Universiteit Leiden
The Netherlands

Department of Computer Science

Finding Anomalies in Sequential Data using Local Outlier Factor

Lars Suanet

Supervisors:
1st Supervisor: Dr. W.J. Kowalczyk
2nd Supervisor: Dr. K. Rietveld

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

Version 1.7
02/09/2017

Abstract

Anomaly detection has many applications in many different fields. Examples of practical applications are credit card fraud detection, network intrusion detection and health monitoring.

Many of the applications in these fields rely on knowing what anomalies look like, which they acquire via supervised learning. In cybersecurity however, an anomaly can have many different forms. While we do have examples of cyber attacks, it is no use to apply supervised learning if one attack is not similar to another. Hence, we shift to unsupervised methods.

In this thesis we propose an unsupervised method of finding anomalies in sequential network data. This method performs a statistical analysis per user on the network to find behavior that is not in line with a user's regular behavior. It dynamically identifies small sequences of actions on the network and compares these with each other.

Using a real world dataset by the Los Alamos National Laboratory, we demonstrate the advantages of using this method over static sequence identification. Finally, we do a careful performance evaluation and show the potential of this methodology.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter will give an introduction to the problem that will be approached in this thesis. It describes the general problem of anomaly detection. It describes our approach to this problem and gives the theoretical foundation that will be required to implement it. Lastly, it will give an overview of the layout of the rest of the thesis.

1.1 General Problem

The problem of anomaly detection is being researched in many different domains. Different anomaly detection techniques have been developed to solve many different problems. Some of these techniques are developed for a specific domain, while others have applications in many. Anomaly detection in computer networks is one of these domains. It aims at finding anomalies in network traffic or human behavior on computers. The main goal of anomaly detection in networks is to find potential network intrusions, the leaking of sensitive information to unauthorized destinations or other malicious behavior. Many different techniques already exist, applied in both real-time systems as well as in hindsight setups [CBK09].

In this thesis, an approach based on the detection of anomalous subsequences in long sequences of network-events will be explored. These subsequences are also referred to as *discords* [BLF+07] [FLKL06] [KLF05] [YKR07]. The two key challenges in finding these subsequences, as pointed out by Chandola, Banerjee and Kumar [CBK09], are:

- The size of anomalous subsequences. These we do not know up front, so a method of finding and evaluating subsequences of variable length is required.
- Creating a model of normalcy. As the input sequences (probably) contains anomalies, it can prove difficult to establish a model of what can be labeled as normal, or non-anomalous, behavior.

1.2 Our Approach

Whereas most of these techniques use a specified window-size w to define the size of such a subsequence (e.g. [GAS05]), the contribution that this thesis will make is the exploration of a method that does not pay attention to the length of these subsequences. Instead, the main factor in defining subsequences will be time-gaps in-between subsequences.

The approach we will take is as follows. First, we will give a definition of a subsequence within a long sequence. Then, we will define a distance-measure between subsequences that is based on the Jaccard similarity [Jac12]. Lastly, we will describe the method of applying Local Outlier Factor [BKNS00] to find a degree of outlier-ness of each subsequence. The described methodology will be applied to a dataset of the Los Alamos National Laboratory (see Chapter 2). We will evaluate the results to conclude how promising the described approach is.

1.3 Theoretical Overview

This section will go over the two theoretical concepts that we need to grasp in order to understand the definitions given in Chapter 3. First, we will make the distinction between outliers and anomalies. Afterwards, the concepts Jaccard distance and the Local Outlier Factor will be discussed.

1.3.1 Outliers vs. Anomalies

In this thesis, sequences will be referred to as being an anomaly or being an outlier. These two are not to be confused, as they are not the same.

Outliers are points of data that are different from most of the other points of data based on some measurable factor. No other information about the data, like context, is necessary to identify an outlier.

On the other hand, an anomaly is an outlier that is different from other outliers because of what it represents. If we take credit card fraud as an example, an outlying transaction or group of transactions are not necessarily fraudulent. Identifying one as being fraudulent requires more context and domain knowledge.
Anomalies are always outlying in some way, but an outlier is not always an anomaly.

The goal of this thesis will be to find anomalies. In order to do so, first outliers must be found. These will then have to be looked at by a domain expert.

1.3.2 Jaccard Similarity

The Jaccard similarity is a similarity-measure of two sets and was first-used by Paul Jaccard in 1901 [Jac12]. It compares the number of elements that the two sets have in common to the total number of elements of the two sets. Therefore, the resulting similarity coefficient can also be explained as the degree to which they overlap. It

makes use of the intersection and the union of the two sets and is therefore also referred to as the Intersection over Union. The Jaccard Similarity is defined as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$$

where A and B are two sets. If both sets are empty, then $J(A,B) = 0$. Also notice that $0 \leq J(A,B) \leq 1$.

If $J(A,B)$ tells us how similar A and B are, we can also arrive at a measure for distance by defining how different they are. This is also called the Jaccard Distance, and is defined as

$$d_J(A,B) = 1 - J(A,B)$$

It is this statistic that will be used to describe the distance between subsequences (see also Section 3.3).

1.3.3 Local Outlier Factor

The Local Outlier Factor (from here on out referred to as *LOF*), as described by Breunig, Kriegel, Ng and Sander [BKNSoo], is the second concept that will be used in this thesis.

The *LOF* is a degree to which an object is an outlier with respect to other objects. To calculate the *LOF* for every datapoint in a set, the density around the point is compared with the density of the neighboring points. As pointed out by the authors, it is different from many other approaches in that it gives a degree of outlierness, not just a binary value. The 'local' part of the term refers to the fact that the method bases the factor on how isolated an object is with respect to the neighborhood of that object.
One important variable that needs to be specified when using this method is the integer *MinPts*. This variable is used to calculate the *k-distance* of the objects, which is the distance to the *MinPts*-th nearest neighbor. The authors of the paper report that it is best to use a range of *MinPts* values for each object and pick the highest *LOF* found to be the final *LOF* for that object.
For the formal definition of how the *LOF* is calculated, we refer to the original paper [BKNSoo].

1.4 Thesis Overview

First, the dataset that we will be using our methodology on will be explored in Chapter 2. It will give some background information on the origin of the dataset. Also, it will inspect the files of the dataset one by one and point out inconsistencies and mention properties worth noting. Definitions for this research will be given in Chapter 3. It defines a subsequence and also the methodology of comparing sequences using the Jaccard Distance. The experiment that will be performed on the dataset is defined in Chapter 4. We will give the context of the experiment, as well as a complete description of how we will perform and evaluate the described methodology. The results of the experiment and an interpretation will be given in Chapter 5. In Chapter 6 we will make a conclusion based on the results in the previous section. We will answer the question whether or

not the method we have defined is effective. Finally, some pointers for future research will be given in Chapter 7.

