Proceedings of the of the 18th Conference on Computer Science and Intelligence Systems pp. 141–150
DOI: 10.15439/2023F4617
ISSN 2300-5963 ACSIS, Vol. 35

Discovering relationships between data in an enterprise information system using log analysis

Łukasz Korzeniowski
0000-0001-8458-9825
Nordea Bank Abp SA
Satamaradankatu 5, FI-00020,
Helsinki, Finland
Email: lukasz.korzeniowski@protonmail.com

Krzysztof Goczyła
0000-0003-3009-8988
Gdańsk University of Technology,
Faculty of Electronics,
Telecommunication and Informatics
Narutowicza 11/12, Gdańsk, Poland
Email: kris@eti.pg.edu.pl

***Abstract*—Enterprise systems are inherently complex and maintaining their full, up-to-date overview poses a serious challenge to the enterprise architects' teams. This problem encourages the search for automated means of discovering knowledge about such systems. An important aspect of this knowledge is understanding the data that are processed by applications and their relationships. In our previous work, we used application logs of an enterprise system to derive knowledge about the interactions taking place between applications. In this paper, we further explore logs to discover correspondence between data processed by different applications. Our contribution is the following: we propose a method for discovering relationships between data using log analysis, we validate our method against a benchmark system AcmeAir and we validate our method against a real-life system running at Nordea Bank.**

I. Introduction

LARGE enterprises, especially those with a long history of operation, face the challenge of modernizing their processes and adapting them to the changing environment. One of the key, and often the most challenging, aspects of such adaptation is the modernization of the enterprise's IT infrastructure. This is usually a complex endeavor, that includes exiting legacy systems, reorganizing the architecture, and harmonizing the data usage across the enterprise system. Each of these activities requires the enterprise architects team to have a good understanding of the existing IT infrastructure constituting the enterprise system, including the processes, applications participating in them, and the data being processed.

In [1] we proposed a method for discovering the knowledge about the interactions between applications in an enterprise system based on the analysis of application logs. We chose this type of analysis as the basis for our method due to some interesting properties of logs. Firstly, logging is a common practice present in IT from its very beginning, meaning that both legacy and modern applications are expected to create some sort of log (a trace of the actions executed by an application). Secondly, logs contain rich information, which blends the working application's technical and business aspects. Lastly, log entries tend to be kept up-to-date with the executed application code. All of these properties make application logs a perfect candidate for deriving the actual knowledge about various aspects of the enterprise system in an automated way.

This work was supported by the Gdańsk University of Technology and Nordea Bank

In this paper, we further explore the potential of application log analysis in terms of supplying enterprise architects with valuable information about the enterprise system. This time, we focus on the data processed by applications. We try to find correspondence between data processed by different applications, which can be treated as good candidates for reconstructing relationships between data models used by different applications of the system. This information can be valuable in many aspects – it allows for the "detection" of pieces of information used by various applications, it allows for explaining information from legacy applications by finding its correspondence to information in modern (well-documented) applications, and it allows for attaching some business meaning to information by utilizing the business part of log entries. Our work fits in the domain model extraction area of the landscape of automated log analysis proposed by the authors of [2]. Other research in this area includes ontology discovery with process mining [3], search query categorization into a predefined taxonomy using search log analysis [4], or learning expert knowledge on applying security rules based on security log [5]. Our contribution to the body of knowledge is three-fold:

- we propose a method for automated discovery of relationships between data processed by applications, using textual analysis of the log content,
- we validate our method on the benchmark system AcmeAir [6],
- we validate our method on a real-life system running at Nordea Bank.

The rest of the paper is organized as follows. In Section II, we present a formal statement of the problem. Section III describes the proposed method of log analysis. In section IV, we introduce the two datasets used for the evaluation of our method and we define the evaluation criteria. In Section V, we compare our method with alternative approaches and describe the related work. We present the conclusions and plans for future work in Section VI.

