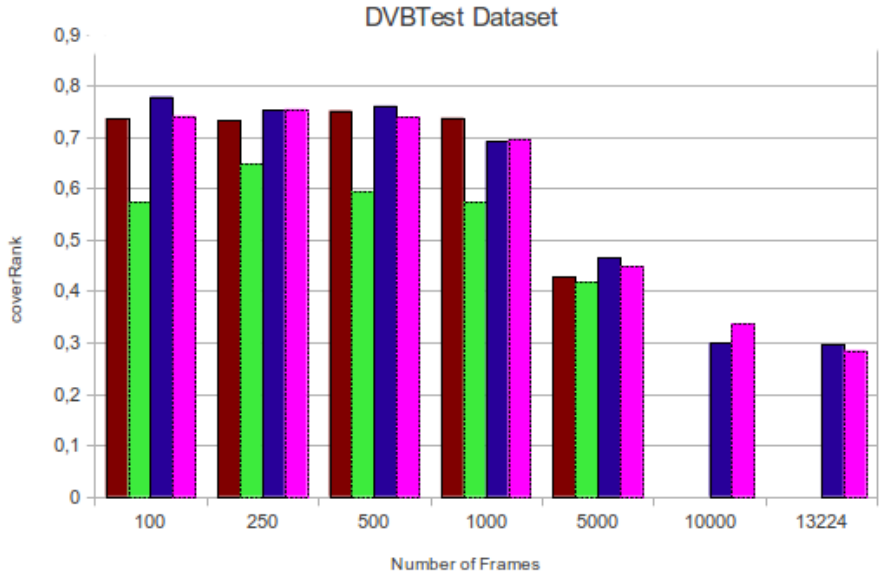
Oppositely, in the GStreamer dataset, there are only application level events, leading to frames with less events and less inter-frame variability. Thus the coverage values for this dataset stay high whatever the number of frames considered.

When comparing the approaches, one can notice that the random selection of $PatPool$ in `RandomBaseline` does not give good results, as this approach has the lowest coverage of all. On the other hand, both `OneStep` approaches achieve coverage results similar to `NaiveBaseline` even if they don't have access to as many candidate blocks. This validates the advantage of our iterative greedy algorithm approach.

Lastly, the `OneStepOneSon` approach, which generates smaller $PatPool$ than `OneStepMultSon`, achieves similar coverage results. This indicates that few well selected patterns in $PatPool$ are enough to allow the greedy algorithm to find a good solution, and that the selection of this pattern can be done with aggressive pruning compared to traditional pattern mining methods. In summary, `OneStepOneSon` has the best tradeoff between the coverage and the computation time, as it presents the best computation time and near best coverage values.



(a) Gstreamer trace



(b) DVBTest trace

Figure 3.8 – Coverage

3.3.4 Practical trace analysis

The previous experiments showed that the methods we proposed can scale to real application traces, and allow to find most representative blocks. We now present how such blocks can be of interest for application developers.

A first simple point is information reduction. In the case of the GStreamer dataset, here reduced to its first 100 frames, usually a developer would have to analyze manually or with graphical tools a trace having 3,915 events. When rewriting a trace using blocks, each event subsequence of a block embedded in the trace is replaced by an event symbol representing the block in the trace. The non replaced subsequence between two blocks could also be regarded as an extra block. Rewriting the Gstreamer trace, using 10-most representative blocks (10-MRB) extracted by one of our algorithms leads to a trace of 320 embeddings of blocks, which gives a 92% reduction factor.

Such rewriting is easier to represent graphically than the original trace. Consider Fig. 3.9 which shows a rewriting of the 50 first frames of the GStreamer dataset.

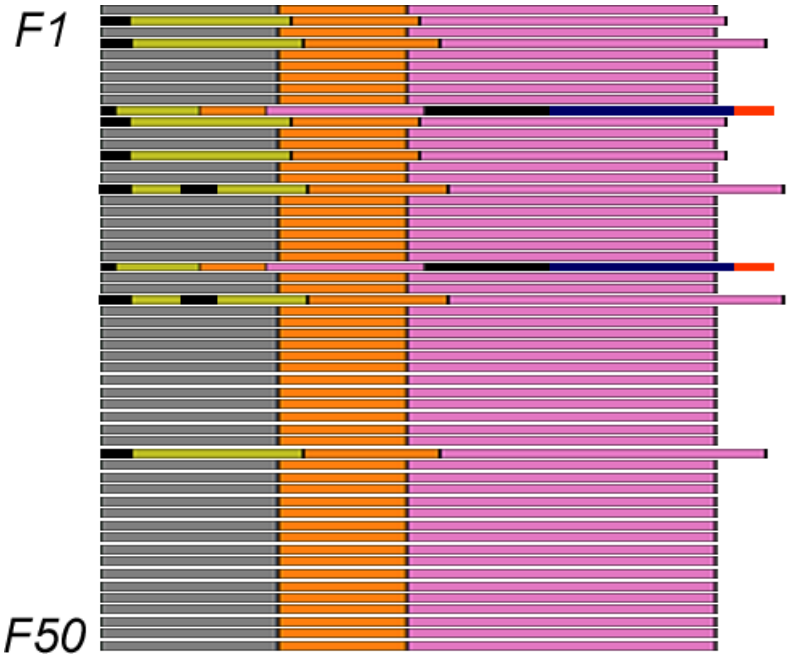


Figure 3.9 – Global View

The frames are the horizontal lines in the picture. Each frame is composed of blocks represented as rectangles, where each of the 10-MRB has a different shade of grey and the parts of the frame not covered by the blocks are in black. The length of a block corresponds to the number of events in this block. One can notice that most frames have similar numbers of events, except for some of them having more events. A developer can quickly notice two things with this representation: first, the regular structure of computation of the frames is exhibited by the regular sequencing of the blocks across the frames. Especially, the middle

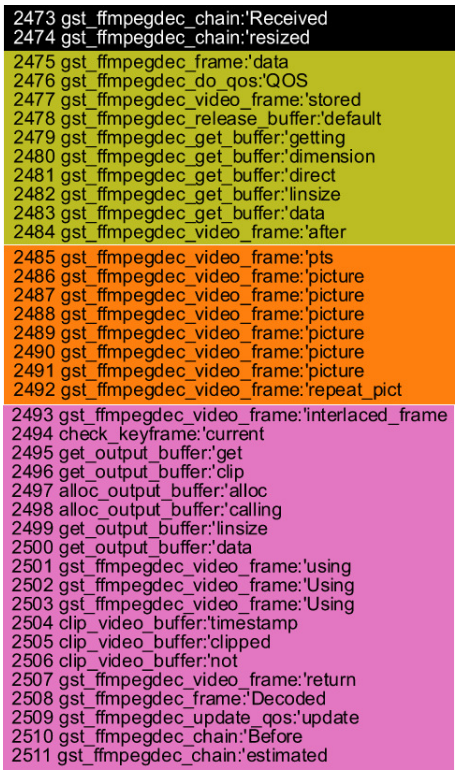


Figure 3.10 – Blocks of fourth frame

and end parts of the frames is very regular and should not require too much attention to be paid. Second, some irregularities can quickly be spotted, either by not covered parts of the trace or by MR-blocks that do not appear as often as the others. The developer can quickly check that these irregularities appear mostly at the beginning of frames. MR-blocks arising in these irregularities can give good hints of what is going on, and suggest that the irregularities they participate in are not anomalies but more likely operations that do not need to appear in all frames. Not covered sections (in black in Fig. 3.9) on an other hand, may be beneficial for the developer to investigate.

Fig. 3.10 shows a detailed view of the fourth frame, which has an uncovered region at its beginning. The figure shows the frame is rewritten with MR-blocks.

For convenience, the events are indicated inside the blocks on this figure. The developer can quickly identify in the uncovered region a rare call to the function `gst_ffmpegdec_chain:'resized`. Such call appearing after receiving new data, means that it is necessary to resize the buffer. However, this operation is usually unneeded, as buffers are supposed to be of sufficient size for handling frame data. By knowing memory operations being critical, the developer, without looking at the whole frame, immediately understands that he has to investigate if this buffer resizing caused problems or not.

To summarize, the MR-blocks allow to rewrite the frame as a sequence of blocks of limited size, which is much more manageable than a large sequence of events. Such sequence of

blocks can for instance easily be displayed by graphical tools, and shows irregular parts of the traces. The developer can then delve into the analysis of a single frame, and in last resort check the events arising at some point of this frame. This approach allows him to quickly pinpoint possible problems.

3.4 Conclusion

In this chapter, we have presented the problem of finding a small set of representative blocks of events that can maximally cover an execution trace of a multimedia application. This problem is a variant of the packing problem, a NP-hard problem for which no polynomial time algorithm is known. We thus presented a baseline approach and several greedy approaches, and showed experimentally that our best approaches scale well to real application traces up to gigabyte size.
We presented a detailed case study on how to analyze a trace with such representative blocks. Our approach allow to drastically reduce the quantity of information a developer has to handle, and is appropriate for graphical visualization. We show that by visualizing a rewritten trace a developer can spot unusual behaviors in the trace with few operations, and understand the reason of such behavior. We think that this approach is promising to help application developers in their everyday debugging or optimization tasks. It is generalizable on other problems such as automatic log analysis or system events analysis, which do not have equivalent notion of frames.

4 A dissimilarity-based comparison method to analyse event sequences

In this chapter, we propose techniques to detect anomalies in multimedia applications, using execution traces. We argue that, in the context of execution traces,using visualization tools to state an hypothesis on application debugging is not satisfactory, because of the amount of data to represent. We propose to automatically provide a diagnosis by comparing two execution traces, the first one is a reference trace corresponding to a good behaviour and the second one the execution trace to analyse. We use dissimilarity based models and conduct a user case to show how TED, our automatic trace diagnosis tool, provides added-value information to the developer. Performance evaluation over real world data shows that our approach is scalable.

Contents

Context

The analysis of execution traces is at the core of the optimization and debugging of applications. Multimedia applications traces size make their exploitation very complex, in particular for outliers detection. Very often in the domain of embedded systems, a reference trace produced by a simulator is available. It is then possible to use a semi-supervised anomaly detection method [CBK12], for which a training (or reference) database containing only *normal* sequences is assumed, and a test sequence is tested against the reference database. Many techniques could be used such as *similarity-based techniques*, *window-based techniques*, *Hidden Markov Model-based techniques* and *Markovian techniques* (see Chap. 2 for more details). We will design methods to quickly find anomalies by comparing an execution trace with a reference trace (without anomalies), using a suitable dissimilarity measure. This technique refers to *similarity-based techniques* whose strength is the possibility to use new similarity (or dissimilarity) measure for sequences, and hence to devise a particular anomaly detection which is best suited for a given problem. These techniques also give opportunity to use existing dissimilarity measure if appropriate. However, although there is an abundant literature about dissimilarity measures on sequences ([Mör06],[TAG07],[BHR00]), very few dissimilarity measures take into account the temporal aspect that is crucial in execution traces. More generally, designing an appropriate dissimilarity measure for a meaningful comparison between multimedia sequences is a difficult task. Indeed, it requires to capture and combine within a single numerical function, several aspects that are specific to such execution traces. Whatever the quality of a dissimilarity measure for suggesting the existence of a bug in an execution trace, by a comparison with a reference trace, the results of the dissimilarity measure calculation are inherently difficult to interpret by human developers, in particular for finding the actual cause of the bug.

Contributions

In this chapter, we propose to replace a black-box approach encapsulated in a single complex dissimilarity measure, by a glass-box approach based on a fine-grained analysis of problems that are likely to occur in multimedia applications. The idea is that anomalies in multimedia applications usually have visible effects such as the desynchronization of sound with the picture or subtitles, the interruption of a video streaming or the loss of some frames (a frame being an image rendered during a known time interval). Hence, it is important to identify and categorize anomalies.

First, we have identified a family of anomalies likely to occur in multimedia applications and that are visually perceptible when a user is watching a video.

Secondly, we have chosen the three most common types of anomalies, and for each type, we have designed a specific dissimilarity score which measures appropriately the amplitude of the corresponding anomaly. Based on these dissimilarity measures, we have designed a diagnosis tool able to detect degraded execution traces and to point out areas where to find

possible causes of such a degraded behaviour. We could later treat other types of anomalies.

4.1 Dissimilarity-based diagnosis: problem statement and general approach

The trace diagnosis problem that we address can be stated as follows:
Given a set $\{P_1, P_2, \ldots, P_l\}$ ***of types of anomalies, given an execution trace*** T_s ***and a reference trace*** T_r***, detect whether*** T_s ***contains anomalies, and if it is the case, determine their types among*** $\{P_1, P_2, \ldots, P_l\}$**.**

This problem requires to identify and categorize the most common anomalies in audio/video streaming.

Section 4.2 reports the methodology that we have followed and resulting in the identification of 9 types of anomalies.

Our approach to solve the diagnosis problem has been to propose dissimilarity measures specifically designed to identify presence of each type of anomaly in a trace by comparison with a reference trace. More precisely, for each type of anomaly A, we have designed a dissimilarity measure d such that:

• For each video decoding presenting a visible anomaly of type A and generating a trace T_s, for a reference trace T_r corresponding to a correct decoding of the video, $d(T_r, T_s) \neq 0$.

• For each video decoding without any visible anomaly of type A and generating a trace T_s, for a reference trace T_r corresponding to a correct decoding of the video, $d(T_r, T_s) = 0$.

A positive value of dissimilarity measure gives an insight on the amplitude of the anomaly. The biggest the value is, the further the trace is from the reference trace (and the heaviest the anomaly is). A *reference* trace is an error-free trace that can be obtained by a simulator. We have focused on three anomalies.

Section 4.3 presents the dissimilarity measures that we have designed for each of them.

Section 4.4 presents the TracE Diagnosis (TED) tool that implements our dissimilarity-based diagnosis approach, and presents a user friendly interface to help developers.

Section 4.5 reports on the qualitative and quantitative experimental evaluation that we have conducted on our method.

Section 4.6 presents an approach to apply our dissimilarity measures on reduced size traces.

4.2 Our categorization of anomalies in audio/video decoding

We have shown in Chapter 1 how to generate execution traces while streaming a video. Note that some common anomalies can occur in such traces, due to the video decoding application. These anomalies are generally revealed by multimedia application users, and to the best of our knowledge, there is no existing study that provides a categorization of them.

Our methodology has been the following. In order to collect well-known types of anomalies, we have visited and analysed 10 websites of Audio/Video (A/V) developers community: [cp13],[Wf13a], [ha14], and discussion forums of multimedia application users: [Gf13], [Ff13], [Wf13b], [Nf13], [ms13], [wA13], [Hf13].

As much there exist websites for A/V developers, as much there could be many formulations of the same anomaly. Users use their own terms to describe a problem. For example, statements found in [Ff13] such that *"There is a lag with audio (...)"*, and *"I wait a couple of seconds to see corresponding subtitles (...)"* reflect the same anomaly, Audio/Video/Subtitle desyncronization (A/V/S desync). Another user formulation *"My movie stopped after few minutes and I started player again"* ([Wf13b]) and *"the video stops during playback"* ([Gf13]) corresponds to "Player crash" in ([cp13]).

We have grouped different formulations into 9 types of anomalies. Such a categorization has helped us to identify common problems occurring in video streaming.

Table 4.1 summarizes the 9 types of anomalies and their occurrence in the 10 websites of multimedia applications developers and multimedia applications users that we have analyzed. They have visual and sound effects on the video streaming.

Table 4.1 – Common Audio/Video decoding problems.