Chapter 2

Exploring the Data

This section will go more in-depth on the contents of the Los Alamos National Laboratory dataset. It will give a brief introduction to the origin of the data. Then, it will go through the five different files and discuss the potential value they might have for anomaly detection, as well as point out some inconsistencies and missing parts that need to be taken into account when using the data. Also some visualizations will be shown in order to improve our understanding of some of the patterns that are present within the data.

2.1 Origin

The dataset [Ken15] on which we will perform our anomaly detection is one that comes from the Los Alamos National Laboratory (from here on out also referred to as LANL). The LANL is a laboratory based in Los Alamos, New Mexico. The laboratory falls under the United States Department of Energy and was originally founded during World War II. The purpose of this laboratory was the research into and design of nuclear weapons. Nowadays, the laboratory states its mission to be "*solving security challenges through scientific excellence ... deliver science and technology to protect our nation and promote world stability*" [mis]. The laboratory consists of over a thousand different facilities and has a total of 11,200 employees.

The dataset contains 58 consecutive days of network data collected from five sources within LANL's internal computer network. As the website states: "*...the data set is approximately 12 gigabytes compressed across the five data elements and presents 1,648,275,307 events in total for 12,425 users, 17,684 computers, and 62,974 processes*" [dat].

2.2 Contents

The dataset consists of five different files. Each of the files contains information on a certain kind of activity, either on the LANL network or on a single computer connected to the network. All files are in the Comma-Separated-Value format. Computer-, process- and user-names have been anonymized. All of these values have

been replaced by an ID, which is a letter followed by an integer. This ID starts with a 'P' if it represents a process, 'U' if a user, 'C' if it represents a computer and 'N' if it represents a port. These ID's represent the same instance in every entry in all files. What follows is a description and a short analysis of each of the files. All of these files can be found on the LANLs website [Ken15]. In the following parts of the paper, the content of these different files will be named 'subsets' of the data. Also, entries in the dataset will be called 'tuples' and 'events' interchangeably.

2.2.1 Authentication Data

The authentication data of this dataset is contained in the 'auth.txt.gz' file. Every line in the file represents an authentication event. These events are collected from either individual desktop computers that run a version of Windows, servers or Active Directory servers. The events are of the form <time, source user@domain, destination user@domain, source computer, destination computer, authentication type, logon type, authentication orientation, success/failure>.

An example of entries in this file can be found in Figure 2.1.

```
1,C625$@DOM1,U147@DOM1,C625,C625,Negotiate,Batch,LogOn,Success
1,C653$@DOM1,SYSTEM@C653,C653,C653,Negotiate,Service,LogOn,Success
1,C660$@DOM1,SYSTEM@C660,C660,C660,Negotiate,Service,LogOn,Success
```

Figure 2.1: Example of authentication data.

Good to note here is that not all fields are always known or provided. For example, if we look at the event

```
55587,U3274@DOM1,U3274@DOM1,C8787,C2327,?,?,TGS,Success
```

we see that the 'Authentication type' and 'Logon type' -fields are filled with a question mark. Since all the data has been anonymized, it would not make sense to remove these fields from the data set for security reasons. The authors of the data state that the fields that are filled with a question mark did not have a valid value. Some likely scenarios are that these fields were lost during the preparation of this data for the public. Another reason may be that this data was never collected, or was collected incorrectly. Whatever the exact reason, the fact that some of these fields might not contain a value must be taken into account when applying anomaly detection methods.

2.2.2 Process Data

The process data can be found in the 'proc.txt.gz' file. Every line represents a process start- or stop event, performed on either an individual Windows-desktop computer or a server. The processes can be started by both the computer itself (root) or by a user that is currently using that computer. These events are tuples of the form <time, user@domain, computer, process name, start/end>.

See Figure 2.2.

```
1,C553$@DOM1,C553,P16,Start
1,C553$@DOM1,C553,P25,End
1,C553$@DOM1,C553,P25,Start
```

Figure 2.2: Example of process data.

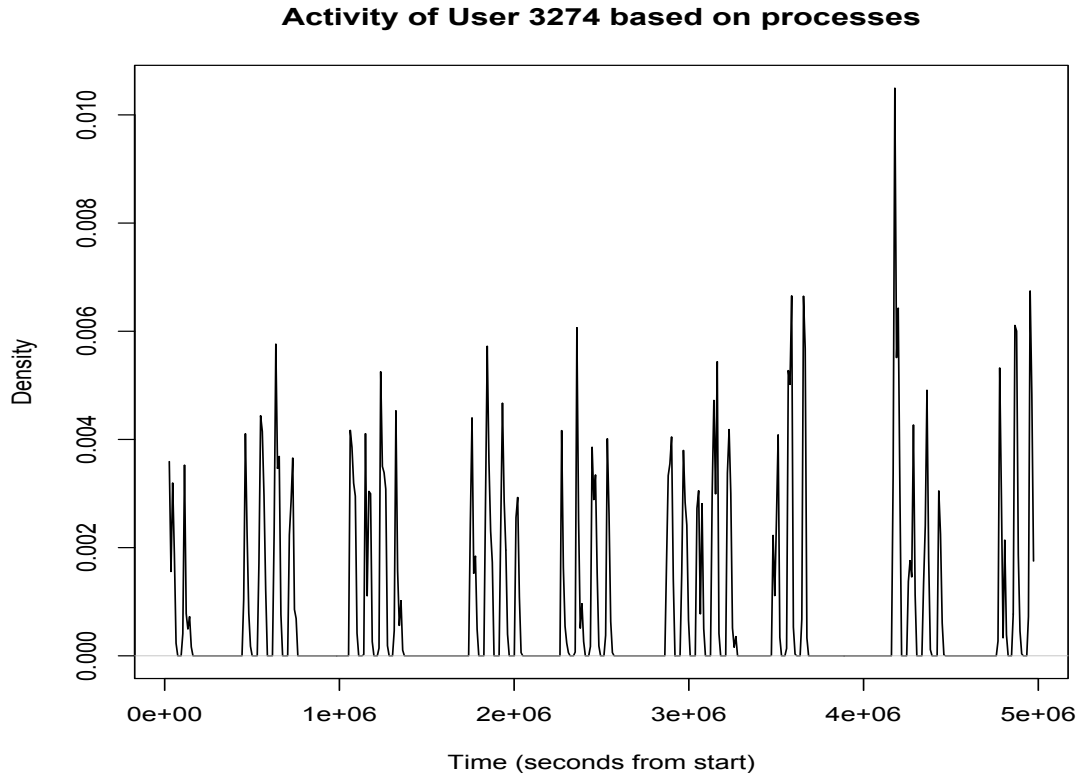


Figure 2.3: Plot showing both the process activity and the authentication activity of user U3274.