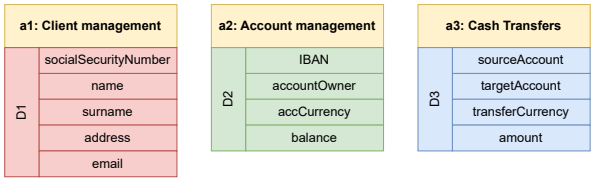

Fig. 1. An example of a fragment of an enterprise system S from the banking domain, related to the execution of cash transfers. The yellow color denotes different applications constituting the system. Red, green, and blue colors denote the data attributes processed by each of the applications.

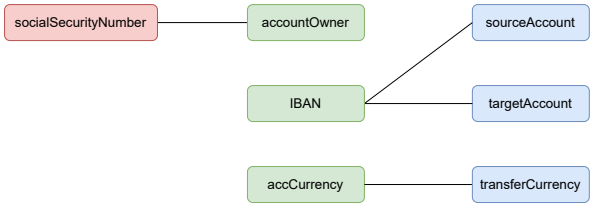

Fig. 2. A data relationship graph for the example system presented in Fig. 1. Edges represent relationships between data attributes from different applications. The coloring of nodes matches the coloring in Fig. 1.

II. Problem Statement

Let enterprise system S consists of a set of applications $A = \{a_1, \ldots, a_n\}$. Let each application $a_i \in A$ process some data represented by a set of data attributes $D^i = \{d^i_1, \ldots, d^i_{k^i}\}$. For each data attribute $d^i_k \in D^i$, we denote the set of potential values that the attribute can take by $V(i,k)$.

A fragment of an example enterprise system with applications and respective data attributes is presented in Fig. 1. The presented fragment is related to the processing of cash transfers in a bank and consists of three applications - managing clients, managing accounts, and cash transfer execution.

For any two data attributes d^i_k, d^j_l, we define their level of similarity using the Jaccard index

$$J(d^i_k, d^j_l) = \frac{|V(i,k) \cap V(j,l)|}{|V(i,k) \cup V(j,l)|} \quad (1)$$

We define a data relationship graph $G(S) = (U, E)$ as an undirected graph, where E is the set of edges consisting of all pairs of related data attributes d^i_k, d^j_l, and U is the respective set of vertices. An example of such a graph is presented in Fig. 2.

Let $L(a_i) = (l^i_1, \ldots, l^i_{k^i})$ denote the log of application $a_i \in A$, represented by a tuple of log entries.

We define the problem of discovering data relationships as follows. Having a set of application logs $L = \{L(a_1), \ldots, L(a_n)\}$, find approximate graph $G' = (U', E')$.

III. Proposed Method

A. Method Overview

We propose a method that treats a log as a text. Such an approach has several benefits. It does not require any arbitrary assumptions to be made about the log content, which results in a broader scope of the method's usability. It also does not require additional preprocessing of the log, apart from unifying the log format across applications. Our method does not require any knowledge of the underlying data attributes for each application – they are discovered from the log automatically. Fig. 3 presents an overview of the steps that our method consists of.

Our method is parameterized by the following hyperparameters:

- N – the maximum length of n-gram embedding of a token,
- ies – inner embedding similarity threshold,

Fig. 3. Overview of data relationship discovery method.

- oes – outer embedding similarity threshold,
- mt – minimum number of tokens sharing the same embedding that is required for the embedding to be considered,
- ml – minimum length for a token to be considered valid.

The following subsections describe each of the steps in detail.

B. Token Embedding

We process the log of each application $a \in A$ separately. For each log entry, we extract a list of tokens using the regular expression $[@.A-Za-z0-9\_]+$. For each token in the log entry and each $k \in \{1, \ldots, N\}$, we create its embedding as a k-gram of words [7] preceding the given token in the log line. Such an approach means that the same token can have multiple embeddings for a given value of k, depending on the context (consisting of k preceding tokens). An example of such embeddings is presented in Fig. 4. It can be noticed that "NRT", "destPort" and "miles" tokens have different embeddings for both log lines due to different neighbor tokens in each of the lines.