Anomalies	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A/V/S desync.	×	×					×	×	×	
Black screen	×									×
Overlay Mixer	×									
Video bug	×									
Wrong colors	×									
Double image	×						×			
Player crash	×	×	×		×		×	×		
Stuttered playback			×			×		×		
Slow Streaming	×			×			×	×		

[1]= ([cp13]) [6]= ([Nf13])
[2]= ([Wf13a]) [7]= ([ha14])
[3]= ([Gf13]) [8]= ([ms13])
[4]= ([Ff13]) [9]= ([wA13])
[5]= ([Wf13b]) [10]= ([Hf13])

We have focused on the three most common errors that a developer encounters in his video players. We consider as most common errors those which appear at least 4 times for all forums. They can be simulated using existing tools that are able to inject those perturbations. According to Tab 4.1, these anomalies are:
P_1**: Audio/video/subtitle desynchronization anomaly**: This anomaly reflects a desynchronization in time between audio, video or subtitles. The audio may be slower than the video or the subtitle may not appear at the right moment.
P_2**: Player crash anomaly**: The player stops abruptly at a random execution time, without any reason.
P_3**: Slow streaming anomaly**: Visually, video is very slow. In this case the audio/video/subtitles are synchronized but take much more time than in a normal execution.

4.3 Our proposal for specific dissimilarity measures

This section explains our general approach for solving the trace diagnosis problem stated in the previous section, using an appropriate dissimilarity measure for each type of anomaly.

4.3.1 Preliminaries

A dissimilarity d between two objects is a numerical measure of how far apart these objects are [PNSK+06]. It should be efficiently computable [MR97]. The term *distance* is frequently used as a synonym of *dissimilarity* but the term distance is used to refer to a special class of dissimilarities, which satisfies following requirements:

For all T_1, T_2 and T_3,

$$\begin{cases} d(T_1,T_2) \geq 0 \; and \; d(T_1,T_2) = 0 \quad only \; if \quad T_1 = T_2 \\ d(T_1,T_2) = d(T_2,T_1) \\ d(T_1,T_2) + d(T_2,T_3) \geq d(T_1,T_3) \end{cases} \tag{4.1}$$

There is no "magic" dissimilarity formula to capture difference between two objects. As we saw in Chapter 2, Section 2.1, dissimilarity measures depend among others on the nature of the data (symbolic sequences, numeric sequences,..), the application domain (telecommunications, biology, education,..), and the anomalies. If a technique is effective for an anomaly, it does not mean that it will be effective for another type of anomaly.
In our case, instead of defining a single dissimilarity measure as a black-box that encapsulates the specificities of various anomalies constraints, we propose a glass-box approach through multiple dissimilarity measures that are appropriate to the types of anomalies we want to detect.

The procedure that we have followed to define a specific dissimilarity measure for each type of

anomaly can be summarized as follows. First, we obtain a reference trace by decoding a movie video with $gstreamer$ [1]. Then, we inject in the streaming, perturbations corresponding to a given type of anomaly and we obtain the corresponding abnormal execution traces. This process is repeated a certain number of times. Finally, for each type of anomaly, we use statistic methods to analyze the reference trace and the execution trace, and extract the meaningful differences that have to be incorporated in each dissimilarity measure.

As a result, we have designed three dissimilarity measures. The first one is the *occurrence dissimilarity*, suitable for detecting an anomaly of type P_1 (Audio/video/subtitle desynchronization anomaly). The second measure is the *dropping dissimilarity*, appropriate to identify anomalies of type P_2 (Player crash anomaly). Finally, we introduce the *temporal distance* designed to detect anomalies of type P_3 (Slow streaming anomaly). For each distance, we give a formal definition and an algorithm for its computation.

4.3.2 Occurrence dissimilarity

For P_1 anomaly, when examining the traces, one can detect different numbers of occurrences of some events in the simulated trace and the abnormal one. It is the only difference in this case.
We first define the *occurrence ratio* of an event in two traces.

Definition 7. *Let T_1 and T_2 be two execution traces. Let $nb\_occur(e,T)$ be the number of occurrences of event e in trace T. The* occurrence ratio *of an event e in the two traces T_1 and T_2 is defined as follows:*

$$occ\_ratio(e,T_1,T_2) = \frac{Min\{nb\_occ(e,T_1), nb\_occ(e,T_2)\}}{Max\{nb\_occ(e,T_1), nb\_occ(e,T_2)\}} \tag{4.2}$$

Note that e should appear in both traces. A value of $occ\_ratio(e,T1,T2)$ close to zero, means that event e occurs in one of the two traces much more frequently than in the other one. Such a situation is related to an anomaly P_1 because a desynchronization in time between audio, video and/or subtitles induces many abnormal occurrences of events.
That is why we define the *occurrence dissimilarity* between two traces as the number of events that have an occurrence ratio less than or equal to a given threshold (the threshold value has been experimentally determined, see further Section 4.5.2 for details). This dissimilarity measure is appropriate to retrieve P_1, A/V/S desync. anomaly, (see section 4.1) because it measures the number of events that differentiate T_1 from T_2. The formal definition of this dissimilarity measure, thereafter denoted d_o is the following:

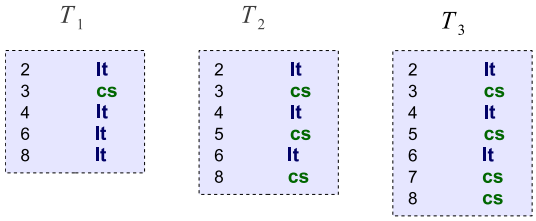
Definition 8. *Let T_1 and T_2 be two execution traces. The* **occurrence dissimilarity** *between T_1 and T_2 is:*

$$d_o(T_1,T_2) = |\{e \mid occ\_ratio(e,T_1,T_2) \leq \theta\}| \tag{4.3}$$

[1] Procedure is detailed in Section 4.4.3

where θ is a given threshold.

Example 1. *Consider the traces T_1, T_2 and T_3 below, and let $\theta = 0.5$.*
$occ\_ratio(It, T_1, T_2) = 3/4 = 0.75$, which is greater than the threshold θ (= 0.5).
$occ\_ratio(CS, T_1, T_2) = 1/3 = 0.33$, which is less than the threshold θ.
Only one event has an occurrence ratio less than θ, then $d_o(T_1, T_2) = 1$.
$occ\_ratio(It, T_2, T_3) = 3/3 = 1$, $occ\_ratio(CS, T_1, T_2) = 3/4 = 0.75$ and $d_o(T_2, T_3) = 0$.



▲

4.3.3 Dropping dissimilarity

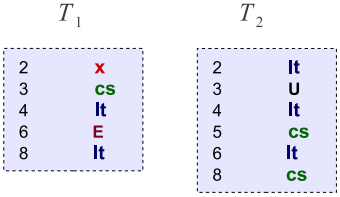
For P_2 anomaly, when comparing the simulated and abnormal traces, we found that some events seem to appear only in one trace and not in the other one.
The corresponding *dropping dissimilarity* refers to the number of distinct events that belong only to one trace. This dissimilarity measure is also used by [WS09] as *mismatch score* on temporal categorical records.

Definition 9. *Let* events(T) *be the set of distinct events in T. The* **dropping dissimilarity** *between T_1 and T_2 is the size of the symmetric difference between* events(T_1) *and* events(T_2).

$$d_d(T_1, T_2) = |events(T_1) \triangle events(T_2)| \tag{4.4}$$

This dissimilarity measure is appropriate to retrieve P_2, i.e. Player crash anomaly (see 4.1).

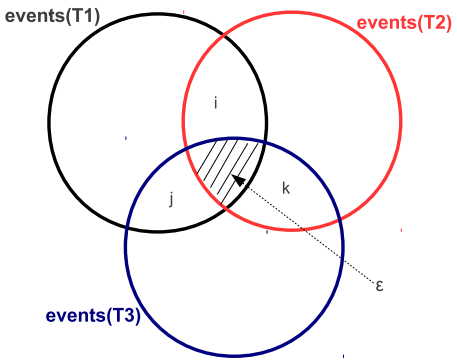
Example 2. *For traces T_1 and T_2 below:*
$events(T_1) = \{X, CS, It, E\}$, $events(T_2) = \{CS, It, U\}$.
$events(T_1) \triangle events(T_2) = \{X, E, U\}$.
The set $\{X, E, U\}$ contains 3 events, then $d_d(T_1, T_2) = 3$.



▲

Proposition 1. *Dropping dissimilarity is a distance.*

Proof. Let T_1, T_2 and T_3 three traces. $events(T_1)$, $events(T_2)$ and $events(T_3)$ are illustrated below.



1. **Positivity**

$$\begin{aligned} d_d(T_1,T_2) &= |events(T_1) \triangle events(T_2)| \\ &= |events(T_1) \setminus events(T_2)| + |events(T_2) \setminus events(T_1)| \end{aligned}$$

$\forall\ T_1, T_2,\ |events(T)| \geq 0 \Rightarrow |events(T_1) \setminus events(T_2)| \geq 0$
hence $|events(T_1) \setminus events(T_2)| + |events(T_2) \setminus events(T_1)| \geq 0$
then $d_d(T_1,T_2) \geq 0$

2. if $T_1 = T_2$,

$$\begin{aligned} d_d(T_1,T_1) &= |events(T_1) \setminus events(T_1)| + |events(T_1) \setminus events(T_1)| \\ &= |\emptyset| + |\emptyset| \\ &= 0 \end{aligned}$$

3. **Symmetry**

$$\begin{aligned} d_d(T_1,T_2) &= |events(T_1) \setminus events(T_2)| + |events(T_2) \setminus events(T_1)| \\ &= |events(T_2) \setminus events(T_1)| + |events(T_1) \setminus events(T_2)| \\ &= d_d(T_2,T_1) \end{aligned}$$

4. **Triangle Inequality**: Considering the previous illustration,

$$\begin{aligned} d_d(T_1,T_2) &= |events(T_1) \setminus events(T_2)| + |events(T_2) \setminus events(T_1)| \\ &= |events(T_1)| - |events(T_1) \cap events(T_2)| + |events(T_2)| - |events(T_1) \cap events(T_2)| \\ d_d(T_2,T_3) &= |events(T_2)| - |events(T_2) \cap events(T_3)| + |events(T_3)| - |events(T_2) \cap events(T_3)| \end{aligned}$$

$d_d(T_1,T_2) + d_d(T_2,T_3) = |events(T_1)| + |events(T_3)| + 2|events(T_2)| - 2|events(T_1) \cap events(T_2)| - 2|events(T_2) \cap events(T_3)|$

$$\begin{aligned} d_d(T_1,T_2) + d_d(T_2,T_3) - d_d(T_1,T_3) &= |events(T_1)| + |events(T_3)| + 2|events(T_2)| \\ &\quad -2|events(T_1) \cap events(T_2)| - 2|events(T_2) \cap events(T_3)| \\ &\quad -|events(T_1)| - |events(T_3)| + 2|events(T_1) \cap events(T_3)| \\ &= 2|events(T_2)| - 2|events(T_1) \cap events(T_2)| \\ &\quad -2|events(T_2) \cap events(T_3)| + 2|events(T_1) \cap events(T_3)| \quad (*) \end{aligned}$$

but $|events(T_1) \cap events(T_3)| - |events(T_2) \cap events(T_3)| = j - k$, and
$|events(T_2)| - |events(T_1) \cap events(T_2)| = \epsilon + k$

$$\begin{aligned} (*) &= 2(\epsilon + k + j - k) \\ &= 2(\epsilon + j) \\ &\geq 0 \end{aligned}$$

Then $d_d(T_1,T_2) + d_d(T_2,T_3) - d_d(T_1,T_3) \geq 0$
$\Rightarrow d_d(T_1,T_2) + d_d(T_2,T_3) \geq d_d(T_1,T_3)$

□

4.3.4 Temporal distance

For P_3 anomaly, the duration and the order of some events differ in the two traces. In the abnormal trace, some events durations are much longer than durations of same events in the simulated trace.
The temporal distance that we propose is an adaptation of the distance model of [MR97]. It is an edit-distance taking into account temporal aspects. It uses three basic operations:

- *Ins(e,t)* that inserts an event e at time t
- *Del(e,t)* that deletes an event e at time t
- *Move(e,t,t')* that moves an event e from time t to time t'.

A cost $c(o)$ is associated with each operation o. There are several ways of defining edit operation costs. Traditionally, a unit cost is used for each basic operation. Ronkainen in [Ron98] thought that it might be more natural to adapt the cost depending on the event type. The $Move$ operation should have a time depending cost which reflects changing in time.
$c(Move(e,t,t')) = V|t'-t|$ where V is a constant such that $V \leq 2.w(e)$.
Without this condition, it would always be better to do a deletion and an insertion of an event e, instead of moving e from t to t'. $|t'-t|$ represents the length of the move. Short moves have lower cost than long moves.
The cost of a sequence of operations can then be deduced. Let $O = o_1 \dots o_k$ be a sequence consisting of k transformations. The cost of O is:

$$c(O) = \sum_{i=1}^{k} c(o_i) \tag{4.5}$$

The temporal distance $d(T_1, T_2)$ proposed by [MR97] is defined as the cost of the cheapest sequence of operations that transform T_1 into T_2.

Definition 10. *if* Θ *is the set of operation sequences that transform* T_1 *into* T_2*, then the* **temporal distance** *between* T_1 *and* T_2 *is:*

$$d(T_1, T_2) = \underset{O \in \Theta}{Min}\, c(O) \tag{4.6}$$

Example 3. *For traces* T_1 *and* T_2 *below, between all the possible sequence of operations, the cheapest order-preserving sequence of operations that transform* T_1 *into* T_2 *is:*
$O = \langle Move(It,2,1), Move(It,4,2), Ins(U,5)\rangle$; $c(O) = 3V + w(U)$.

T_1

2	It
4	It
7	cs

T_2

1	It
2	It
5	U
7	cs

▲

Let $T_1 = (e_1, \dots, e_n)$ and $T_2 = (f_1, \dots, f_m)$ be two execution traces, and let $r(i,j)$ denote the minimum cost of the operations needed to transform the first i events of T_1 into the first j events of T_2. the temporal distance between T_1 and T_2 is:

$$d(T_1, T_2) = r(n,m) \tag{4.7}$$

where $r(i,j)$ is computed according to the following dynamic programming algorithm:

$$\begin{cases} r(0,0) &= 0 \\ r(i,0) &= r(i-1,0)+w(e_i) \\ r(0,j) &= r(0,j-1)+w(f_j) \\ r(i,j) &= min\{\, r(i-1,j)+w(e_i), r(i,j-1)+w(f_j), r(i-1,j-1)+cost(i,j)\} \end{cases} \tag{4.8}$$

$w(e_i)$ is the cost of deleting event e_i at position i. $w(f_j)$ the cost of inserting event f_j at position j and

.

$$cost(i,j) = \begin{cases} w(e_i)+w(f_j) & if\ e_i \neq f_j \\ V.|t_i - t'_j| & if\ e_i = f_j \end{cases} \tag{4.9}$$

t_i is the timestamp of e_i and t'_j, the timestamp of event in f_j.

We have adapted this temporal distance and the algorithm in order to firstly, adapt the weight to the context of multimedia execution traces and secondly, consider the magnitude of the two traces. The magnitude refers to the beginning timestamps of the two traces which are not always the same.
In a reference trace, some types of events occur more often than others. They are considered as less important. For a given event e, the cost $(w(e))$ of inserting or deleting e is taken as a constant value proportional to the occurrence number of e in the reference trace T_r: $w(e) = \frac{1}{nb\_occur(e,T_r)}$.

Because of the magnitude of the two traces, results obtained with the method used in [MR97] are not satisfactory. We illustrate this problem in example 4.