One thing to note concerning this file is that not all process that have an event that starts them, have a record of that process ever being stopped. For example, there is no record of the event

```
30233,U3274@DOM1,C8787,P107,Start
```

being stopped, neither by the user nor the computer. Like this event, some processes also have no record of ever being started. So, during our process of anomaly detection, it has to be kept in mind that we can not reliably reason from process run time or the stop events in itself.

To increase our understanding of this dataset, we decided to visualize the process-data of a single user. In order to do so, all timestamps of process-events of a single user were extracted from the dataset and plot into a kernel density plot. See Figure 2.3.

If we analyse the plot with the knowledge that we have of the dataset, we can say a few things. We know the dataset spans a time-frame of 58 days, as this is a given. This means that it spans approximately 8,3 weeks. The amount of peaks we can count in the plot is equal to 33. If we multiply the amount of weeks by 4, we get approximately 33. Taking into account the fact that the peaks are grouped in groups of 4 and also the distribution of these groups, we can conclude that each group of 4 peaks equals a 4-day workweek.

```
1,9,C3090,N10471,C3420,N46,6,3,144
1,9,C3538,N2600,C3371,N46,6,3,144
2,0,C4316,N10199,C5030,443,6,2,92
```

Figure 2.4: Example of flow data.

```
31,C161,C2109
35,C5642,C528
38,C3380,C22841
```

Figure 2.5: Example of DNS data.

An interesting observation is the consecutive groups of 5 and 3 peaks and the relatively large peak of the first day of the next group of peaks. This might be the planning of an extra long weekend, after which the amount of work of this specific user has piled up. This explains the increased amount of activity at the beginning of the next group.

2.2.3 Flow Data

The flow data of this dataset is in the 'flows.txt.gz' file. On every line, it contains a data flow event from one computer to another. The data was collected from some central routers of the LANL network. These events are also in the form of a tuple, which is of the form <time, duration, source computer, source port, destination computer, destination port, protocol, packet count, byte count>. See Figure 2.4.

Contrary to the authentication and process data, this data shows the duration of a flow event and is therefore one of the only types of data in this set that does not 'miss' any data. Or at least, we would not know if any of this data was left out or has gone missing during the process of preprocessing the data. This makes this data a very reliable source for basing conclusions upon concerning anomaly detection.

2.2.4 DNS Data

The DNS data of this dataset can be found in the 'dns.txt.gz' file. It contains DNS-lookup events. The data in this file is collected from the central DNS-servers of the network. Every line contains a lookup event, which is a tuple of the form <time, source computer, computer resolved>. See Figure 2.5.

This file contains some useful information. A DNS-lookup event potentially has a high correlation with other types of data contained in this dataset. For example, it is very likely that a DNS-lookup event of a computer to a resolved computer is succeeded by an authentication event or data flow event between those computers. Unfortunately, not much context information is given on the nature of this data. Therefore we can not directly say whether for internal connections a lookup-event is generated, or whether there might be some caching of IP addresses (which is most-likely the case). This would make such relations much less consistent.

```
151648,U748@DOM1,C17693,C728
151993,U6115@DOM1,C17693,C1173
153792,U636@DOM1,C17693,C294
```

Figure 2.6: Example of redteam data.

2.2.5 Redteam Data

The last file of this dataset is the 'redteam.txt.gz' file. It contains events from the 'auth.txt.gz' file that are flagged as redteam compromise events. As described by N. Adams and N. Heard: "*These events were intentionally created to test the security of the computers and user accounts within the enterprise network by a group of authorized hackers, commonly known as a red team*" [AH16]. It contains only the necessary information to link it to a single event in the authentication data. Every line represents an event and is a tuple of the form <time, user@domain, source computer, destination computer>. See Figure 2.6.

This data could be used in this research in two ways. The first is in optimizing the methods that we will use to find anomalies, to also find these specific events. This would almost be equal to applying supervised learning. As this research aims to inspect the performance of an unsupervised method to find anomalies in this data, using the redteam data in this way would be counterproductive.
The second option for using this data is to see whether this unsupervised method would come up with these redteam events as being anomalies. While this would be an interesting action to perform, we have to be careful in using the outcome of this comparison in the evaluation of the results of this method. For example, if none of the events that have been flagged as outlying by this method contains a redteam-event, this does not necessarily mean that the events that are flagged by our method are not outliers.

Chapter 3

Definitions

This chapter will give the foundations for this research. It will describe how our data will be organized in preparation for our experiment. Also, a definition will be given for how we define sequences in the data, as well as a definition for how to compare these sequences.

3.1 Data Organization

This section will briefly describe the way in which the data is organized. This will provide the reader with a better understanding of the line of thought in the next section. As we have seen in the previous chapter, the format the data comes in is five files with each containing a different aspects of behavior within the system. As will be explained in the next chapter, the perspective from which we will be inspecting the dataset is the source of any initiated behavior in the system.

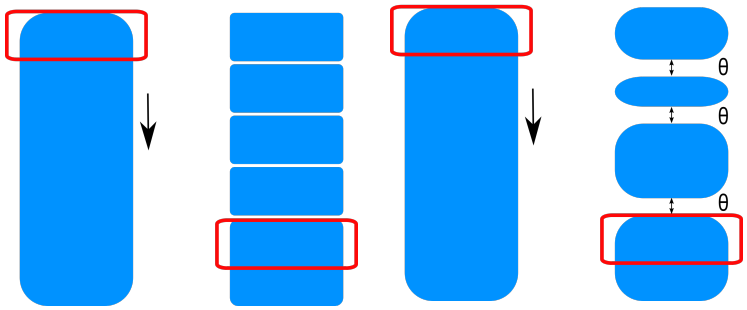
For example, if we take this random line out of the decompressed 'auth.txt.gz'-file

```
189732,U6680@DOM1,U6680@DOM1,C612,C612,?,Network,LogOff,Success
```

the actor of this specific behaviour is the 'source user' part out of the 'source user@domain' field (column 2).