For each value of k and for each unique embedding e, we maintain the set of all tokens sharing this embedding within the $L(a)$, which we denote as $Emb^a(e,k)$. We denote the set of all k-gram embeddings within a as Emb^a_k, and the set of

Log entries of application a

Line 1: flightsegment = {"_id":"AA382","originPort":"NRT","destPort":"DEL","miles":"4959"}
Line 2: flightsegment = {"_id":"AA87","originPort":"FRA","destPort":„NRT","miles":"400"}

Embeddings for Line 1

Token	1-gram	2-gram
flightsegment		
_id	flightsegment	
AA382	_id	flightsegment, _id
originPort	AA382	_id, AA382
NRT	originPort	AA382, originPort
destPort	NRT	originPort, NRT
DEL	destPort	NRT, destPort
miles	DEL	destPort, DEL
4959	miles	DEL, miles

Embeddings for Line 2

Token	1-gram	2-gram
flightsegment		
_id	flightsegment	
AA87	_id	flightsegment, _id
originPort	AA87	_id, AA87
FRA	originPort	AA87, originPort
destPort	FRA	originPort, FRA
NRT	destPort	FRA, destPort
miles	NRT	destPort, NRT
400	miles	NRT, miles

Fig. 4. 1- and 2-gram embeddings for sample log entries. Yellow color denotes tokens with multiple embeddings.

MOST WIEDZY Downloaded from mostwiedzy.pl

1-gram embeddings

Embedding e	$Emb^a(e,1)$	
flightsegment	_id	
_id	AA382	AA87
AA382	originPort	
originPort	NRT	FRA
NRT	destPort	miles
destPort	DEL	NRT
DEL	miles	
miles	4959	400
AA87	originPort	
FRA	destPort	

2-gram embeddings

Embedding e	$Emb^a(e,2)$	
flightsegment, _id	AA382	AA87
_id, AA382	originPort	
AA382, originPort	NRT	
originPort, NRT	destPort	
NRT, destPort	DEL	
destPort, DEL	miles	
DEL, miles	4959	
_id, AA87	originPort	
AA87, originPort	FRA	
originPort, FRA	destPort	
FRA, destPort	NRT	
destPort, NRT	miles	
NRT, miles	400	

Fig. 5. Tokens sharing the same 1- and 2-gram embedding for sample log entries from Fig. 2.

all embeddings within $L(a)$ as $Emb^a = \bigcup_{k\in\{1,\ldots,N\}} Emb^a_k$. Fig. 5 shows groups of tokens sharing the same embedding.

For given k-gram embedding e consisting of tokens $(t_k, t_{k-1}, \ldots, t_1)$ and $l < k$, we define an l-cut operation on e, denoted as $e|l$, as follows: $e|l = (t_l, t_{l-1}, \ldots, t_1)$.

C. Embedding Optimization

The result of the previous step is ambiguous – the same groups of tokens are represented with multiple embeddings, for different values of $k \in \{1, \ldots, N\}$. To remove this ambiguity, we search for the highest value of k such that k-gram embedding of the groups of tokens represents them better than $k+1$-gram embedding. The longer the embedding, the more precisely it describes the represented tokens.

We start with the set of initial embeddings E that consists of all 1-gram embeddings. For each 2-gram embedding e, we then take the set of all tokens that it represents $Emb^a(e, 2)$. We compare this set with the set of all tokens represented as 1-cut of e, using the Jaccard index. If the index value is above the ies threshold, we substitute the $e|1$ embedding in E with e. We repeat this procedure for longer k-grams until $k = N$, each time comparing the k-gram embedding with its k-1-cut. In the end, the set E contains all the embeddings for log $L(a)$ after optimization, which we denote as $OEmb^a$. Fig. 6 shows the two consecutive steps of the optimization process.