Example 4. *For the two traces below,* $d(T_1,T_2) \neq 0$.

T_1

1	lt
2	U
4	cs

T_2

11	lt
12	U
14	cs

▲

In the previous example, considering $d(T_1,T_2) \neq 0$ is not satisfactory because T_1 and T_2 have exactly the same events, and the same time intervals between events. Clearly, such traces should be considered as similar. Therefore, we propose to adapt the *Mannila and Ronkainen* [MR97] distance model in order to have $d_3(T_1,T_2) = 0$ when T_2 is obtained from T_1 by a time shift.
More precisely, we adapt the $cost(i,j)$ computation by integrating beginning timestamps of each execution trace into the formula. We therefore consider:

$$cost(i,j) = \begin{cases} w(e_i) + w(f_j) & if\ e_i \neq f_j \\ V.||t_i - t'_j| - |t_0 - t'_0|| & if\ e_i = f_j \end{cases} \tag{4.10}$$

t_0 is the timestamp of the first event in T_1 and t'_0, the timestamp of the first event in T_2.

The **temporal distance** between T_1 and T_2 that we use in the following is:

$$d_t(T_1, T_2) = r(n,m) \tag{4.11}$$

where $r(i,j)$ is computed according to equation 4.8 and $cost(i,j)$ is given by equation 4.10.

Example 5. *Consider Example 4 where the two execution traces do not have the same magnitude. Table 4.2 is the dynamic programming table used to compute temporal distance between both traces, assuming that* $w(e) = 1$ *i.e insertion operation and deletion operation have a unit cost.*

Table 4.2 – Dynamic programming applied to an example

r(i,j)	**j**	0	1	2	3
i			It	U	CS
0		0	1	1	1
1	It	1	0	1	2
2	U	1	1	0	1
3	CS	1	2	1	0

$r(3,3)$ *is the temporal distance between* T_1 *and* T_2*; thus* $d_3(T_1, T_2) = 0$.

▲

This distance is appropriate to retrieve P_3, i.e. slow streaming anomaly (see subsection 4.2), because it takes into account events duration and sequentiality between events.

Remark 1: The temporal distance is constructed as editing distances, thus respect all properties of a distance [Cor03].

Remark 2: In the rest of this chapter, we generalize and use the terms distance and dissimilarity as synonyms for all our measures: *occurrence distance*, *dropping distance* and *temporal distance*.

Remark 3: All the distances have been experimentally validated (see Section 4.5). Each distance allows to detect the specific visible anomaly for which it has been designed.

4.3.5 Measure normalization and complexity

For each dissimilarity measure defined above the output is a value in $\mathbb{R}^+$. This value allows the developer to appreciate the amplitude of a perturbation between several abnormal traces. The dissimilarity value can be a large number which depends on trace size. Developers can have difficulties to compare different distance values. In order to better interpret the results, it is important to normalize the output. We use a non-linear transformation g, in order to normalize the dissimilarities:

$$\begin{array}{rcccl} g & : & \mathbb{R}^+ & \rightarrow & [0,1] \\ & & d & \mapsto & d/(1+d) = g(d) \end{array}$$

Computation of *occurrence distance* and *dropping distance* are done in linear time complexity since a simple scan of traces is necessary. With the dynamic programming algorithm presented above, the computation of *temporal distance* has a quadratic complexity $O(m \times n)$, where m and n are the lengths of the two traces. Assuming $n \geq m$, Wu et al. in [WMMM90] proposed some improvements with a $O(np)$ time complexity, where $p = D/2 - (n-m)/2$ and D being the length of a shortest edit script (consisting of insertions and deletions) between the two sequences to compare.

4.4 TED: the execution TracEs Diagnosis tool

After identifying an anomaly type, a remaining question is: *how to locate it into the execution trace?* Our answer is: by decomposing execution trace and by applying dissimilarities on parts of trace. We show this process in Section 4.4.1. We then describe TED, our TracE Diagnosis tool (Section 4.4.2), which integrates the computation of the three dissimilarities previously defined: occurrence dissimilarity, dropping distance and temporal distance. We illustrate its functioning on two use cases and evaluate performances.

4.4.1 Measure computation by portion of traces

It is important to emphasize that each of these dissimilarities can be computed on the entire trace, but also on portions of trace. Recall that a pipeline, constituted by different components, is used to obtain an execution trace (as described in Section 1.1.1); a portion is then a sequence of events related to a specific component. Moreover, when a dissimilarity measure computation is applied by component, the developer is easily oriented towards a specific component, according to dissimilarity values at this component. Assuming the pipeline has p components, we compute $d(T_{1j}, T_{2j})$ with $1 \leq j \leq p$.

Fig. 4.1 [2] shows audio/video pipeline used to obtain two execution traces T_1 and T_2. T_{ij} is the portion of trace T_i corresponding to component j. For instance, T_{11} is sequence of events corresponding to component $file\text{-}source$ for trace T_1. An illustration of the process in a use

[2] *credits figure of pipeline to ([Gst14])*

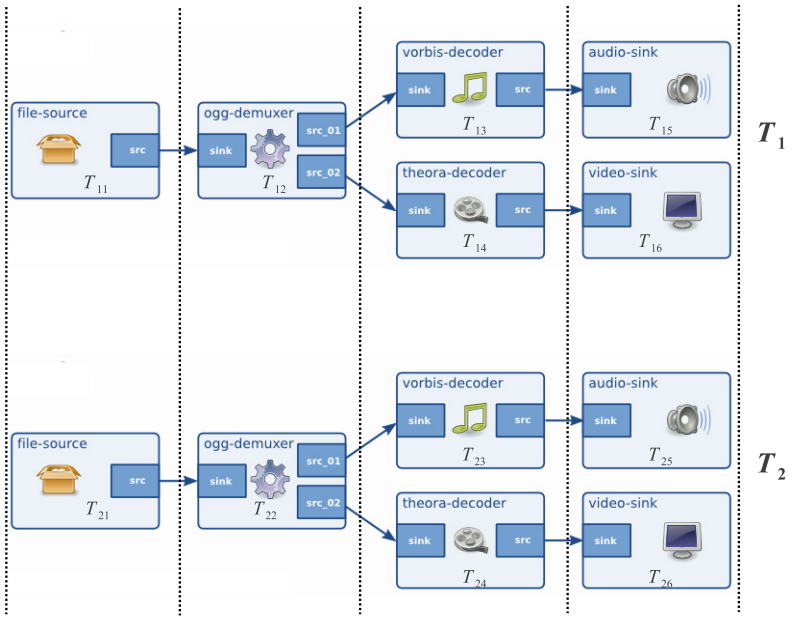


Figure 4.1 – In order to detect which specific component is concerned by the anomaly, we compute $d(T_{11}, T_{21}), d(T_{12}, T_{22}), d(T_{13}, T_{23}), d(T_{14}, T_{24}), d(T_{15}, T_{25}), d(T_{16}, T_{26})$.

case is further given in Fig. 4.4 of section 4.4.3.

4.4.2 Architecture of TED

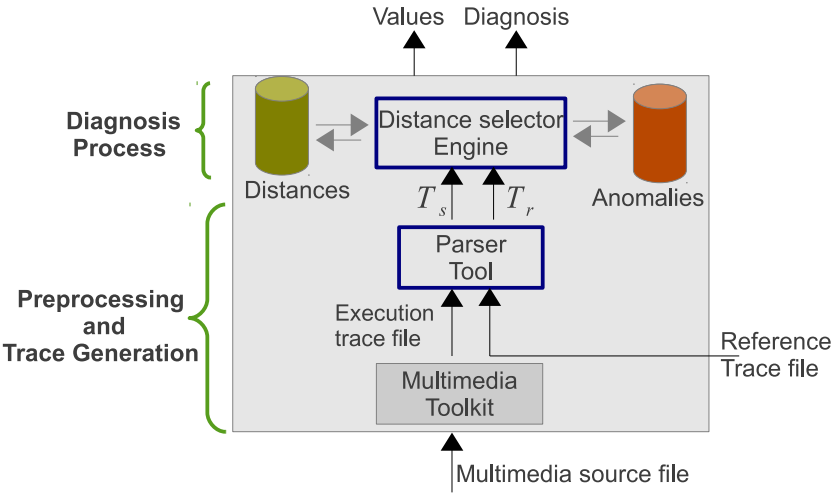
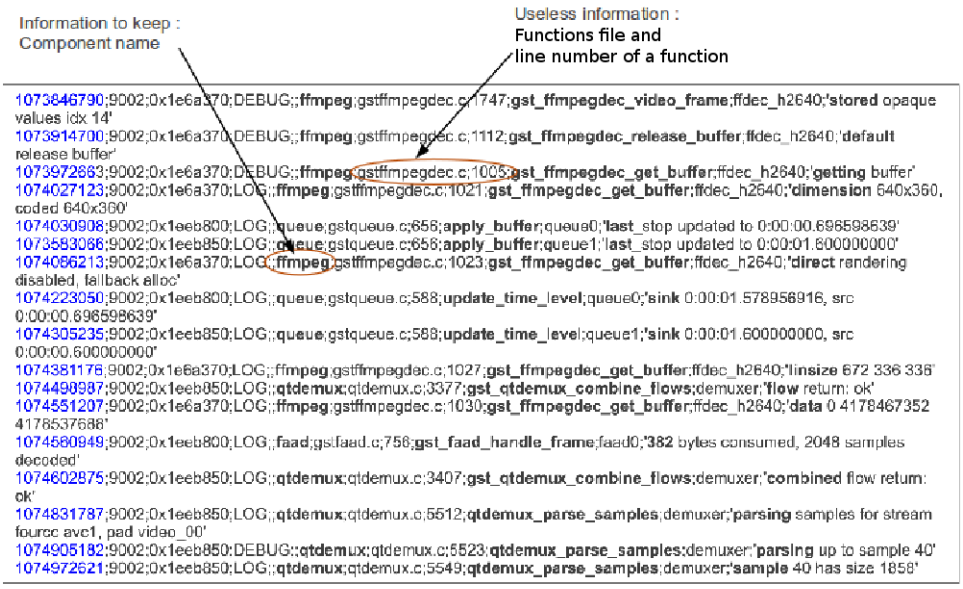


Figure 4.2 – TED Architecture

TED handles two main phases:

- The *Preprocessing and trace generation* phase takes as input a source file to generate an execution trace T (via the *multimedia Toolkit*) and a reference trace. Both are preprocessed.

This step is very important for a successful outcome of the analysis as a non cleansed and non normalized data can lead to spurious and meaningless results [Mör06]. T has the format of Fig. 4.3(a). Each entry of trace file has 11 distinct fields: *timestamp,processID, threadID, debugCategory, unknowInfo,componentName, functionsFile, line, function, object, message* [Gst14]. A parsed trace (Fig. 4.3(b)) is obtained from T by removing redundant or useless information (*processID, threadID, debugCategory, unknowInfo, functionsFile, line, object*). For instance, the *object* field is removed because it is a part of the *componentName*. The value of *processID* is the same in each line of the trace, then the field is removed.



(a) original trace

1073846790 ffmpeg:gst_ffmpegdec_video_frame :'stored
1073914700 ffmpeg:gst_ffmpegdec_release_buffer:default
1073972663 fmpeg:gst_ffmpegdec_get_buffer:'getting
1074027123 ffmpeg:gst_ffmpegdec_get_buffer:'dimension
1074030908 queue:apply_buffer:'last
1073583066 queue:apply_buffer:'last
1074086213 ffmpeg:gst_ffmpegdec_get_buffer:'direct
1074223050 queue:update_time_level:'sink
1074305235 queue:update_time_level:'sink
1074381176 ffmpeg:gst_ffmpegdec_get_buffer:'linsize
1074498987 qtdemux:gst_qtdemux_combine_flows:'flow
1074551207 ffmpeg:gst_ffmpegdec_get_buffer:data
1074560949 faad:gst_faad_handle_frame:382
1074602875 qtdemux:gst_qtdemux_combine_flows:'combined
1074831787 qtdemux:qtdemux_parse_samples:parsing
1074905182 qtdemux:qtdemux_parse_samples:'parsing
1074972621 qtdemux:qtdemux_parse_samples:'sample

(b) parsed trace

Figure 4.3 – Example of preprocessing of data: from original trace, information in bold are kept.

- The *Diagnosis process*, is the second and core phase of TED. The *distance selector engine* chooses an appropriate distance from the set of *Distances* and applies it on traces. For instance, if we want to detect a desynchronization anomaly, the *distance selector engine* applies the occurrence distance on suspicious trace T_s and the reference trace, T_r. An option allows

to apply the distance directly on the whole trace or to select portions of trace for applying distance.

4.4.3 Use cases

We consider the following scenario. A user is watching a video and (**Scenario a**) the video streaming becomes very slow or, (**Scenario b**) the sound is desynchronized with images.

In the *Preprocessing and trace generation* phase, we decode the movie with *gstreamer* to obtain the reference trace T_r. We use a gstreamer element *identity* ([Gst14]), with property *sleep-time*, to obtain a A/V/S desync. anomaly (**Scenario b**). The resulting abnormal trace is T. We generate another abnormal trace, with a slow streaming anomaly (**Scenario a**) by a stress of CPU and memory in the system. As a result of the preprocessing step, the dataset was reduced to 26,5% of its original size.

In the *Diagnosis process* phase, the developer uses TED as follows:

- The developer has an idea of the anomaly and just want to verify if his hypothesis is true or not. He selects the distance to apply, the option for using portions of traces, and TED gives the diagnosis. In Fig. 4.4, *temporal distance* is used (**Scenario a**). The developer suspects a slow streaming anomaly (P3). TED detects the anomaly and returns the value of temporal distance between the two traces per components. For instance, the distance between audioresample components is approximatively equal to 0.94736. We recall that the resulted value of distance is normalized as explained in Section 4.3.5. TED points out the *qtdemux* components of the two traces as the components with the most dissimilar events. The distance value between *qtdemux* components is 0.99995. This value gives an insight to the developer of how far is the trace from the reference trace, regarding to *qtdemux* components. The developer can compare between several abnormal traces and detects those with heaviest anomaly.

- The developer has no idea of what is happening and would like to find if there exists an anomaly in T_s. He selects the choice *find anomaly*, and TED applies successively all the dissimilarities, and stops when one of them gives a non-zero value. In **Scenario b**, the computation of distances is directly applied on the entire trace. Dropping and occurrences dissimilarities have been tested and a A/V/S desync. anomaly was detected. In Fig. 4.5 the value of distance is 0.5 and the function which does not respect the threshold for occurrence ratio is *gst_ffmpegdec_update_qos* from component $ffmpeg$.

- The developer wants to find all potential anomalies in T_s (*All tests* option in the tool). Indeed, it is possible to have simultaneously a A/V/S desync. and a player crash anomaly. for the same trace.