As this source will be the primary attribute of interest, it would only make sense to sort the data and group every entry according to its source user. Still, we need to keep the separation between each different subset. This leads to a data structure were all data has been sorted per source user. More concretely, every user has its own folder, containing plain text files that each contain data on a different subset of which that user is the actor.

It has to be noted that the altered organization of the dataset has no impact on results or any other definitions given in this chapter. The effort of regrouping the dataset in the described way is mostly performed to increase the ease in which results can be gathered, resulting in less complex programming work and a decrease in RAM-usage.



(a) Fixed window-size (b) Time-gap

Figure 3.1: Illustration showing the use of a fixed windows size (a) and a minimal time-gap between events to identify subsequences (b).

3.2 Sequence

The approach taken to identifying outliers in this dataset completely relies on the identification of different sequences of data. In order for this to work, these sequences need to consist of elements that are closely connected through being part of the same set of actions that is taken to arrive at a certain outcome. More concretely, a sequence needs to represent a certain part of a user's workday that can, as a whole, be labeled as being an outlier.

First, we need to formally define the format of a sequence in terms of the dataset. Since the dataset consists of tuples that are part of different subsets of this dataset, it only makes sense to let the sequences consist of these tuples.

The primary element that we want to identify in this thesis are moments where certain users perform actions that are not coherent with the rest of their actions. It is therefore important that the source users of all of these events in a sequence is the same, so that we can compare them.
Also, to fully describe all the activity of a single user or computer, tuples of all the different subsets must be included in these sequences.

Two randomly selected tuples by the same source user do not necessarily have to be related in any way. The dividing element here is time, for actions that are closer in time will have a higher chance of being part of the same sequence of actions. Therefore, it is very important that our definition of a sequence reflects this.

As already mentioned in Section 1.2, many of the existing techniques make use of a window of fixed size to identify sequences in a long sequence. These sizes can be measured with either the amount of events or with time. In Figure 3.1a we see an illustration of this. The red box illustrates the window size, the blue boxes the sequence. We move through the sequence and

In this thesis we will use a different approach to identifying sequences in a long sequence. Instead of having a fixed window size, we allow the size to vary from sequence to sequence. The factor that will determine where to cut the long sequence will be the size of the time gap between events. We introduce a minimal time gap θ that represents the minimal difference in time between two consecutive elements of the sequence. See Figure 3.1b. As an example, take a look at the following part of the long sequence of user U3274:

```
...
26813,U3274@DOM1,C8787,P247,Start
27199,U3274@DOM1,C8787,P4,Start
27345,U3274@DOM1,U3274@DOM1,C4784,C4784,?,Network,LogOff,Success
30233,U3274@DOM1,C8787,P107,Start
30245,U3274@DOM1,C8787,P107,Start
30268,U3274@DOM1,U3274@DOM1,C625,C625,?,Network,LogOff,Success
...
```

Suppose we have set θ to be 900 seconds, or 15 minutes. The difference between the timestamps of the first two events is $27.199 - 26.813 = 386$ seconds, so here we do not cut. However, the difference between the third and the fourth event is $30.233 - 27.345 = 2.888$ seconds. This is larger than our θ, so this is where we cut. The current subsequence ends here and a new one starts.

The origin of the idea for this approach comes from the notion that humans take breaks or small pauses between activities. Examples of these pauses are coffee-breaks, lunch, or even a small 5-minute break to stretch the legs. These small breaks are a natural division between events in sequences. An advantage of this approach is that we have a bigger chance of actually capturing all events, and just those events, that are part of a group of actions that can be seen as an episode of events. Using fixed window size is a more rigid way of trying to find subsequences and may lead to having to overlap sequences in order to capture a possible sequence.
The downside of this approach is that we have to deal with comparing sequences of different lengths. This can prove to be very difficult, as will be discussed in the next section.
One last thing to note is that we do not know when the collection of the data began. For example, it might have started in the middle of a day. The same goes for the end of the collection. This means that our first and last subsequence of the long sequence might be incomplete. Therefore, we will not consider these to be full sequences and delete them from any use in experiments.

To summarize, the definition of a subsequence in a long sequence will be the set of events from the dataset which are close to each other in the time-space. We 'cut' the long sequence in smaller subsequences where there is a time gap between events that is equal to or greater than θ.

3.3 Sequence Similarity

Now that we have defined the format of sequences, we can begin do define the way in which we can compare them.

There are two factors to keep in mind while constructing a methodology of comparing the sequences. The first is the content of the sequences. In order for two sequences of events to be similar, the processes that have been started need to be similar, the amount of data that has flowed from the source computer to a destination computer need to be similar and the user must have produced similar authentication events.

The second factor is the factor of time. If we have two sequences that are completely equal content-wise, they can still be completely different from each other if the order of the events is different. Therefore, the place in time that a specific event has relative to other events in the same sequence needs to be taken into account.

3.3.1 Content

To find a method that will compare two sequences content-wise, we can turn to the Jaccard similarity as discussed in Chapter 1. A big advantage of using Jaccard similarity over other methods is that it naturally works with sequences that are of different size.
To make this work, we need to define 'items' in the sequences that we can compare. Since events are the items in our sequences, we let these events be the items in our sequence. Also, we need to define a way to determine whether two items are equal. One thing that is immediately clear, is that two items that do not belong to the same subset are not the same.
This leaves us with comparing items from the same subset. For some subsets, like the process data, it can immediately be seen whether two items are similar. For example, if we take the event

```
30233,U3274@DOM1,C8787,P107,Start
```

we can immediately see that it is not equal to the event

```
4971966,U3274@DOM1,C8787,P51,Start
```

for the processes are not the same. However, comparing these events raises the question: what fields should and should not be taken into account when comparing events? This differs per subset of the data, so these fields need to be defined for every subset. Most of these are pretty straight-forward, like for example the process number and the timestamp. For others it is not so easy to say whether they should be considered.

As an example, consider the 'computer' field in a process event. Should this be a factor when comparing sequences? If we, perhaps in further research, decide to compare sequences of different users with each other, we definitely need to keep this field out of the comparison. If we compare sequences of the same user, it might still pose a problem. If a user decides to work on a different computer for one day, as a result, all process data is then seen as not similar, which does not seem a desirable property of our definitions here.