$OEmb^a(e)$ denotes the set of tokens represented by embedding e within log $L(a)$. The above algorithm ensures that there is only one embedding (one value of k) that represents a given set of tokens. Fig. 7 presents the example outcome of the optimization process.

D. Token Filtering

We further optimize the set of embeddings. For each application log $L(a)$ and each embedding $e \in OEmb^a$:

- we discard e if $|OEmb^a(e)| < mt$,
- we define the filtered set of tokens, such that $\forall_{e\in OEmb} FEmb^a(e) = \{t \in OEmb^a(e) | length(t) \geq ml\}$.

$FEmb^a(e)$ denotes the final set of tokens represented by the embedding e within application log $L(a)$, while $FEmb^a$ represents the set of all final embeddings for application a,

k = 1

Initial embeddings

flightsegment	_id	AA382	originPort	NRT
destPort	DEL	Miles	AA87	FRA

k = 2

Embedding e	$Emb^a(e,2)$		Embedding e|1	$OEmb^a(e\|1)$		Jaccard index
flightsegment, _id	AA382	AA87	_id	AA382	AA87	1
_id, AA382	originPort		AA382	originPort		1
AA382, originPort	NRT		originPort	NRT	FRA	0.5
originPort, NRT	destPort		NRT	destPort	miles	0.5
NRT, destPort	DEL		destPort	DEL	NRT	0.5
destPort, DEL	miles		DEL	miles		1
DEL, miles	4959		miles	4959	400	0.5
_id, AA87	originPort		AA87	originPort		1
AA87, originPort	FRA		originPort	NRT	FRA	0.5
originPort, FRA	destPort		FRA	destPort		1
FRA, destPort	NRT		destPort	DEL	NRT	0.5
destPort, NRT	miles		NRT	destPort	miles	0.5
NRT, miles	400		miles	4959	400	0.5

Fig. 6. Steps of the example optimization process for embeddings presented in Fig. 5 and ies threshold of 0.9. The yellow color denotes the $k-1$ cut of the embedding. Each row in the table presents a step in the process. Green cells denote the embeddings that have been accepted and red cells denote the embeddings that have been rejected.

Initial embedding	Optimized embedding	$OEmb^a$	
flightsegment	flightsegment	_id	
_id	flightsegment, _id	AA382	AA87
AA382	_id, AA382	originPort	
originPort	originPort	NRT	FRA
NRT	NRT	destPort	miles
destPort	destPort	DEL	NRT
DEL	destPort, DEL	miles	
miles	miles	4959	400
AA87	_id, AA87	originPort	
FRA	originPort, FRA	destPort	

Fig. 7. The outcome of the example process presented in Fig. 6 Green cells denote the embeddings that have been extended as part of the optimization.

and $FEmb = \bigcup_{a\in A} FEmb^a$ is a set of all embeddings in the system.

The first optimization removes embeddings that represent only a few tokens. Such embeddings are interpreted as representations of static parts of the log entry, which are of less interest in terms of data relationship discovery. The second optimization removes short tokens, which decreases the chance of discovering accidental relationships.

E. Graph Estimate Construction

We calculate a distance matrix between all $FEmb$ embeddings using the Jaccard index as the distance measure:

$$\forall_{e_1,e_2\in FEmb, e_1\neq e_2} dist(e_1, e_2) = \frac{|FEmb(e_1) \cap FEmb(e_2)|}{|FEmb(e_1) \cup FEmb(e_2)|} \quad (2)$$

We filter pairs of embeddings based on their distance, using oes as the threshold, above which the embedding relationship is considered strong enough and should be retained. The retained embedding pairs form the edges of our approximate graph G' and respective embeddings become the vertices of the graph. Fig. 9 presents an example distance matrix

MOST WIEDZY Downloaded from mostwiedzy.pl

Final embedding	$FEmb^{a}$	
flightsegment, _id	AA382	AA87
originPort	NRT	FRA
NRT	destPort	miles
destPort	DEL	NRT
miles	4959	400

Fig. 8. The final set of embeddings for sample log entries presented in Fig. 4.