Figure 4.4 – execution trace with a *slow streaming anomaly*. The developer selects the distance to apply (Scenario a)

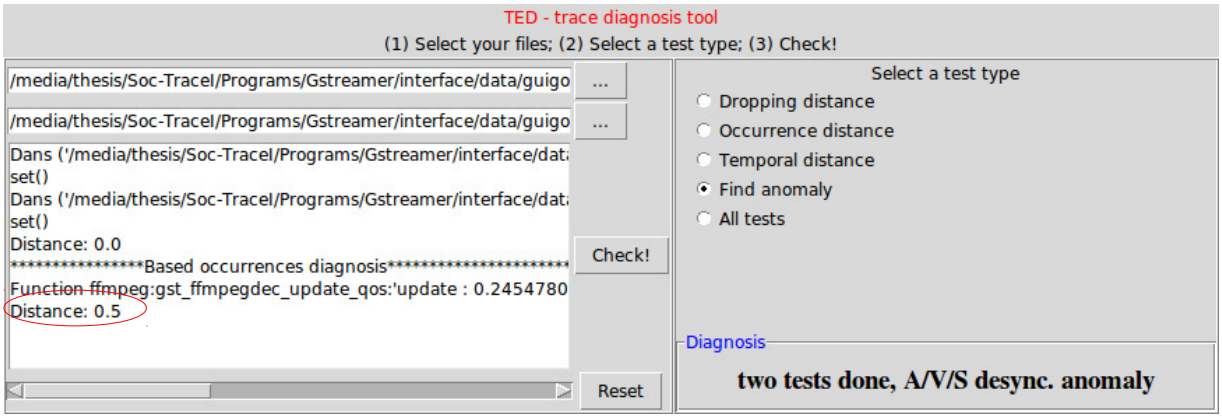


Figure 4.5 – TED finds and detects one anomaly: A/V/S desync. anomaly (Scenario b)

By using TED, a developer analyzing an execution trace is notified of anomalies, their types and where they appear in the trace (the plugin concerned). TED is a time saver for developers

as they can quickly detect anomalies in their execution traces and fix them.

4.5 Experiments

4.5.1 Experimental goals

We have conducted a set of experiments with 4 main goals:

[G1]-*Validate our proposed dissimilarity measures:* for this validation, for each type of anomaly we use a sample of suspicious traces generated by videos decoding in which an anomaly of the corresponding type may be injected. For such a trace T_s and a reference trace T_r, we then check whether $d(T_r, T_s) \neq 0$ if and only if T_s corresponds to a trace generated by an execution of video decoding in which the specified type of anomaly is visible.

[G2]-*Compare our proposed dissimilarity measures to standards sequence distances and check whether they are better discriminant.*

[G3]-*Show the interest of having specific distances for anomaly types, instead of using a given existing distance, for diagnosing the types of anomalies present in a trace.*

[G4]-*Evaluate the efficiency of our approach*: for doing this evaluation, we collect the running time of algorithms which compute proposed measures.

4.5.2 Experimental settings

► **System configuration:** Our prototype system is implemented in Python 3.2. The experiments were run on an Intel Xeon E5-2650 at 2.0GHz with 32 Gigabytes of RAM with Linux.

► **Data Set:** We use traces from two real applications, described below:

Gstreamer application: Gstreamer ([Gst14]) is a powerful open source multimedia framework for creating streaming applications, used by several corporations as Intel, Nokia, STMicroelectronics and many others. For these experiments we decoded several movies using Gstreamer on a Linux platform, with the $ffmpeg$ plugin for video decoding.
GSTapps application: It is a test video decoding application for STMicroelectronics development boards. This application is widely used by STMicroelectronics developers. The execution trace contains both application events and system-level events. It is generated from a $ST40$ core of the SoC, which is dedicated to application execution and device control.
Table 4.3 gives a description of reference traces.

► **Choice of the threshold** θ for computing occurrence distance.
In order to state a value of θ, we have used the following protocol. We did 30 decoding of a video with visible A/V/S desync. anomaly at different stage of degradation. We then obtained 30 abnormal traces ($\mathcal{T}_s$). We did 5 correct decoding of the video in order to obtain 5

Table 4.3 – Experimental dataset

	Trace	Video source duration	Nb. of events	Size
Gstreamer	*gen*	$17s$	39,646	$7.6M_o$
	pub	$49s$	74,436	$14.3M_o$
	movie	$3628s$	12,423,095	$2457,6M_o$
GSTapps	*SDK2*	$335s$	2,382,720	$73.2M_o$

reference traces ($\mathcal{T}_r$). For each pair of reference traces (T_1, T_2) $s.t.$ $T_1, T_2 \in \mathcal{T}_r$, we computed the occurrence ratio of each event (context 1). For each couple (T_1, T_3) $s.t.$ $T_1 \in \mathcal{T}_r$ and $T_3 \in \mathcal{T}_s$, we computed the occurrence ratio of each event (context 2). For each event, we compared occurrence ratio obtained in (context 1) and (context 2). We then observed that in some cases of (context 2), the gap with occurrence ratio value obtained in (context 1) is really significant: a difference of more than 50%. The threshold of occurrence ratio used for our experiments is obtained by averaging occurrence ratio values of events in (context 1). This value is set to $\theta = 0.25$ in our experiments.

4.5.3 Experimental results

Experimental validation of distances and precision of the diagnosis tool - [G1]

In order to validate the proposed distances, we considered a sample of 50 traces per anomaly types for a total of 150 abnormal traces. Each of the abnormal traces results of the decoding of a video with only one visible type of anomaly. For each comparison of an abnormal trace, we successively applied the three distances. In each case, only the distance corresponding to the observed anomaly outputs a non-zero value.

For evaluating the quality of the diagnosis done by TED, we use the standard criteria of precision and recall [HKP12].
• The *precision* is the ratio of returned results that are relevant w.r.t. all returned results.
• The *recall* is the ratio of returned results that are relevant w.r.t. all relevant results.
We run the tool on a sample of 300 execution traces (130 are normal and 170 are abnormal) as presented in Table 4.4. The first observation is that all execution traces initially considered as normal were diagnosed as such by TED. However, the tool gave 11 *false-positive* which are execution traces considered by TED as normal but which contain anomalies. Thus, TED has a precision of 0.96 and a recall of 1. A reason of this lack of precision can be the fixed value of threshold for *occurrence distance*. We will surely gain to adapt the threshold value to the length of the video decoded by finding correlation between video length and threshold value.

Table 4.4 – TED precision

Nb. traces	Original sample	TED results
Sample of 300 traces	normal: 130	normal: 141
	abnormal: 170	abnormal: 159

Comparison with standards sequence dissimilarities - [G2]

We show in this section the added-value of our distances which point out specific anomalies and give good insights about the amplitude of an anomaly. We used existing implementations of two well known sequence dissimilarities *DTW* ([SFY07]) and *LCS*([BHR00]). These implementations are given by mlpy ([AVM+12]), a Python module for machine learning built. For our experimentations, the events of execution traces were coded as integers, as required by *mlpy*. $LCS(x,y)$ returns the length of the longest common sequence of x and y. We then obtain distance between x and y by $d(x,y) = |x| + |y| - 2 * LCS(x,y)$. In this experimentation we use one reference trace and two suspicious traces. T_r is the reference trace of *gen* video presented in Table 4.3. T_1 is obtained by using the gstreamer element *identity* before the video decoding plugin, with property $sleep\text{-}time = 30000$. T_2 is obtained by using the gstreamer element *identity* before the video decoding plugin, with property $sleep\text{-}time = 5000$. During the decoding, A/V/S desync. anomaly is visible, but the visual degradation obtained while generating T_2 is slighter than those obtained while generating T_1. Table 4.5 gives the name of each trace and the corresponding anomaly.

Table 4.5 – Description of traces to compare

T_r	reference trace
T_1	A/V/S desync. suspicious trace
T_2	slight A/V/S desync. suspicious trace

A/V/S desync. anomaly is detected with occurrence dissimilarity, we then compare DTW and LCS distances to occurrence dissimilarity. Table 4.6 shows the values of dissimilarities obtained w.r.t. to two execution traces T_1 and T_2 given as input and compared to T_r. T_1 is more disturbed than T_2 by the desynchronization anomaly, then we expect that for a distance d that $d(T_r,T_1) > d(T_r,T_2)$. For a better observability, we do not normalize the distance values obtained.

Table 4.6 – Comparison with DTW and LCS distances

	DTW	**LCS**	**Occurrence dissimilarity**
$d(T_r,T_1)$	509069	28035	132090.5
$d(T_r,T_2)$	504472	28086	131525

T_1 is more disturbed than T_2, occurrence dissimilarity and *DTW* dissimilarity reveal this expectation. In contrary, *LCS* gives the opposite statement as result. The added value of occurrence dissimilarity comparing to DTW is the ability to point out a specific anomaly type, in addition to determine the most disturbed trace between two abnormal traces.

Interest of specific distances to capture anomalies - [G3]

T_3 is obtained with property *error-after*. An error occurs during the video streaming after a given number $N(N = 500)$ of buffers. It corresponds to the player crash anomaly. Table 4.7 shows the values of dissimilarities obtained by comparing T_3 to T_r. As before, the distance values are not normalized. $dtw(T_r, T_3)$ and $lcs(T_r, T_3)$ show that T_3 is far from T_r but an

Table 4.7 – Comparison with DTW and LCS distances

	DTW	LCS	Dropping dissimilarity
$d(T_r, T_3)$	920600	18377	48

analyst can not determine the involved anomaly. More than give a distance value between the suspicious trace and the reference trace, TED diagnoses a player crash anomaly.

Remark 4: to the best of our knowledge, there is no standard dissimilarity on sequences which take into account temporality, that is why we do not compare the temporal distance with standard distances.

Running time and Scalability - [G4]

Fig. 4.6 reports the wall clocks of TED for *occurrence* and *dropping* distance, when varying events number of execution traces. Horizontal axis represents the maximum number of events of the two compared traces. In practice, we consider $\theta = 0.25$, as threshold of occ_ratio. One

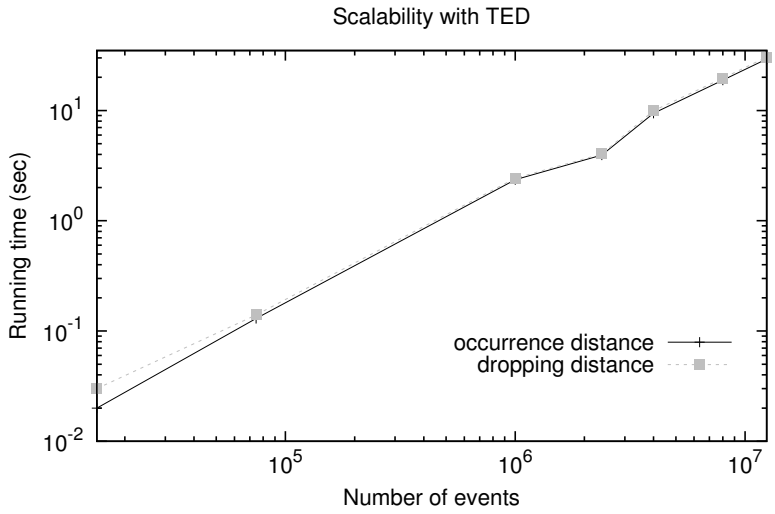


Figure 4.6 – Running time

can notice that, for traces of more than $1G_o$, corresponding to approximatively $4,000,000$ events, TED can give a diagnosis in less than $10s$. For the *pub* video of table 4.6, an output

is obtained in $0.12s$. The experiments showed that the proposed methods can scale to real application traces. This makes TED suitable for analysis of real traces.
However, we have to notice that *temporal distance* takes much more time to be computed, due to algorithm complexity. For instance a suspicious trace from pub video dataset with slow streaming anomaly (the trace contains $74,436$ events) takes up to $3000s$ to be diagnosed as abnormal. It is the same order of magnitude as the standard distance LCS, which is an edit distance. The running time of the *temporal distance* can be improved by considering computation by portions of trace.

4.6 Applying distances on reduced execution traces

In the previous section, we observed that the computing distances, especially the temporal distance could be expensive in term of computation time. This time is strongly dependant of the trace size. Reducing trace size could lead to reduce time computation. A way to apply such reduction is to eliminate common events to both traces (under some constraints to define) in order to keep only events which indicate the presence of an anomaly. This intuition is consistent with the approach developed by Comode et al. in their work [Cor03] on sequence distances. This approach aims to perform a transformation on sequences, in order to produce new sequences, such that the distance between the transformed sequences approximates the distance between the original sequences.

We propose to use blocks obtained in Chapter 3 in order to perform the trace reduction. The approach is explained in paragraph 1 of Section 4.6.1.

The trace reduction should be done by taking into account our proposed distances. The difficulty is to have distances on reduced traces which give a good approximation of distances on original traces. The adaptation is detailed in paragraph 3 of Section 4.6.1.

The approach is evaluated in Section 4.6.2.

Problem statement
The diagnosis problem on reduced traces that we address can be state as follows:
Given d the dissimilarity measure allowing to detect anomaly A, assuming a reference trace T_r and a suspicious trace T_s, given t_r (resp. t_s) the reduced trace obtained from T_r (resp. T_s), determine d' such that:

$$d(T_r, T_s) \neq 0 \Leftrightarrow d'(t_r, t_s) \neq 0$$

This means that the diagnosis should be the same both on original and reduced traces. In the rest of the chapter, *reduced comparison* refers to the comparison applied on reduced traces; *baseline comparison* refers to the comparison applied on original traces.

4.6.1 Adapt dissimilarity measures on reduced execution traces

This section explains our approach to first reduce traces and then apply dissimilarity measures in order to detect the three anomalies: A/V/S desynchronization, player crash and slow streaming.

1-The approach

Our proposal is to use $k\text{-}blocks$ (see Chapter 3) to reduce traces size. We recall that the method described in Section 3.2 of Chap. 3 produces a set S_k of k blocks $\{B_1, B_2, \ldots, B_k\}$ allowing to maximally cover a trace. A $nonBlock$ is a sequence of events out of blocks. Fig. 4.7 shows an example of 3 blocks and 2 nonBlocks in a trace.

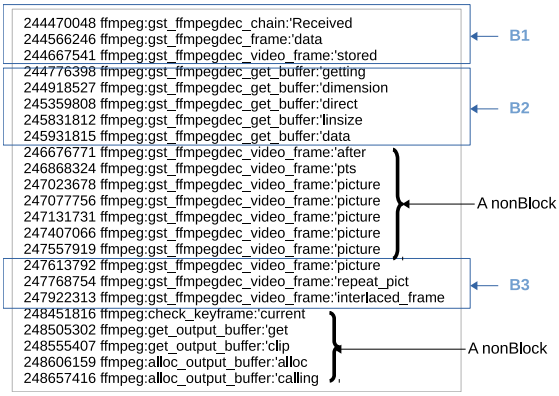


Figure 4.7 – Blocks and nonBlocks in a trace

Since a reference trace is an error-free trace (at least concerning our three anomalies), the reduction function discovers k blocks from reference trace, and use these k blocks to reduce reference and suspicious traces. Using the same blocks allows to ensure some identical information in both traces. Moreover it ensures that we do not remove sequence of events "containing" anomalies, as the blocks are discovered from the reference trace which is error-free. The reduction step should also ensure to keep properties of the traces which guarantee to detect a specific anomaly.

2- The data reduction step

By definition, occurrence and dropping distances are essentially based on events frequency (see Sections 4.3.2 and 4.3.3), while temporal distance (see Section 4.3.4) integrates a constraint of temporality. These two fundamental differences lead to define two distinct processes for the reduction step.

a- Case of occurrence and dropping distances

In this case, we need to take care of events frequency. That is why for each block we remove the same number of occurrences in both traces. The blocks order for removing is given by the sequence of blocks which lead to the best coverage of the reference trace. It is a good heuristic for hoping to remove the maximum number of blocks.

Fig. 4.8 shows an example of data reduction applied on a trace.