For both the authentication as well as the process data, we have defined whether a field will be part of the comparison or not. See Table 3.1. These two subsets will be the main focus of the experiment described in Chapter 4.

Now, we can define two events to be equal when all their considered fields are the same. Notice that our subset consideration allows for duplicate elements in the same set. For example, it might contain two elements that refer to a Process 21 start event. As a consequence, we are actually working with multisets, not with regular sets. This needs to be taken into account when applying Jaccard.

The last hurdle in defining similarity between sequences is then to define whether the Jaccard Distance should

Subset	Field	Considered in similarity?	Motivation
Authentication/Process	time	No	Is never equal, so no point in comparing
Authentication/Process	(source) user@domain	No	The data is sorted according to the source user
Authentication	destination user@domain	Yes	No matter the computer you are working from, authenticating to a certain user is still the same
Authentication/Process	(source) computer	No	This can differ without the sessions being inherently different
Authentication	destination computer	Yes	Same as destination user@domain
Authentication	authentication type	Yes	This field is part of the main characteristics of the event
Authentication	logon type	Yes	Same as authentication type
Authentication	authentication orientation	Yes	Same as authentication type
Authentication	success/failure	Yes	Same as authentication type
Process	process name	Yes	This field is part of the main characteristics of the event
Process	start/end	Yes	Same as process name

Table 3.1: Subset fields in consideration for the similarity.

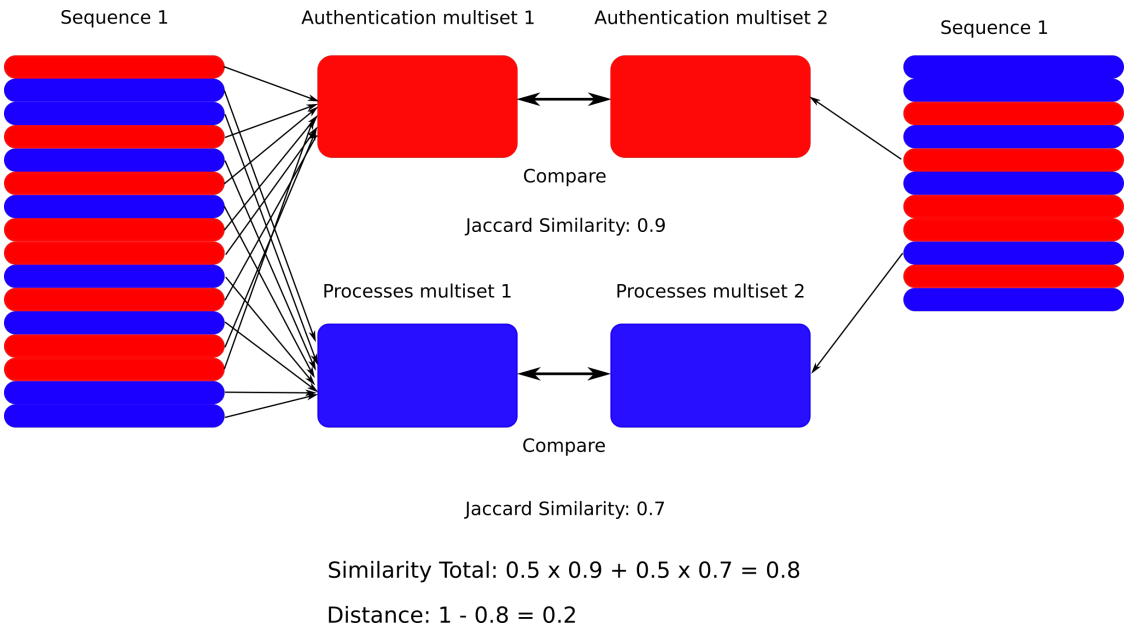


Figure 3.2: Illustration of the defined methodology to compare sequences. All events in the sequences are put into the multiset of their own kind, after which the multisets are compared with multisets of other sequences.

be calculated over all events in a sequence (and thus over all subsets) or whether we need to calculate this over each individual subset and take the weighted sum. We can easily make this decision once we have noticed that the authentication subset per user is almost always bigger (sometimes even more than ten times!) than the process subset per user. This would lead to an over representation of this subset in the calculated similarity coefficient if we calculate it over all subsets. Therefore, it is more convenient to go with the second option. We have now defined a method of comparing sequences content-wise. Figure 3.2 gives an example of this methodology. We can see how two sequences are transformed into two multisets. Authentication events here are red, process events blue. After this transformation, these multisets are compared with the multisets of the other sequence, which give similarity coefficients of 0.9 and 0.7. The total similarity coefficient is the average of these, which gives us 0.8. Because later on we will use the distance between sequences, we also calculate the distance as defined be Jaccard (see Section 1.3.2). This gives us 0.2.

3.3.2 Time

Comparing the time series of two sequences will prove to be much harder. An example of a method to do this is Dynamic Time Warping [Mö7], a technique much used in speech recognition to account for different speaking speeds.

The main problem with this method is that it assumes that we have two time-sequences with a regular interval and at each interval a value. Both of these are not present in our dataset. The time at which our events take place is anything but regular. Assuming we would be able to fix this, it would still leave us with the problem that we do not have mere 'values' that we can compare. We are dealing with events. If we want to make this method work, we need to define a cost-matrix that would define the cost of substituting an event for another event. For reasons described in the next section, this would not really work with this dataset. It is for these

reasons that we will not use dynamic time warping for this research.

3.3.3 Hybrid approaches

Also, there exists a hybrid approach to this problem, namely the Levenshtein distance [Lev66]. This method compares two strings of characters and computes a distance-value that indicates how different the two strings are. Here, a cost-function or matrix defines the 'penalty' for not containing the same character at that exact location in the sequence.
However, the chance that this method will work well without much modification is small. This is because the only way that we can define a cost-matrix is if we know more about the characters in the strings. For example, if we have two sequences of events where the first event of the first sequence starts process P20 and the first process in the other sequence is P30, we do not have a way of finding out how 'bad' this is for the equality of the events. The data has been de-identified, which renders us unable to come up with a clear penalty-value using the identity of an event.
The only way in which we could still try this, is to penalize according to the frequency of a certain character, either in the whole dataset or for that specific user only. In this way, characters that do not appear frequently and are substituted, can be penalized heavily, where characters that appear relatively often can have a low penalty. Since this approach just makes the assumption that this is a correct way of creating a cost-matrix, this method will not be used as there is too much 'guessing' involved in making it work.