Distance	flightsegment, _id	originPort	NRT	destPort	miles
flightsegment, _id		0	0	0	0
originPort	0		0	0.33	0
NRT	0	0		0	0
destPort	0	0.33	0		0
miles	0	0	0	0	

Fig. 9. Distance matrix for final embeddings from Fig. 8, based on the Jaccard index.

for the final embeddings presented in Fig. 8. The respective graph estimate is shown in Fig. 10, which shows a detected relationship between *originPort* and *destPort* data attributes.

IV. Method Evaluation

A. Dataset Overview

We evaluate our method using two datasets:

- AcmeAir – a dataset allowing us to assess the accuracy of our method on a benchmark system, that we have full knowledge about,
- NDEASET2 – a dataset from a real-life enterprise system running at Nordea Bank.

Both datasets are described in detail in subsequent sections.

B. Benchmark Dataset

AcmeAir [6] is an open-source implementation of a fictitious flight reservation system that is commonly used as a benchmark system [8]. It consists of a web application and several restful web services that contribute entries to a common log file. AcmeAir provides very limited documentation that allows deriving only the data model (Fig. 11).

The AcmeAir documentation does not provide a component diagram but limits itself only to listing backend services with their responsibilities:

- BookingService - creating new bookings and canceling existing bookings for a given customer,
- CustomerService - creating and updating customer information, managing the customer's session,
- FlightService - allowing users for searching for flights by airports and/or departure dates and searching for flight segments.

However, based on the analysis of the system's source code, we have reconstructed missing elements of the component model, which are presented in Fig. 12.

All AcmeAir components put their log entries into a common log file in a standardized manner. Each log entry produced by AcmeAir consists of the following elements (see Fig. 13 for a log sample):

- Timestamp – date and time of creating the log entry,
- Logging level – either DEBUG or INFO,
- Source – the name of the component that created the entry,
- Content – logged message.

In our analysis, we use logs produced by a fork of AcmeAir [9] which extends the logging to cover HTTP requests/responses issued within AcmeAir. We use the AcmeAir driver – a built-in simulator of user interactions with the system – to generate a log file used in further analysis. The workload is generated in a multi-threaded manner. We use five threads to simulate concurrent actions taken by users of the web application. It is important to note that although the log is generated in a multithreaded way, log entries do not contain any information allowing one to identify a thread within which specific entries are logged. We use a log fragment of 1000000 lines (file size: 314MB). We have found that larger log fragments do not introduce more information as the patterns appearing in the log keep repeating.

As part of log preprocessing, we performed the following transformations to the log:

- we transform the content into a CSV format to be aligned with our real-life dataset. Each log entry is described by *timestamp*, *source*, and *message*.
- For the log entries where HTTP request/responses are logged, we derive the source from the HTTP endpoint. Such an approach better reflects the actual relationship between the data and the service that processes them.

Both the original and the preprocessed logs are available in [9].

C. Real-life Dataset

We used a log dataset NDEASET2 from a real-life system deployed at Nordea Bank, which we refer to as NDEASYS. As compared to the NDEASET1 dataset described in our previous work [1], NDEASET2 covers a full working week of the NDEASYS. The total size of the dataset is larger by an order of magnitude (95GB). It covers one more application and one additional process (daily reporting). We followed the same rules for removing bias as described in [1] to ensure proper diversity of the dataset, which include:

- team diversity – applications built by teams of different sizes, experiences, locations,

Fig. 10. Graph estimate for the set of final embeddings and oes threshold of 0.25.