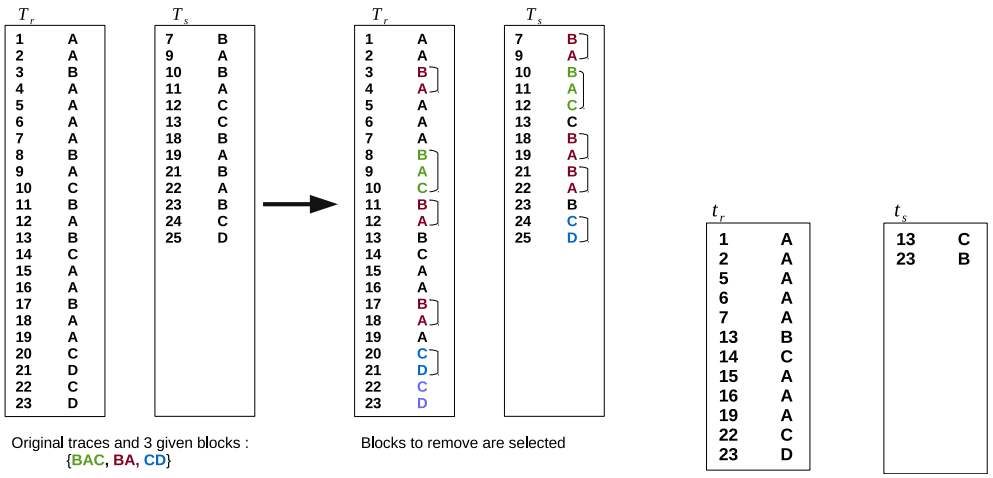


Figure 4.8 – Reducing step - A 3-block set removed from the both traces. occurrences of BAC are first removed, followed by occurrences of BA and occurrences of CD

Algorithm 9 describes the reduction step. This process is done before applying occurrence distance and dropping distance. In lines 4 and 5, the occurrence number of a block B is computed in each trace. The function $removeOccBlock(t_s, B, k)$ aims to remove k occurrences of the block B in the trace t_s.

b- Case of temporal distance

In this case, the data reduction considers the sequentiality (order of events) and the duration of the events. Moreover, an occurrence of a block is effectively removed from the two traces only if the block duration in the two traces is the same. During the reduction step: **1)** the order of events is preserved; **2)** the timestamps are not affected. These points ensure to maintain the properties of the original traces.

In order to illustrate this reduction step, consider the pseudo-code of Algorithm 10. The function $posBlocksCov$ in lines 2-3 uses the ordered set of blocks S_k and a trace. S_k was discovered from T_r and leads to the best coverage of T_r. For the first block B_0 of S_k, the function returns in a list $List$ all the occurrences of B_0 in the trace. $List$ is a list of pair (c_0, B_0)

Algorithm 9 `traceReductionOD`

Input: traces to compare T_r and T_s, an ordered set of k blocks S_k.
Output: t_r and t_s s.t., $|t_r| \leq |T_r|$ and $|t_s| \leq |T_s|$

1: $t_r \leftarrow T_r$
2: $t_s \leftarrow T_s$
3: **for all** B in S_k **do**
4: $occ_1 \leftarrow |occurrences(B, T_r)|$ {where $|occurrences(B, T_r)|$ means the occurrence number of block B in T_r}
5: $occ_2 \leftarrow |occurrences(B, T_s)|$
6: $t_r \leftarrow removeOccBlock(t_r, B, min(occ_1, occ_2))$ {$min(occ_1, occ_2)$ is the minimum value between occ_1 and occ_2}
7: $t_s \leftarrow removeOccBlock(t_s, B, min(occ_1, occ_2))$
8: **end for**
9: **return** t_r, t_s

Algorithm 10 `traceReductionT`

Input: traces to compare T_r and T_s, an ordered set of k blocks S_k.
Output: t_r and t_s s.t., $|t_r| \leq |T_r|$ and $|t_s| \leq |T_s|$

1: $t_r \leftarrow T_r$; $t_s \leftarrow T_s$
2: $List_r \leftarrow posBlocksCov(T_r, S_k)$
3: $List_s \leftarrow posBlocksCov(T_s, S_k)$
4: $c_r \leftarrow timestamp\ of\ first\ block\ in\ List_r$ {c_r indicates the timestamp from which it is possible to remove a block in T_r}
5: **for all** $block\ B_s$ in $List_s$ **do**
6: $p \leftarrow c_r$
7: **while** $(p \leq |T_r|)$ and $(c_r \leq |T_r|)$ **do**
8: $B_r \leftarrow block\ at\ timestamp\ p\ in\ T_r$
9: **if** $(B_r == B_s)$ and $(duration(B_r) == duration(B_s))$ **then**
10: $t_r \leftarrow removeBlock(t_r, B_r,)$
11: $t_s \leftarrow removeBlock(t_s, B_r,)$
12: $c_r \leftarrow p$
13: **end if**
14: $p \leftarrow$ next timestamp in $List_r$
15: **end while**
16: **end for**
17: **return** t_r, t_s

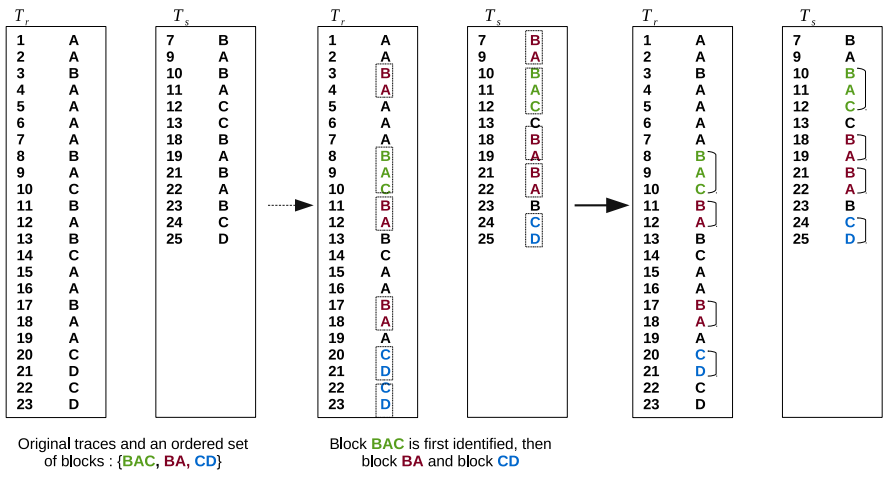
where c_0 is the timestamp of the first event of B_0. The list of occurrences of the second block of S_k, which do not overlap the occurrences of B_0 are added in $List$. The process ends after scanning the whole set S_k and the $List$ is sorted by positions.

Considering example of Fig 4.9, $S_k = \{BAC, BA, CD\}$ and posBlocksCov(T_s, S_k) returns the occurrences list of blocks in S_k. $List_s = \langle(7, BA), (10, BAC), (18, BA), (21, BA), (24, CD)\rangle$ meaning that, for T_s, block BA appears at timestamp 7, following by block BAC at timestamp 10, ..etc.

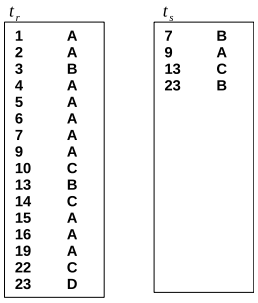
traceReductionT scans all positions of blocks in the suspicious trace (line 5) and for each block in the list, it searches if this block exists in reference trace with the same duration. It preserves constraint **1)** above, meaning that timestamps are not affected during the reduction step. If there is a such block, the block is removed from both traces and a cursor marks the position on T_r from which searching could continue (line 12). The constraint **2)** is ensured, meaning that sequentiality between events is preserved during the reduction step.

Considering example of Fig 4.9(b), the first occurrence of the block BA was not removed from both traces because its duration is 2 in T_r and 3 in T_s.

Fig. 4.9 shows an example of trace reduction for temporal distance.



(a)



(b) reduced traces

Figure 4.9 – Reducing step - The first occurrence of block BA is not removed. In T_r the duration is 2, instead of 3 in T_s

3-Adapted dissimilarity measures

In the following, we assume that both traces to compare are reduced as explained in section 4.6.1. Some important question remain: *could we directly apply our distances on reduced traces without any adaptation?* By construction of dropping and temporal dissimilarities, there is no need to change. In the other hand, the threshold in occurrence distance need to be adapted to reduced traces.

► Dropping dissimilarity

The dropping distance value is proportional to the number of events which appears in one trace and not in the other (See Section 4.3.3). The dropping distance can be directly applied on reduced traces because the blocks removed from both traces contain events occurring in the two traces. The events concerned by the computation of the dissimilarity in original traces are still present in reduced traces t_r and t_s.

The dropping dissimilarity between reduced traces t_r and t_s is then:

$$d'_d(t_r, t_s) = |events(t_r) \triangle events(t_s)| \tag{4.12}$$

► Temporal distance

The reduction function applied in case of temporal distance keep the sequentiality of events and the duration of events. Temporal distance described in Section 4.3.4 deals with order between events and timestamps. Following the reduction process described in Algorithm 10, we deduce without loss of generality that the original temporal distance can be used without transformation on reduced traces and lead to the same diagnosis as on original traces.

The temporal dissimilarity between reduced traces t_r and t_s is then:

$$d'_t(t_r, t_s) = r(n, m) \qquad n = |t_r|, m = |t_s| \tag{4.13}$$

where $r(i, j)$ is computed according to a dynamic programming algorithm.

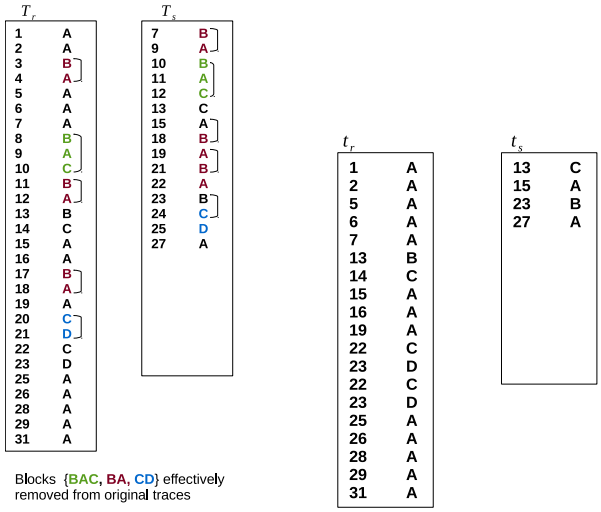
► Occurrence dissimilarity

Before presenting the occurrence dissimilarity for reduced traces, let start by giving in Tab. 4.8 useful notations for the rest of the section.

Table 4.8 – Useful notations

t_r	reduced trace of reference trace T_r
t_s	reduced trace of suspicious trace T_s
$Events(t_r)$	set of distinct events in t_r
$Events(t_s)$	set of distinct events in t_s
RB	set of blocks effectively removed from both original traces
$Events(RB)$	set of distinct events in RB
	for a given event e:
m	maximum number of occurrences of e in two reduced traces t_r and t_s
n	minimum number of occurrences of e in two reduced traces t_r and t_s
α	number of occurrences of e removed from both traces

Example 6. *In Fig. 4.10,* $RB = \{BAC, BA, CD\}$, $events(t_r) = \{A, B, C, D\}$, $events(t_s) = \{A, B, C, D\}$. *By considering the event* A, $m = 13$ *and* $\alpha = 4$.



(a) (b) reduced traces

Figure 4.10

▲

we recall that the occurrence dissimilarity value between two original traces T_r and T_s (see Section 4.3.2) is defined as the number of events that have an occurrence ratio less than or equal to a given threshold θ: $d_o(T_r, T_s) = |\{e \mid occ\_ratio(e, T_r, T_s) \leq \theta\}|$. This measure depends on a ratio computed from occurrence number of individual events. The reduction step has an impact on the value of $occ\_ratio(e, t_r, t_s)$. It is then necessary to consider a suitable threshold value θ' to apply for removed events.

The occurrence dissimilarity between t_r and t_s is:

$$d'_o(t_r, t_s) = |\{e \mid occ\_ratio(e, t_r, t_s) \leq \theta'\}| \tag{4.14}$$

Proposition 2. *The new threshold θ' to consider is*

$$\theta' = \begin{cases} \frac{\theta * m + \alpha(\theta - 1)}{m} & if\ e \in Events(RB) \\ \theta & else \end{cases}$$

Proof. The occurrence ratio of events in RB (see Tab. 4.8 for notations) should be compared using a new value of threshold θ', which takes into account the removed events. The new value of threshold θ' is computed as follows:

Let $N = Min\{nb\_occ(e, T_r), nb\_occ(e, T_s)\}$ and
$M = Max\{nb\_occ(e, T_r), nb\_occ(e, T_s)\}$, for all event e of both traces.
Then $d_o(T_r, T_s) = |\{e \mid \frac{N}{M} \leq \theta\}|$

In baseline comparison, an event e "participates" to the computation of distance if and only if $\frac{N}{M} \leq \theta$ (1).

We want to determine n and m such that $\frac{n}{m} \leq \theta'$ in reduced comparison.
$n = N - \alpha$ and $m = M - \alpha$; $n \geq 0$, $m \geq 0$

$$\begin{aligned} (1) \Rightarrow N &\leq M * \theta \\ \Rightarrow N - \alpha &\leq M * \theta - \alpha \\ \Rightarrow \frac{N - \alpha}{M - \alpha} &\leq \frac{M * \theta - \alpha}{M - \alpha} \\ \Rightarrow \frac{n}{m} &\leq \frac{M * \theta - \alpha}{M - \alpha} \end{aligned}$$

However, $M = m + \alpha$. So,

$$\begin{aligned} \frac{M * \theta - \alpha}{M - \alpha} &= \frac{(m+\alpha) * \theta - \alpha}{(m+\alpha) - \alpha} \\ &= \frac{m * \theta + \alpha * \theta - \alpha)}{m + \alpha - \alpha} \\ &= \frac{m * \theta + \alpha(\theta - 1)}{m} \end{aligned}$$

Then $\frac{n}{m} \leq \frac{\theta * m + \alpha(\theta - 1)}{m}$.

$\theta' = \frac{\theta * m + \alpha(\theta - 1)}{m}$ is a suitable boundary for the new threshold on reduced traces. □

Example 7. *Let a given $\theta = 0.25$. If the occurrence dissimilarity is applied on original traces in Fig. 4.10(a), $D_o(T_r, T_s) = 0$ and T_s is diagnosed as normal trace.*
If we try to apply occurrence distance on reduced traces (Fig. 4.10(b)) with the initial value of threshold θ: $nb\_occ(A, t_r) = 13$ and $nb\_occ(A, t_s) = 2$.
Then $\frac{Min\{nb\_occ(A,t_r), nb\_occ(A,t_s)\}}{Max\{nb\_occ(A,t_r), nb\_occ(A,t_s)\}} = \frac{2}{13} = 0.15 \leq \theta$.

The result of distance is thus $d_o(t_r, t_s) = |\{A\}| = 1$, *which means that there exists an anomaly in this trace. This diagnosis is not correct.*

In our example, $m = 13$, $\alpha = 4$. *The correct value of threshold to apply is* $\theta' = \frac{0.25*13+4(0.25-1)}{13} = 0.02$. *Considering this value of* θ', $\frac{Min\{nb\_occ(A,t_r),nb\_occ(A,t_s)\}}{Max\{nb\_occ(A,t_r),nb\_occ(A,t_s)\}} \geq \theta'$ $(0.15 \geq 0.02)$.
We find that the diagnosis on original traces is the same as the diagnosis on reduced traces: both consider T_s *as a normal trace.*

▲

3- TED: new architecture

By taking into account abstraction, the modified architecture of TED (TracE Diagnosis tool presented in previous chapter) is illustrated in Fig. 4.11.