3.3.4 Summary

As we have seen, we can take multiple approaches when defining a similarity measure. The factors time and content can each be tackled by different (or a hybrid) approach. To come to a final conclusion on which method to use, we decided to not include the time-factor in the similarity measure. In this decision two factors played a role.
The first is that it will prove difficult to alter dynamic time warping for our purposes. Also, coming up with a proper cost matrix for the Levenshtein distance can only be done if we use a frequency-based approach. This approach relies heavily on the assumption that the frequency of a certain event says something about the importance of that event. This is an assumption we cannot support with proof and therefore this method is also not chosen.
The second factor at play here is our guess that we can make a relatively good estimation of the similarity of sequences based on the content, given that our sequences are not too big (therefore having a low chance of containing many different events). This means that we will be using a version of the Jaccard similarity to define similarity, as described in Section 1.3.2.

To summarize, we define the sequence similarity to be equal to the Jaccard similarity and the distance between sequences to be equal to the Jaccard distance. This is done by comparing events of sequences, where two sequences are considered equal if all considered fields (Table 3.1) are equal. The Jaccard similarity coefficient

of events of a single subset is then calculated, after which the weighted sum of all indices determines the final distance between the sequences.
The order in which the events of sequences have occurred will be beyond the scope of this research.

Chapter 4

Experiment

This chapter will describe the experiment that will be done in this research.
There are several goals we try to reach with this experiment. The first goal will be to formulate a method of finding anomalies in the LANL dataset. The second is to implement this method efficiently. The third is to execute it and to gather our results. The final goal is to analyze these results, where we try to evaluate the effectiveness and efficiency of this method. If possible, we can generalize our findings to cover not only the LANL dataset, but datasets of internal network data in general.

4.1 Methodology

The general outline of what the methodology that we will use is easy to understand. We want to find subsequences per user, compute distances between the subsequences of that same user and then use the Local Outlier Factor to find outliers.

If we zoom in, we see that there are some variables that we need to define in order to use these methods.

The first is *which* users we want to evaluate. For this research, we want to restrict ourselves to analyzing human behavior on the network. This implies that we will only be analyzing users with a user-id that starts with a 'U'.

The first parameter is the θ in the creation of sequences, as defined in Section 3.2. We do not want this number to be too big, as this would allow for very big sequences of which we can not guarantee that all events belong to a single 'session'. We also do not want this number to be too small, as we then can not guarantee that the number of events that a single sequence contains will be too small. This would lead to a sequence that does not cover a complete session. For this experiment, θ will be set to be 3600, which is equal to an hour. This value seems to be a sensible time for a break between sessions where a human uses a computer (think of lunch, a meeting, etc.). Given that calculations may take a long time (see Section 4.2 for more details on the implementation), and also keeping in mind the time-frame for this thesis, this is the only *theta* we will evaluate.

As we have now defined our parameters for finding subsequences, we need to think about the weight that each individual subset will have in the calculation of the distance between sequences. The initial methodology that we describe here will only take into account the process and the authentication subsets. This allows for a more easy evaluation of the results, so that we have to be less careful in formulating our conclusions. Also, it makes it possible for us to, if it seems promising, add other subsets in future. As we do not know how much information on anomalies each subset contains, it is most safe to set the weight of the considered subsets to be equal, therefore 0.5 each.
We then calculate the distance between all subsequences of every single user, which we put in a distance matrix.

Finally, we can think about how to apply the Local Outlier Factor methodology to our generated distance matrix to compute these values for each sequence. This will be done by using the range of *MinPts* values, as suggested by Breunig, Kriegel, Ng and Sander [BKNS00]. Using a range of values takes away some of the uncertainty that is part of choosing the right *MinPts* value, as we now consider multiple of them. The *LOF*-score for each sequence will be calculated for a range with a *MinPtsLB* (lower bound) of 15 and an *MinPtsUB* (upper bound) of 30. These seem to work best in their own research, where the number of elements per dataset is comparable to that of ours. The maximum value that is found in this range per sequence will then represent the *LOF*-score for that sequence.
Choosing the *MinPtsUB* does restrict us in which users we can analyze, as this upper bound is the lowest number of elements that can be analyzed using *LOF*. Therefore, only users with at least 30 sequences with a θ of 3600 will be analyzed. It is not directly necessary to specify a maximum number of sequences (after all, more data points means more precision on the outlier factor). However, for every extra element that we have per user, we also have to calculate the distances to all other sequences (n-1 comparisons) and calculate the *LOF* 15 times. We consider that it takes around 20 seconds to analyze a user with 70 sequences and 180 seconds for 230 sequences. In order to reduce the amount of time it will take to calculate all this, we set the maximum number of sequences of analyzed users to be 250.
Both the upper and lower bound on the number of sequences reduces the number of users that we will analyze from 10094 back to 6207. Considering that the most of them had more than 250 sequences, we can assume that we have more than halved the amount of time it takes to get our results.

4.2 Implementation

All of the programming is done in Python. For system calls we will import the *sys* library, for handling the data the *csv* library and lastly for some of the mathematical operations *numpy* will be used.

Since the calculating Jaccard similarity consist of only one division, there is no need to import any modules. The structure for comparing events also needs to be programmed, as this is a very specific operation.
There exist many modules that implement *LOF*, however, these only work with an input that consists of coördinates for each object. As our input is a distance matrix per user, we will also make our own

implementation of *LOF* to work with this matrix.
Additional scripts will be written to perform all of these actions for every user. Calculations for every user will happen in sequential fashion, but later on we have also extended the implementation to run in parallel.

One observation that we make is that the implementation of the experiment is not optimal. For example, give the fact that Python code runs 10 to 100 times slower than, for example, compiled C++ code [pyt], the obvious choice here would be to pick C++ as main language for implementation. The choice for Python is made mainly because of the ease of using an interpreted language over a compiled language during the exploratory phase. We have not shifted to C++ after completion of that phase, which, in hindsight, might have been the better choice.

4.3 Evaluation

Evaluating the effectiveness of this methodology on this dataset will be difficult. This methodology falls in the category of unsupervised learning, and we have no clear examples of anomalies within this dataset. Also since all the data is anonymized, we can not confirm whether a sequence is indeed an anomaly based on content.

One option that we have for our evaluation is to try and characterize our outlying and non-outlying sequences and compare them. To do this we can inspect the number of processes and authentications. We can compute the standard deviation for every user and compare the most outlying sequences with this deviation. This would not completely verify whether our methodology works, but it can see if there is a natural difference between the most outlying sequences and the more regular ones.