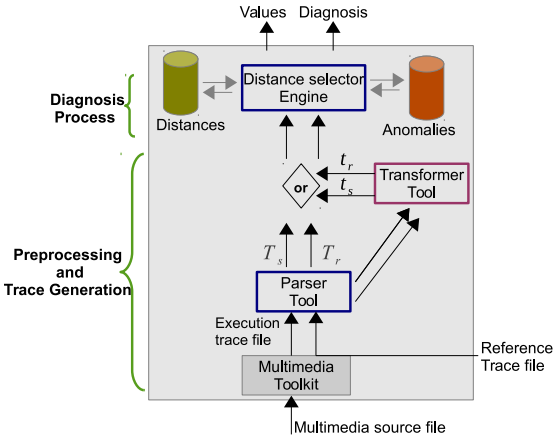


Figure 4.11 – TED Architecture

In the new architecture of TED the *Preprocessing and trace generation* phase integrates a transformation step which allows to use reduced traces or original traces. In the *Diagnosis process*, an appropriate distance is chosen depending on the format of the traces to compare.

4.6.2 Experiments

We have conducted a set of experiments to evaluate how much our proposed reduced comparison improve time computation.

System configuration: Our prototype system is implemented in Python 3.2. The experiments were run on an Intel Xeon E5-2650 at 2.0GHz with 32 Gigabytes of RAM and Linux operating system.

Data Set: We use traces from the application *Gstreamer application*. Reference traces are described in Table 4.9.

Table 4.9 – Experimental dataset: reference traces

	Trace	Video source duration	Nb. of events	Size
Gstreamer	*gen*	$17s$	39,646	$7.6M_o$
	pub	$49s$	74,436	$14.3M_o$
	mov	$3000s$	5,964,485	$1228,8M_o$

*For *gen* trace and *pub* trace, we generated:

▷ : five abnormal traces with A/V/S desync. anomaly (P_1) by using a gstreamer element *identity* ([Gst14]), and different parameters

. ▷ : five abnormal traces with player crash anomaly (P_2) by using a gstreamer element *identity* with property *sleep-time*, and different parameters.

▷ : five abnormal traces with slow streaming anomaly (P_3) by using a stress of CPU and memory in the system, with different parameters.

*For *movie2* trace, we generated:

▷ : five abnormal traces with player crash anomaly.

▷ : one abnormal trace with slow streaming anomaly.

Tab. 4.10 presents all generated traces. For instance, P1-gen1 corresponds to suspicious trace number one of trace *gen*, with anomaly $P1$.

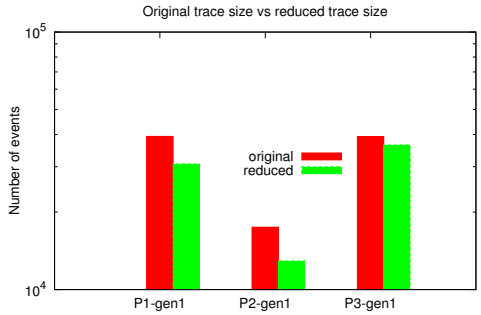
*For the three reference traces, we discovered 10 blocks, using `OneStepOneSon` algorithm (Chapter 3, Section 3.2.2) with a support of 60%.

Table 4.10 – Experimental dataset: names of suspicious traces

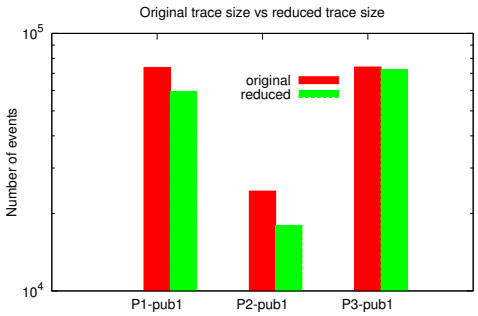
	Anomaly P_1	*Anomaly* P_2	*Anomaly* P_3
Names of suspicious traces of trace *gen*	P1-gen1	P2-gen1	P3-gen1
	P1-gen2	P2-gen2	P3-gen2
	P1-gen3	P2-gen3	P3-gen3
	P1-gen4	P2-gen4	P3-gen4
	P1-gen5	P2-gen5	P3-gen5
Names of suspicious traces of trace *pub*	P1-pub1	P2-pub1	P3-pub1
	P1-pub2	P2-pub2	P3-pub2
	P1-pub3	P2-pub3	P3-pub3
	P1-pub4	P2-pub4	P3-pub4
	P1-pub5	P2-pub5	P3-pub5
Names of suspicious traces of trace *mov*	-	P2-mov1	P3-mov1
	-	P2-mov2	-
	-	P2-mov3	-
	-	P2-mov4	-
	-	P2-mov5	-

Data reduction evaluation

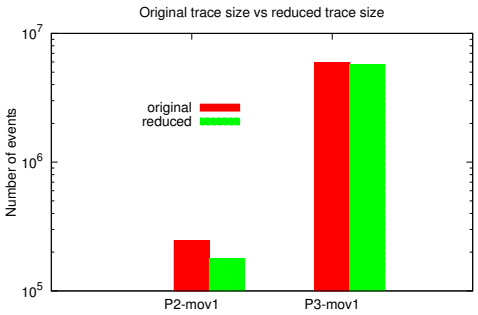
Fig. 4.12 shows the size of traces after data reduction step. In the case of dropping dissimilarity,



(a) A sample of *gen* traces



(b) A sample of *pub* traces



(c) A sample of *mov* traces

Figure 4.12 – Reduced traces size

one can notice that we obtain a reduction of almost the half of the original trace size (see traces P2-gen1, P2-pub1, P2-mov1).For temporal distance, few events are removed. This comes from the duration constraint for removing block. In our experiments, blocks to remove should have the same duration. A relaxation would consist in considering a threshold value (ϵ_T) for the duration. Hence, two blocks would be removed if their duration difference is less than a given threshold ϵ_T. This variant could visibly improve the reduction percentage after data reduction for temporal distance. For instance by considering $\epsilon_T = 0.01ms$, the reduced trace of P3-gen1 has 38,684 events, instead of 39,309 events with the current reduction method. However, this type of relaxation requires also to evaluate the confidence in the diagnosis depending on the value of ϵ_T.

Fig. 4.13 presents the running time needed to reduce two traces (a reference trace and a suspicious trace) given an ordered set of blocks. We can observe that this operation is not time consuming.

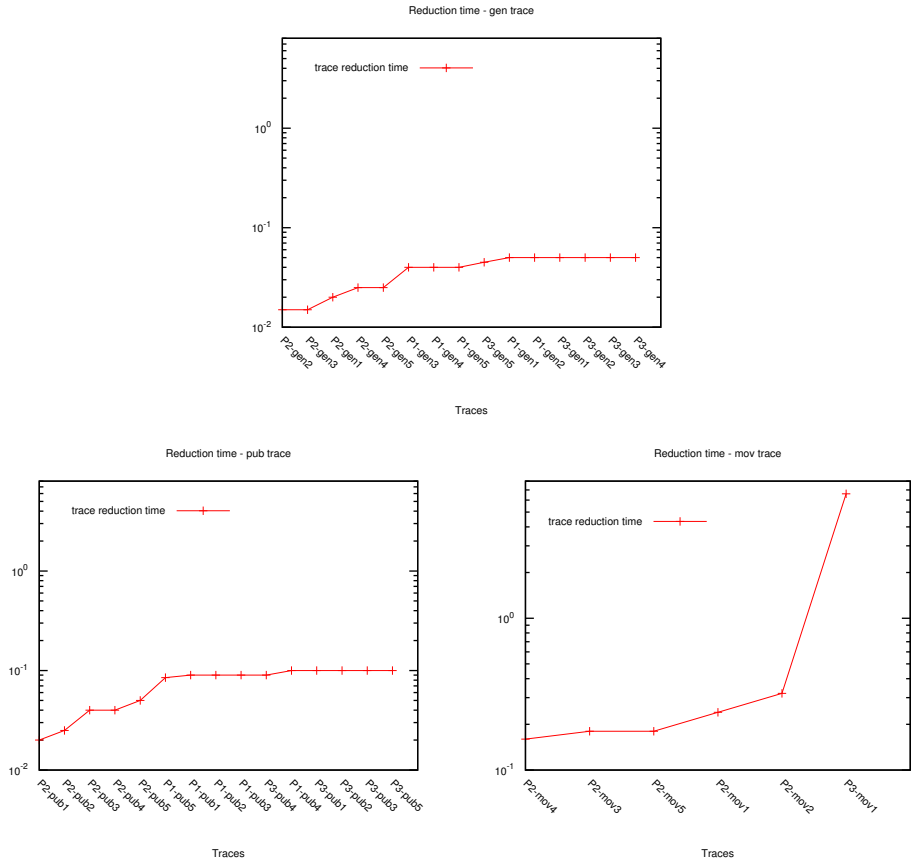


Figure 4.13 – Time (in seconds) to reduce reference and suspicious traces for traces *gen*, *pub*, and *mov*

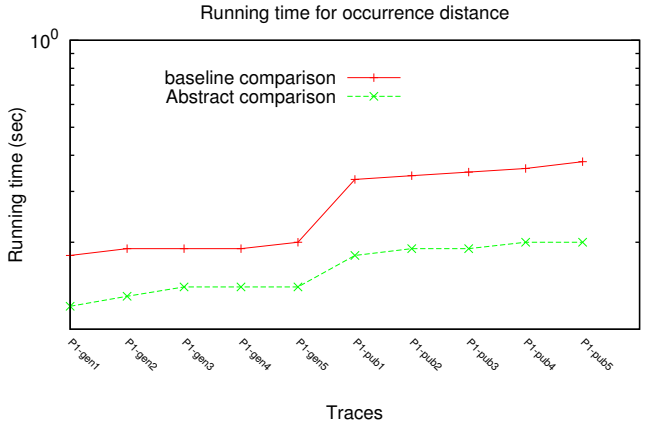
Running time comparison

Fig. 4.14 reports the wall clock time of each distance algorithm for baseline comparison and reduced comparison.

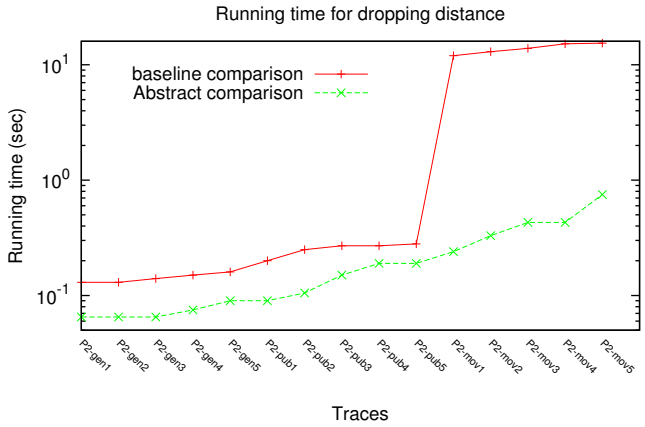
One can observe that reduced comparison improves execution time for less than one order of magnitude, except for *mov* traces in Fig 4.14(c). The reported time include running time for data cleaning, but does not consider time to discover blocks.
Tab. 4.11 presents running time (in seconds) for blocks discovering. This step is only done

Table 4.11 – Time for discovering 10-*blocks* from reference traces

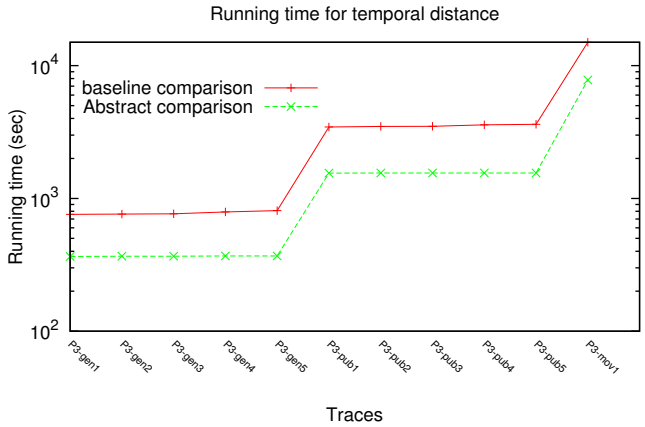
Reference traces	Time (in s)
gen	4
pub	13
mov	1001



(a) Occurrence distance - Running time



(b) Dropping distance - Running time



(c) Temporal distance - Running time

Figure 4.14 – Running time comparison

once for each reference trace. So, the time for doing this becomes insignificant when there is several suspicious traces to compare with one reference trace. It is very often the case in the debugging context. To illustrate this idea, assume that the execution time comparison is almost the same for the next fourth execution on mov trace. After 5 different reduced comparison on mov trace, the running time average of all the executions is $7,706s$. It is less than $15,000s$, which is the running time average for baseline comparison. We also notice that we obtain the same diagnosis by applying baseline comparison or reduced comparison.

4.6.3 Discussion

In the previous sections we presented an approach to reduce execution traces (based on $k\text{-}blocks$) before comparing them. We now discuss about another method for reduced traces comparison, which consists in: first, discovering k blocks sets for each trace (one $k\text{-}block$ set is found per trace), then independently reduce each trace with the corresponding blocks before applying distances. In this case, the usage of a knowledge domain is recommended to compare blocks which could be syntactically different, but semantically similar.

Example 8. *The two subsequences* $\langle i, I : soft, Ix\rangle$ *and* $\langle i, I : usb, Ix\rangle$ *are syntactically different but the two correspond to an* $Interruption$.

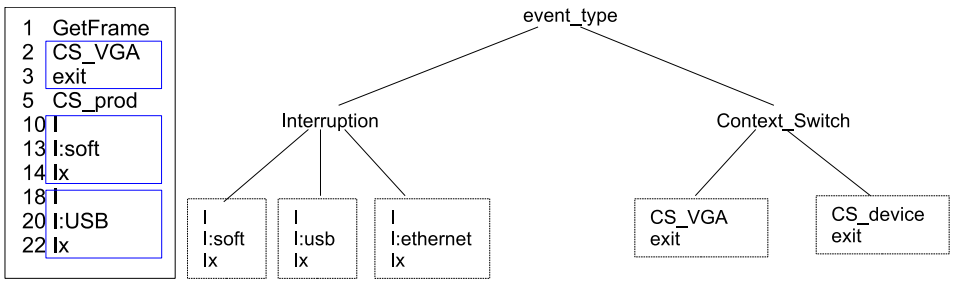
Assuming data there exists a domain knowledge (as a taxonomy or an ontology [GOS09]) about multimedia decoding, this domain is used to annotate each block. Obviously, an ontology development has a substantial cost [STM07] which is not discussed in this work and needs to be take into account for performance tests. Fig. 4.15(b) presents a toy example of an ontology and a trace with blocks. In Fig. 4.15(c) the trace is labelled using the taxonomy; "Interruption" is a label.

There exists many possibilities to explore when comparing two reduced traces in this context. One idea is to consider that two labels are equal if they have the same parent in the taxonomy.

A semantic-based approach for reduced comparison opens a better possibility of abstraction that benefits to the analyst. The analyst can more easily understand the abstracted trace as each block is labelled. However, we have to take care of the fact that, using a taxonomy for trace reduction brings a level of semantic. It is then less intuitive to adapt distances on semantic-based reduced traces and obtain the same diagnosis as on original traces, because a semantic matching level was added. The matching is not exact because it is based on a taxonomy.

4.7 Conclusion

To analyse execution traces and fix bugs, programmers use several tools such as trace visualizers ([CdKSB00, Rob05, MWM06, Sey08]) and techniques such as tracepoints on the execution traces. These techniques need to have an expert to interpret the graphical representation. In



(a) A trace T

(b) A taxonomy of multimedia decoding domain

1 NotMatched
2 Context_Switch
5 NotMatched
10 Interruption
18 Interruption

(c) Trace T rewritten using the taxonomy

Figure 4.15 – An example of knowledge domain usage to label execution traces

contrast, our work based on dissimilarities develops a technique which limits the developer intervention.