Another option for evaluation is to check whether redteam events are included in the most outlying sequences. However, doing this will be meaningless if we want to evaluate the methodology. If some of the events are included in the most outlying sequences, it does not confirm that we have found outlying sequences. On the other hand, if they are not included in these sequences, we cannot conclude that we are not finding outlying sequences. We have too little information about this data to make conclusions based on it.

To summarize, out of these two options, we choose to try and characterize the outlying and non-outlying sequences to see if there is an inherent difference between them based on size.

Chapter 5

Results

This chapter will go over the results that are gathered from the experiment described in the previous chapter. It will give visualizations of the distribution of the results, an analysis of some of the most outlying sequences in the whole set and evaluations of the methodology that we introduced.

5.1 Initial Observations

On the first run, the implementation took about 40 hours to complete all the calculations in sequential fashion. This includes writing all results to a disk. The parallel implementation ran on 32 cores and only needed 1,5 to 2 hours of calculating to complete.

The two concrete results we gathered from the experiment, are the two types of lists: one for every user, containing the LOF-scores for all their sequences, and one list with the maximum LOF-score found for every user.

Let us start by visualizing the distribution of these maximum LOF-scores, see figure 5.1.

Immediately, we can see that most of the maximum scores are close to 1. In fact, over half of our 6207 analyzed users (4767 users) have a score $\leq$ 2.0, with a minimum of 1.0. This directly corresponds with the lemma 1 drafted by Breunig, Kriegel, Ng and Sander [BKNS00], which states that the LOF-score of an element that resides within a cluster is close to 1. If the maximum scores of more than half of the users is close to one, we can consider all actions of these users to not be outlying, or at least, not as outlying as some of the other sequences that we have found. The maximum LOF-score of all sequences is 38.0

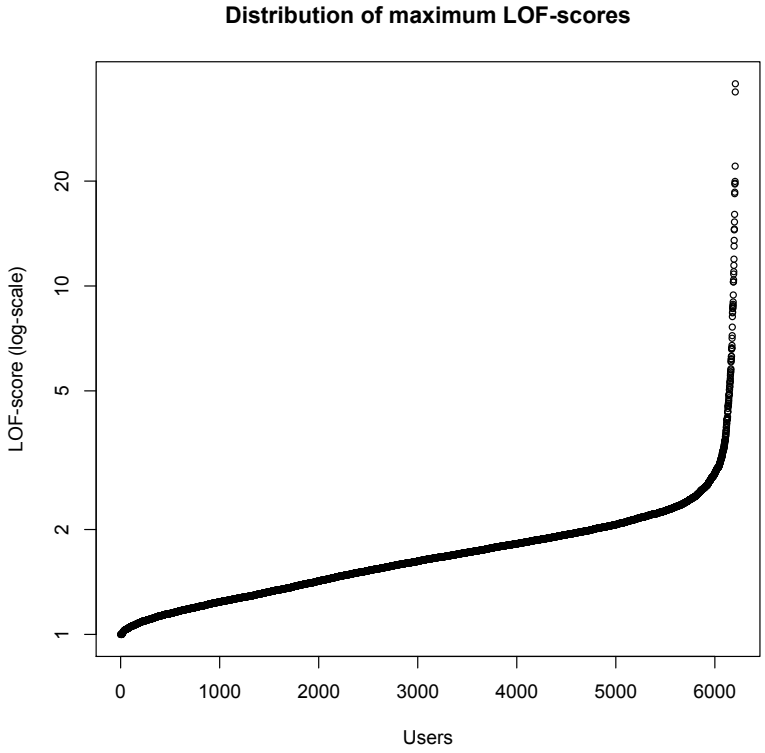


Figure 5.1: Distribution of the maximum Local Outlier Factor scores of all evaluated users.

5.2 Example

Let us examine a specific user to see what the the resulting data looks like. We pick a random user with a high maximum LOF, user U4874, and look at the first 10 subsequences. This user has a total of 82 sequences (84 if we also count the first and last sequences, which were deleted). See Table 5.1.

Interesting to see is that almost all sequences have not run any processes, except for sequence 5. Also we can make out a few different authentication patterns. Sequence 5 has been labeled with a LOF greater than 8. If we look at the sequences plainly, we can indeed see that this sequence is an outlier compared to the other sequences.

5.2.1 Evaluation: Characterization of behavior

Now that we have labeled sequences with a grade of outlier-ness, we can see or verify that the most outlying sequences are indeed different from the sequences that have a relatively low LOF. Two factors that we will look at are the number of events in a sequence and the standard deviation of the number of events for every user. In order to look at these factors, we will divide the inspected users into two groups: the users with a max LOF close to one, or < 2.0 (group 1) and those with a max LOF ≥ 2.0 (group 2). The sizes of the resulting groups are 4767 and 1440 users respectively.

The sequence size can be measured by the number of started processes and the number of authentication

Sequence	LOF	Processes started	Authentications
1	2.8801984933		'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success', 'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
2	3.65467120158		'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Fail', 'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success', 'U4874@C11714,C586,NTLM,Network,LogOn,Fail', 'U4874@C11714,C586,NTLM,Network,LogOn,Fail'
3	0.832847320947		'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
4	0.832847320947		'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
5	8.39874127828	P168,P155,P155,P658,P1007,P2103,P254,P291,P2978,P4, P485,P585,P162,P155,P125,P1345,P1473,P1651,P1656,P1660, P2487,P292,P560,P750,P155,P153,P155,P107,P155,P107, P155,P125,P155,P155,P155,P107,P155,P107,P155,P125, P155,P107,P155,P155,P372,P155,P155,P372,P107,P155, P155,P125,P155,P155,P155,P155,P364,P155,P372,P155, P152,P152,P152,P152,P155,P1262,P155,P1262,P155,P1352,P152	'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success', 'U4874@C11714C11714NegotiateInteractiveLogOnSuccess', 'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
6	0.832847320947		'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
7	0.832847320947		'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success', 'U4874@C11714,C11714,?,Unlock,LogOff,Success'
8	0.832847320947		'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
9	0.832847320947		'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
10	2.42234083801		'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Fail', 'U4874@C11714,C11714,?,Unlock,LogOff,Success', 'U4874@C11714,C11714,Negotiate,Unlock,LogOn,Success'
...	...	...	...

Table 5.1: The first 10 subsequences of user U4874 and their LOF-scores.