Our approach diagnoses anomalies in an execution trace of multimedia application, by comparison with a reference trace. We use dissimilarities as models of comparison and specifically design three distinct dissimilarities in order to tackle well-known anomalies of the multimedia domain. We experimentally show the added-value of our solution compared to existing sequence dissimilarities and show that our proposed approach scales well to real application traces. Our proposed dissimilarities allow to identify a specific problem and thus give an added-value to the analysis. Moreover, as all dissimilarities, they also provide insights of how far an abnormal trace is from a correct one. We lastly present a use case on how TED performs an analysis trace and conduct some experiments to evaluate TED scalability and accuracy.

We proposed a reduction, using $k\text{-}blocks$, which allows to guarantee that a same diagnosis is found for original and reduced traces. This reduced comparison does not bring a semantic level which could better improve the trace analysis. A more challenging goal is to allow an external source, a knowledge domain (as ontology) to semantically approximate the distance. The main difficulty is to design this ontology of the domain of multimedia for embedded systems. Strategies to use this ontology are varied. The ontology designing is a part of the ongoing thesis of Leon Fopa [3].

Our approach can be seen as a *hypothetico-deductive model* [Gri90]: each type of anomaly is considered as a hypothesis that is tested on a observable trace using a "predicting" model encapsulated into a specific measure of dissimilarity.

[3] PhD student at Laboratoire d'Informatique de Grenoble

5 Conclusion

Contents

Embedded systems are everywhere in our daily life. The development of multimedia applications is a competitive field where many works showed the interest of using execution traces for analysis. Execution traces are event sequences which generally have large size. Therefore, our contributions focus on useful trace processing techniques for debugging embedded multimedia applications.

5.1 Contributions summary

We review in Chapter 2 recent works on abstractions techniques and event sequences comparison. We highlight some aspects which are not all taking into account in the related approaches, especially sequentiality, temporality and added-value in comparison.

Based on a mix of sequence mining and greedy algorithms, we propose in Chapter 3 an approach to improve trace exploration by abstracting execution trace. This approach discovers a set of representative blocks. Experiments performed show that this method is scalable. This allows to tackle real world execution traces. We also demonstrate by a practical analysis, how helpful representatives blocks can be for application developers. Obviously, this approach can be applied on execution traces, or events sequences from any domain.

Chapter 4 presents our second contribution which is an efficient method implemented in *TED* (TracE Diagnosis tool), in order to detect anomalies in multimedia applications. Instead of designing a complex measure hoping that it will find all anomalies, we choose to deconstruct this process by designing appropriate dissimilarities to compare suspicious traces with a reference trace. We have conducted experiments which show that our method efficiently allows to detect if a trace is abnormal, using comparison. Moreover, the method brings an added-value by giving a diagnosis.

We also propose a first step toward an application of dissimilarities on reduced traces. Trace reduction is done using *k-blocks* discovered in Chapter 3. The execution time improvement is less than one order of magnitude and results are promising. We discuss another idea that can be retained for improvement in the perspectives below.

Fig. 5.1 gives an overview of our contributions and shows how they are linked. We notice that trace abstraction can be use for exploring trace but also for detecting anomalies.

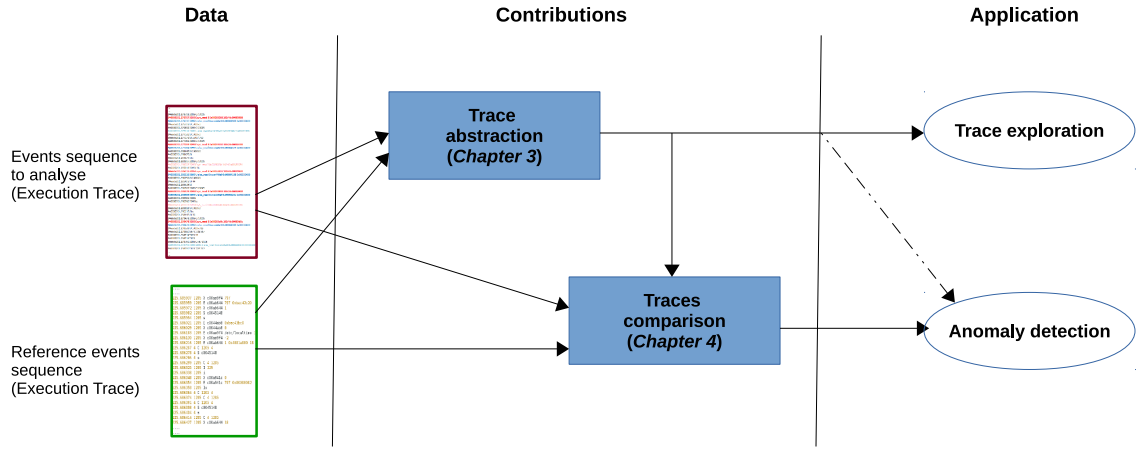


Figure 5.1 – Summary of the contributions

5.2 Perspectives

In this thesis, we present two main contributions for execution traces analysis. Below, we explain several research possibilities identified during this work:

Trace abstraction

- *Coverage optimization:* in Chapter 3 we present several efficient algorithms to discover blocks. We propose some greedy approaches for scalability, which avoid to explore all the search space of candidate blocks. Nevertheless in some applications different to debugging tasks, value of coverage can be more important than execution time. One possible alternative of our method is to compute the maximum coverage of a *k-blocks set* in the trace by simply solving binary integer linear programming [Bal65] of the maximum coverage on individual frames. The blocks candidates can be ordered such that the *k-blocks* that brings the optimal coverage will be find far earlier than exhausting all candidates. This method will increase running time, but will ensure that the obtained coverage is the best.
- *Labelling of k-blocks:* for now blocks are simply sequences of events, and the developer has to find out himself what is the block about. Therefore, integrating some domain knowledge, discovered blocks could be labelled by an automatic or semi-automatic

method.

- *Analysis of parallel traces:* the sequencing of events is only important for events having some temporal dependency. We would like to detect such dependencies, in order to restrict the covering conditions on blocks to only such time-dependent events.
- *Exploration of other types of traces:* our method can be applied on any sequence of events. In this thesis, we have focused on multimedia application traces. It could be very interesting to observe discovered blocks when applying our approach on traces of different domains as:
  - education; $k\text{-}blocks$ can be representative of the sequence of actions performed by a student who failed the examination.
  - health; $k\text{-}blocks$ can represent specific behaviour of people who have the same pathology.

Traces comparison

- *Semantic comparison:* in Section 4.6 of Chapter 4, we show that it is possible to adapt distances on reduced traces. Nevertheless, in some cases, the gain in term of execution time is not really significant. We can consider to use some domain knowledge, as discussed in Section 4.6.3 to define semantic distances. We believe that a semantic comparison, based on ontologies for instance, is necessary to highlight another type of problems.
- *Enhanced trace diagnosis tool:* the second direction is to enlarge TED to other types of anomalies for instance the image is completely fuzzy, upside down and/or cut in half. The strength of our contribution is that it is easily extensible to other types of anomalies. For each new anomaly, we only need to follow the same methodology as explained in Chapter 4 to find the best suitable distance able to clearly detect the anomaly. There is no need to do any changes in TED existing architecture. Finally, additional constraints can be introduced such as parallel execution traces and the challenge is to identify, for example, streams of different execution and take them into account for the computation of distances.

A French Summary

Contents

A.1 Introduction

Dans cette thèse, nous proposons des solutions pour l'analyse d'applications multimédias pour systèmes embarqués. La prolifération de systèmes embarqués, box, tablettes, smartphones, fournit un accès permanent aux contenus multimedias. Le développement d'applications multimédias est un domaine hautement compétitif dans lequel chaque seconde perdu par un développeur pour debugger une application en coûte financièrement à l'entreprise. La survie des entreprises dépend de la capacité des développeurs à rapidement développer, debugger, optimiser les logiciels et s'adapter à la constante évolution.

Les systèmes embarqués peuvent être définis comme des systèmes de traitement de l'information embarqués dans des équipements comme des voitures ou des équipements de télécommunication. Selon des statistiques fournies par plusieurs sites web, les ventes de tablettes ont été multipliées par six entre 2010 et 2012, d'où l'intérêt des entreprises à se lancer dans la course au développement d'applications pour systèmes embarqués.

Les applications multimédias sont parmi les plus utilisées sur les systèmes embarqués. Les applications multimédias effectuent une série de transformations (appelé décodage multimedia) sur un flux de données. Les frameworks multimedia tels que Gstreamer [Gst14] or VLC [Vid14] offrent offrent un grand choix de composants de calcul qui peuvent être combinés dans un pipeline. La structure et la taille de ce pipeline dépendent du type d'application multimedia.

Les tâches d'identification des comportements inattendus ou indésirables peuvent être fastidieuses car même un code syntaxiquement correct conduit souvent à des problèmes de mémoire ou de tâches en arrière plan qui peuvent condure à un comportement incorrect. L'analyse d'applications multimedias peut révéler d'importantes informations pour améliorer la compréhension de l'exécution du programme. Plusieurs traveux précédents The analysis of multimedia application traces can reveal important information to enhance program execution comprehension. [PR11],[Pou14],[Cue13] ont montré que les techniques de traçage sont celles à utiliser dans l'environnement des systèmes embarqués. Les techniques de traçage impliquent de détecter et de stocker des événements pendant l'execution du programme afin d'en faire une analyse off-line. Cependant, les traces obtenues peuvent être très volumineuses, ce qui empêche leur exploitation effective par les développeur.

Les traces sont des séquences d'évenements horodatés produits par une application ou un système. Elle peuvent être obtenues par des techniques logicielles (qui consistent à instrumenter le code et insérer des instructions d'écriture afin d'obtenir un log de l'éxécution), ou matérielles (qui consistent à avoir des modules matériels dédiés où les composants de l'architecture peuvent écrire leurs traces.

Dû au grand volume d'information disponible, il est très difficile d'analyser manuellement

les traces d'exécution. Il est alors essentiel de mettre au point des techniques d'analyse qui prennent en compte la masse de données. Notre approche est donc de réduire la taille de la trace afin de permettre une meilleure interpretation par le développeur. Cette réduction doit évidemment éviter toute perte d'information, et doit guider le développeur dans son analyse en présentant une trace plus accessible en terme de nombre d'événements à explorer.

Contributions de la thèse

Nous proposons dans ce travail deux techniques d'analyse de traces d'exécution : la première vise à abstraire la trace afin d'en réduire la taille et permettre une meilleure exploitation. L'idée sous-jacente est de grouper les sequences d'événements et remplacer les groupes par des blocs significatifs. Ainsi, une trace initialement vue comme une séquence d'événements devient une séquence de blocs et est considérablement réduite. La seconde méthode consiste à détecter des erreurs dans la trace (où la trace est une séquence d'évenements ou de blocs). La détection d'erreurs est réalisée en comparant la trace suspecte à une trace de référence. Cette comparaison a pour objectif d'extraire des anomalies, c'est-à-dire des motifs ou des correlations contenues dans la trace suspecte mais pas dans la trace de référence. Nos contributions peuvent être résumées comme suit :

- **Abstraction de traces.** Nous réalisons une abstration de la trace à l'aide de séquences d'événements appelés *blocs*. Ces blos sont automatiquement extraits de la trace à grâce à des techniques de fouille de données. Les techniques classiques produisent un certain nombre de résultats, pas toujours maitrisé. Nous conservons uniquement les blocs candidats les plus prometteurs, c'est-à-dire les blocs qui assurent la meilleure couverture de la trace d'origine.Nous proposons également une méthode originale qui combine en une seule étagpe les phases de découverte des blocs et de réécriture de la trace.

- **Détection d'anomalies par comparaisons de traces.** Nous proposons de fournir automatiquement un diagnostic sur les traces en comparant deux traces d'éxécution. L'une d'elles est une trace de référence correspondant à un comportement correct, et la seconde trace est la trace à analyser. Nous identifions premièrement une famille d'anomalies ayant tendance à apparaitre dans les applications multimédias. Nous choisissons les types d'anomalies les plus récurrentes et concevons un score spécifique de dissimilarité pour chacune d'elles. Ces scores aident le développeur à mesurer à quel point la trace à analyser s'éloigne du comportement normal. Nous proposons ensuite une version de comparaison applicable sur traces réduites.

A.2 Une méthode pour abstraire les séquences d'événements

Bien vouloir se referer au Chapitre 3 pour plus de détails.

A.3 Une méthode de comparaison basé sur la dissimilarité pour analyser les séquences d'événements

Bien vouloir se referer au Chapitre 4 pour plus de détails.

A.4 Conclusion et perspectives

Dans les paragraphes précédents, nous avons mis en lumière des aspects qui n'étaient pas pris en compte dans les approches de l'état de l'art relatives aux techniques d'abstraction et de comparaison de séquences.
Grâce à une combinaison de techniques de fouille de séquences et d'algorithmes gloutons, nous proposons dans notre première contribution une approche afin d'améliorer l'exploration de la trace. Cette approche découvre un ensemble de blocs représentatifs. Les expérimentations réalisées montrent que la méthode passe à l'échelle et s'applique sur des traces d'éxécution réelles. À l'aide d'un cas d'analyse pratique, nous démmontrons comment les blocs représentatifs sont utiles pour le développeur. Cette approche est applicable autant sur les traces d'éxécution que sur les séquences d'événements provenant de différents domaines.

Notre seconde contribution a été implémentée dans *TED* (TracE Diagnosis tool), un outil qui permet de détecter des anomalies dans les applications multimédias. Au lieu de concevoir une mesure complexe en espérant qu'elle trouvera toutes les anomalies, nous avons choisi de déconstruire ce processus en concevant des mesures de dissimilarité appropriées pour comparer les traces suspectes à une trace de référence.

Nous avons réalisé des expérimentations qui montrent que notre méthode permets de détecter à l'aide de la comparaison, si une trace est anormale. Mieux encore, la méthode apporte une plus-value en fournissant un diagnostic. Nous avons également proposé une première étape orientée vers l'application des distances sur traces réduites. Cette réduction de l atrace est réalisée grâce aux $k\text{-}blocks$ découvert à l'aide de notre méthode d'abstraction. L'amélioration en terme de temps d'exécution est de moins d'un ordre de grandeur avec cependant des résultats prometteurs. Dans les perspectives, nous présentons une autre idée qui peut être utilisisée pour une amélioration.

Perspectives
Les différentes possibilités de recherche identifiées pendant ce travail sont les suivantes :

Concernant *l'abstraction* : nous avons présenté plusieurs algorithmes de découverte de blocs basés sur une approche gloutonne. Cependant dans certaines tâches de debugging, la valeur de couverture est plus importante que le temps mis pour l'obtenir. Une alternative à notre méthode est d'utiliser la programmation linéaire afin d'obtenir l'ensemble des $k\text{-}blocks$ [Bal65]. Ces blocs sont manuellement labelisés pour l'instant, il serait pertinent d'intégrer une base de connaissances afin de réaliser un labeling automatique ou semi automatique des blocs. Pour finir, notre méthode est applicable sur toute s'quence d'événements. Nous

avons choisi dans cette thèse de l'appliquer aux traces d'exécution mais il serait intéressant de l'appliquer sur d'autres types de traces venant du domaine de la santé ou de l'éducation par exemple.

Concernant *la comparaison* : nous avons montré la possibilité d'adapter les distances sur traces réduites, nous pouvons opter pour l'utilisation d'une base de connaissances (comme une ontologie) afin de définir une distance sémantique, afin de mettre en lumière d'autres types de problèmes. De plus, la force de notre contribution est qu'elle est facilement extensible à d'autres types d'anomalies, sans changement de l'architecture proposée.