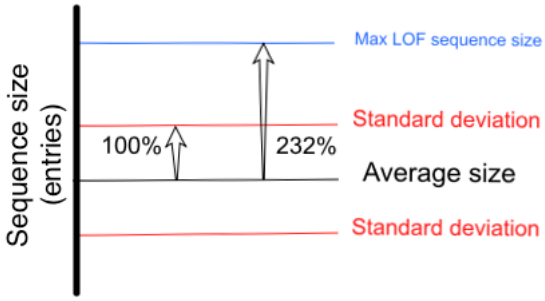


Figure 5.2: Visualization of the measure that compares different users based on sequence size.

events. We look only at the sequence with the max LOF for every user. We calculate the average number of processes and authentications for both groups, see Table 5.2.

Group	Average # of processes	Average # of authentications
1	159.99	2004.98
2	449.69	3070.83

Table 5.2: Sequence size for group 1 and group 2.

We see that both the amount of processes and the amount of authentications is higher for group 2. This means that the most outlying sequences are indeed larger in size. However, these statistics alone do not verify that the size of these sequences makes them the most outlying. It can be the case that outliers are more easily found for users that tend to have more events in a sequences in general. That does not necessarily mean that outlying sequences have more events than the more regular sequences per se . To find out if that is that is true, we will look at the second factor.

The standard deviation of the number of events in a sequence can give us the decisive answer on whether outlying sequences have more events or if sequences with more events are more often outliers. If we take the standard deviation as our base (100%), we can get the relative distance to the average sequence size per user. See Figure 5.2 for a visualization of this measure. If we compare the average of the two groups, we can say whether sequence size plays a role.

In Table 5.3 we present the average difference between the max LOF sequence size and the average sequence size for both groups. We can see that the max LOF sequences in the second group are substantially bigger in terms of both the number processes and the number of authentications. Max LOF sequences in the second group are on average almost three times the size in terms of processes, and 1.5 times in terms of authentications.

Group	Average distance processes (%)	Average distance authentications (%)
1	200.21	219.82
2	482.10	354.18

Table 5.3: Average difference between max LOF sequence size and average sequence size per user for group 1 and group 2.

We can now conclude that the outlying sequences that we have found using our methodology have one characteristic in common, which is that they are bigger in size than the more regular sequences. This characterization of regular and irregular sequences provides us with a measure to confirm a distinction

between the two types of behavior.

Chapter 6

Conclusions

This chapter covers the conclusions of this research. It will give some final insights on the methodology used and whether or not it is suitable to find anomalies in this dataset.

By observing the data presented in Section 5, we can safely conclude that we have found each users' most outlying sequence. We have characterized two groups of outlying sequences and we can conclude that really outlying sequences in terms of LOF are measurably different from regular sequences.

However, to conclude that we have found anomalies would be an overstatement. Finding anomalies requires domain knowledge and the comparison of sequences with sequences of other users. Also, this dataset is anonymized on many fronts, making it hard to say that an outlier is an anomaly.

Finally, we do conclude that this methodology is promising for finding anomalies. If the methodology can be expanded to compare all sequences with each other and to include more data sources, we can present domain experts with interesting findings that might just prove to be anomalies.

Chapter 7

Future Research

This chapter will cover some of the topics that can be examined in order to improve sequence-based anomaly detection (in this dataset).

As mentioned in Chapter 4, not all subsets were considered in the experiment. In an attempt to improve on the methodology as described in this paper, one can try to make flow- and DNS data part of the sequences. A challenge that will arise is the selection of new comparators, as the Jaccard similarity will not suffice when comparing data flows of different sizes. Another challenge that might be encountered is setting the weights of the subsets in the similarity measure. For example, does flow data tell us more than authentication data?

In this experiment, sequences that were created were only compared with other sequences of the same user. But what if we compare sequences with all other sequences? Computationally, this would be very (if not too) heavy. Maybe it is possible to create different classes of sequences and assign sequences to a class, so that a new sequence would only have to be compared with a number of classes. No doubt, if the right implementation is found for this idea, this would yield even better results than the ones we have so far.

Finally, in this experiment we left the ordering and timing within sequences out of scope. It is very well possible that we would be able to better identify outliers if we do take these factors into account. To do so, a new similarity metric would have to be defined, which comes with many challenges, as also discussed in this paper.

Bibliography

[AH16] N. Adams and N. Heard. pages 53–54. World Scientific, 2016.

[BKNS00] M. M. Breunig, H. Kriegel, R. T. NG, and J. Sander. Lof: Identifying density-based local outliers. pages 93–104, 2000.

[BLF+07] Y. Bu, O. T.-W. Leung, A. W.-C. Fu, E. Keogh, J. Pei, and S. Meshkin. Wat: Finding top-k discords in time series database. In *Proceedings of 7th SIAM International Conference on Data Mining*, 2007.

[CBK09] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41:No. 15, 2009.

[dat] Lanl dataset website. `http://csr.lanl.gov/data/cyber1/`. Accessed: 2017-04-12.

[FLKL06] A. W.-C. Fu, O. T.-W. Leung, E.J. Keogh, and J. Lin. Finding time series discords based on haar transform. In *Proceeding of the 2nd International Conference on Advanced Data Mining and Applications*, pages 31–41. Springer Verlag, 2006.

[GAS05] R. Gwadera, M. Atallah, and W. Szpankowski. Reliable detection of episodes in event sequences. In *Knowledge and Information Systems*, volume 7, pages 415–437. Springer, 2005.

[Jac12] P. Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11:37–50, 1912.

[Ken15] A. D. Kent. Comprehensive, Multi-Source Cyber-Security Events. Los Alamos National Laboratory, 2015.

[KLF05] E. J. Keogh, J. Lin, and A. W.-C. Fu. Hot sax: Efficiently finding the most unusual time series subsequence. In *Proceeding of the Fifth IEEE International Conference on Data Mining*, pages 226–233. IEEE Computer Society, Washington, DC, USA, 2005.

[Lev66] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 1966.

[Mö7] M. Müller. Dynamic time warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer, 2007.

[mis] Los alamos national laboratory website. http://www.lanl.gov/mission/mission.php. Accessed: 2017-04-30.

[pyt] Python 3 programs versus c++ g++. http://http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=python3&lang2=gpp. Accessed: 2017-06-13.

[YKR07] D. Yankov, E. J. Keogh, and U. Rebbapragada. Disk aware discord discovery: Finding unusual time series in terabyte sized datasets. In *Proceedings of International Conference on Data Mining*, pages 381–390, 2007.