Bibliography

[Agg13] Charu C Aggarwal. *Outlier analysis*. Springer, 2013.

[AVM+12] Davide Albanese, Roberto Visintainer, Stefano Merler, Samantha Riccadonna, Giuseppe Jurman, and Cesare Furlanello. mlpy: Machine learning python. *arXiv preprint arXiv:1202.6548*, 2012.

[Bal65] Egon Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–546, 1965.

[BHR00] Lasse Bergroth, Harri Hakonen, and Timo Raita. A survey of longest common subsequence algorithms. In *Proceedings Seventh International Symposium on String Processing and Information Retrieval, 2000. SPIRE 2000.*, pages 39–48. IEEE, 2000.

[CBK12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012.

[CCT10] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C Tappert. A survey of binary similarity and distance measures. *Journal of Systemics, Cybernetics and Informatics*, 8(1):43–48, 2010.

[CdKSB00] J Chassin de Kergommeaux, B Stein, and Pierre-Eric Bernard. Pajé, an interactive visualization tool for tuning multi-threaded parallel applications. *Parallel Computing*, 26(10):1253–1274, 2000.

[Cor03] Graham Cormode. *Sequence distance embeddings*. PhD thesis, Department of Computer Science, University of Warwick, 2003.

[cp13] cccp project. Discussion page: Troubleshooting guide. http://www.cccp-project.net/wiki/index.php?title=Troubleshooting_Guide, Jan 2013. Accessed: 2013/01/04.

[CRF03] William Cohen, Pradeep Ravikumar, and Stephen Fienberg. A comparison of string metrics for matching names and records. In *KDD Workshop on Data Cleaning and Object Consolidation*, volume 3, pages 73–78, 2003.

[Cue13] Patricia Lopez Cueva. *Debugging Embedded Multimedia Application Execution Traces through Periodic Pattern Mining*. PhD thesis, Université de Grenoble, 2013.

[CZvD11] Bas Cornelissen, Andy Zaidman, and Arie van Deursen. A controlled experiment for program comprehension through trace visualization. *Software Engineering, IEEE Transactions on*, 37(3):341–355, 2011.

[DZPM09] Simon Dobrisek, Janez Zibert, Nikola Pavesic, and France Mihelic. An edit-distance model for the approximate matching of timed strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):736–741, 2009.

[Ege08] Jens Egeblad. *Heuristics for Multidimensional Packing Problems*. PhD thesis, University of Copenhagen, Department of Computer Science, 2008.

[Ff13] Freebox-forum. Forum: freebox discussion. www.freebox-forum.net/forum/la-television-freebox/lecture-video-hd-1080p-mkv-saccadee-t7159963.html, 2013. Accessed: 2013/03/12.

[Gf13] Gene-forum. Forum: freebox revolution discussion. http://www.generation-nt.com/freebox-revolution-nouveaux-problemes-avec-lecture-videos-actualite-1711302.html, 2013. Accessed: 2013/03/12.

[GGAH14] Manish Gupta, Jing Gao, Charu Aggarwal, and Jiawei Han. Outlier detection for temporal data. *Synthesis Lectures on Data Mining and Knowledge Discovery*, 5(1):1–129, 2014.

[GM04] Hubert Garavel and Radu Mateescu. Seq. open: a tool for efficient trace-based verification. In *Model Checking Software*, pages 151–157. Springer, 2004.

[GOS09] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

[Gri90] Thomas R Grimes. Truth, content, and the hypothetico-deductive method. *Philosophy of Science*, pages 514–522, 1990.

[Gst14] Gstreamer. Gstreamer website. http://www.gstreamer.net, 2014. Accessed: 2014-02-26.

[gue10] Xavier guerin. *Approche Efficace de Developpement de Logiciel Embarque pour des Systemes Multiprocesseurs sur Puce*. PhD thesis, 2010.

[ha14] helpx adobe. Troubleshoot video files. http://helpx.adobe.com/premiere-elements/kb/troubleshoot-video-premiere-elements.html, 2014. Accessed 2014/06/04.

[HEJ09] Kevin J. Hoffman, Patrick Eugster, and Suresh Jagannathan. Semantics-aware trace analysis. In *ACM SIGPLAN Notices*, volume 44, page 453, May 2009.

[Hf13] Hardware-forum. Forum: Hardware. http://forum.hardware.fr/hfr/VideoSon/Traitement-Video/probleme-codec-video-sujet_103890_1.htm, 2013. Accessed: 2013/03/12.

[HFS98] Steven A Hofmeyr, Stephanie Forrest, and Anil Somayaji. Intrusion detection using sequences of system calls. *Journal of computer security*, 6(3):151–180, 1998.

[HKP12] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan kaufmann, 2012.

[IRBT14] Sylvain Iloga, O. Romain, L. Bendaouia, and Maurice Tchuente. Musical genres classification using markov models. In *In proceedings of ICALIP*, pages 973–977, 2014.

[KIA+11] Hyungsul Kim, Sungjin Im, Tarek Abdelzaher, Jiawei Han, David Sheridan, and Shobha Vasudevan. Signature Pattern Covering via Local Greedy Algorithm and Pattern Shrink. *2011 IEEE 11th International Conference on Data Mining*, pages 330–339, 2011.

[KT09] Jerry Kiernan and Evimaria Terzi. Constructing comprehensive summaries of large event sequences. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(4):21, 2009.

[KWK10] Johan Kraft, Anders Wall, and Holger Kienle. Trace recording for embedded systems: Lessons learned from five industrial projects. In *Proceedings of the First International Conference on Runtime Verification (RV 2010)*. Springer-Verlag (Lecture Notes in Computer Science), November 2010.

[ld14] Online language dictionaries. Dictionary reference. http://dictionary.reference.com/browse/abstraction, 2014. Accessed: 17/06/2014.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. Technical Report 8, 1966.

[LMFC14] Hoang Thanh Lam, Fabian Mörchen, Dmitriy Fradkin, and Toon Calders. Mining compressing sequential patterns. *Statistical Analysis and Data Mining*, 7(1):34–52, 2014.

[LQ12] Mengchi Liu and Junfeng Qu. Mining high utility itemsets without candidate generation. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, CIKM '12, pages 55–64, New York, NY, USA, 2012. ACM.

[LSC97] Wenke Lee, Salvatore J Stolfo, and Philip K Chan. Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop on AI Approaches to Fraud Detection and Risk Management*, pages 50–56, 1997.

[LWF12] Junqiang Liu, Ke Wang, and Benjamin Fung. Direct discovery of high utility itemsets without candidate generation. In *IEEE 12th International Conference on Data Mining (ICDM), 2012*, pages 984–989. IEEE, 2012.

[LYC08] Yu-Chiang Li, Jieh-Shan Yeh, and Chin-Chen Chang. Isolated items discarding strategy for discovering high utility itemsets. *Data & Knowledge Engineering*, 64(1):198–217, 2008.

[LZW12] Guimei Liu, Haojun Zhang, and Limsoon Wong. Finding minimum representative pattern sets. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 51–59. ACM, 2012.

[Mör06] Fabian Mörchen. *Time series knowledge mining*. Citeseer, 2006.

[MR97] H. Mannila and P. Ronkainen. Similarity of event sequences. In *Proceedings of the 4th International Workshop on Temporal Representation and Reasoning (TIME '97)*, TIME '97, pages 136–, Washington, DC, USA, 1997. IEEE Computer Society.

[MRS08] Sabrina Mantaci, Antonio Restivo, and Marinella Sciortino. Distance measures for biological sequences: Some recent approaches. *International Journal of Approximate Reasoning*, 47(1):109–124, 2008.

[ms13] microsoft support. Support: Microsoft. http://support.microsoft.com/kb/265523, 2013. Accessed: 2013/03/12.

[MTV97] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.

[MWM06] Mike McGavin, Tim Wright, and Stuart Marshall. Visualisations of execution traces (vet): an interactive plugin-based visualisation tool. In *Proceedings of the 7th Australasian User interface conference - Volume 50*, AUIC '06, pages 153–160, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[Nf13] Numeric-forum. Forum numericable. http://entraide.numericable.fr/threads/2124-Lecture-fichiers-audio-et-vid%C3%A9os-via-media-center, 2013. Accessed: 2013/03/12.

[PHL11] Heidar Pirzadeh and Abdelwahab Hamou-Lhadj. A Novel Approach Based on Gestalt Psychology for Abstracting the Content of Large Execution Traces for Program Comprehension. In *2011 16th IEEE International Conference on Engineering of Complex Computer Systems*, pages 221–230. IEEE, 2011.

[PNSK+06] Tan Pang-Ning, Michael Steinbach, Vipin Kumar, et al. Introduction to data mining. In *Library of Congress*, 2006.

[Pou14] Kevin Pouget. *Programming-Model Centric Debugging for Multicore Embedded Systems*. PhD thesis, Université de Grenoble, 2014.

[PPC+01] Jian Pei, Helen Pinto, Qiming Chen, Jiawei Han, Behzad Mortazavi-Asl, Umeshwar Dayal, and Mei-Chun Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 0215–0215. IEEE Computer Society, 2001.

[PR11] Carlos Hernan Prada Rojas. *Une approche à base de composants logiciels pour l'observation de systèmes embarqués*. These, Université de Grenoble, 2011.

[PSHLM11] Heidar Pirzadeh, Sara Shanian, Abdelwahab Hamou-Lhadj, and Ali Mehrabian. The Concept of Stratified Sampling of Execution Traces. In *2011 IEEE 19th International Conference on Program Comprehension*, pages 225–226. IEEE, 2011.

[Rob05] James Roberts. Tracevis: an execution trace visualization tool. In *In Proc. MoBS 2005*. Citeseer, 2005.

[Ron98] Pirjo Ronkainen. Attribute similarity and event sequence similarity in data mining. *Ph. lic. thesis, University of Helsinki, 1998. Available as Report C-1998-42, University of Helsinki, Department of Computer Science, October 1998*, 1998.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. *Mining sequential patterns: Generalizations and performance improvements*. Springer, 1996.

[SCA06] Pei Sun, Sanjay Chawla, and Bavani Arunasalam. Mining for outliers in sequential databases. In *SDM*, pages 94–105. SIAM, 2006.

[Sey08] Justin Seyster. Techniques for visualizing software execution. Technical report, Citeseer, 2008.

[SFY07] Yasushi Sakurai, Christos Faloutsos, and Masashi Yamamuro. Stream monitoring under the time warping distance. In *IEEE 23rd International Conference on Data Engineering, 2007. ICDE 2007.*, pages 1046–1055. IEEE, 2007.

[ST11] SoC-Trace. Fui project soc-trace. http://www.minalogic.com/TPL_CODE/TPL_PROJET/PAR_TPL_IDENTIFIANT/2717/15-annuaire-innovations-technologiques-nanotechnologie-systeme-embarque.htm#.U0wUsaRdlEg, 2011. Accessed: 20/03/2014.

[Ste03] B De Oliveira Stein. Pajé trace file format. 2003.

[STM07] Elena Paslaru Bontas Simperl, Christoph Tempich, and Malgorzata Mochol. Cost estimation for ontology development: applying the ontocom model. In *Technologies for Business Information Systems*, pages 327–339. Springer, 2007.

[SV12] Koen Smets and Jilles Vreeken. Slim: Directly mining descriptive patterns. In *SDM*, pages 236–247. SIAM, 2012.

[TAG07] Romain Tavenard, Laurent Amsaleg, and Guillaume Gravier. Estimation de similarité entre séquences de descripteurs à l'aide de machines à vecteurs supports. In *BDA*, 2007.

[TV12] Nikolaj Tatti and Jilles Vreeken. The long and the short of it: Summarising event sequences with serial episodes. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 462–470. ACM, 2012.

[TWSY10] Vincent S. Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S. Yu. Up-growth: an efficient algorithm for high utility itemset mining. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 253–262, New York, NY, USA, 2010. ACM.

[VA03] Susana Vinga and Jonas Almeida. Alignment-free sequence comparison—a review. *Bioinformatics*, 19(4):513–523, 2003.

[VGW02] Frank Vahid, Tony Givargis, and John Wiley. *Embedded system design: a unified hardware/software introduction*, volume 4. John Wiley & Sons New York, NY, 2002.

[Vid14] VideoLAN. Vlc website. http://www.videolan.org/vlc/, 2014. Accessed: 20/03/2014.

[VN12] V Vijayakumar and R Nedunchezhian. A study on video data mining. *International journal of multimedia information retrieval*, 1(3):153–172, 2012.

[VVLS11] Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[wA13] wiki A/V. Wikipedia: A/v sync. http://en.wikipedia.org/wiki/Audio_to_video_synchronization, 2013. Accessed: 2013/03/12.

[Wf13a] Windows-forum. Faq: Play an audio or video file. http://windows.microsoft.com/en-us/windows7/play-an-audio-or-video-file-frequently-asked-questions, 2013. Accessed: 2013/01/04.

[Wf13b] Woobees-forum. Forum woobees. http://forum.bouyguestelecom.fr/questions/467605-lecture-videos-bloque-via-media-center-bbox-sensation-freeze, 2013. Accessed: 2013-03-12.

[WFP99] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy, 1999. Proceedings of the 1999*, pages 133–145. IEEE, 1999.

[WH] Zellescher Weg and Robert Henschel. Introducing OTF / Vampir / VampirTrace. *Memory*.

[wik14] Wikipedia knapsack problem. http://en.wikipedia.org/wiki/Knapsack_problem, 2014. Accessed: 25/04/2014.

[WLYT13] Cheng-Wei Wu, Yu-Feng Lin, Philip S Yu, and Vincent S Tseng. Mining high utility episodes in complex event sequences. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 536–544. ACM, 2013.

[WMMM90] S. Wu, U. Manber, G. Myers, and W. Miller. An o(np) sequence comparison algorithm. *Inf. Process. Lett.*, 35(6):317–323, September 1990.

[WS09] Krist Wongsuphasawat and Ben Shneiderman. Finding comparable temporal categorical records: A similarity measure with an interactive visualization. In *IEEE Symposium on Visual Analytics Science and Technology, 2009. VAST 2009.*, pages 27–34. IEEE, 2009.

[WSTY12] Cheng Wei Wu, Bai-En Shie, Vincent S. Tseng, and Philip S. Yu. Mining top-k high utility itemsets. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 78–86, New York, NY, USA, 2012. ACM.

[YL00] Nong Ye and X Li. A markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, volume 166, page 169, 2000.

[YZC12] Junfu Yin, Zhigang Zheng, and Longbing Cao. Uspan: an efficient algorithm for mining high utility sequential patterns. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 660–668, New York, NY, USA, 2012. ACM.

[YZC+13] Junfu Yin, Zhigang Zheng, Longbing Cao, Yin Song, and Wei Wei. Efficiently mining top-k high utility sequential patterns. In *IEEE 13th International Conference on Data Mining (ICDM), 2013*, pages 1259–1264. IEEE, 2013.

[Zdn14] Zdnet. zdnet website. http://www.zdnet.fr/actualites/chiffres-cles-le-marche-des-tablettes-39789571.htm, 2014. Accessed: 17/03/2014.

[ZHT06] Zhang Zhang, Kaiqi Huang, and Tieniu Tan. Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes. In *18th International Conference on Pattern Recognition, 2006. ICPR 2006.*, volume 3, pages 1135–1138. IEEE, 2006.

[ZXHW10] Jia Zou, Jing Xiao, Rui Hou, and Yanqi Wang. Frequent Instruction Sequential Pattern Mining in Hardware Sample Data. *2010 IEEE International Conference on Data Mining*, pages 1205–1210, 2010.