MOST WIEDZY Downloaded from mostwiedzy.pl

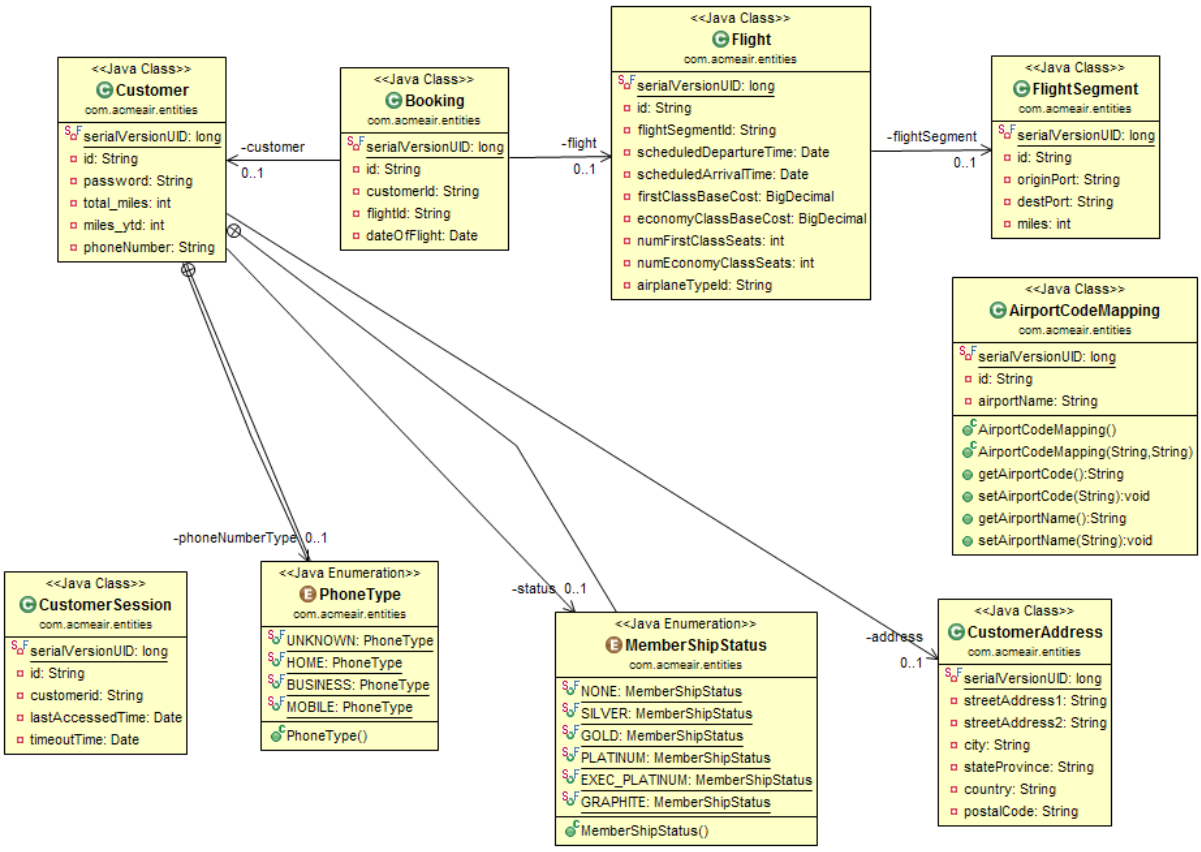

Fig. 11. Class diagram of AcmeAir system available in its documentation.

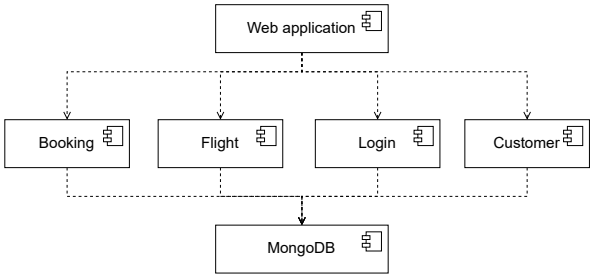

Fig. 12. Component model of AcmeAir.

```
[2020-06-07 19:56:52.676] [DEBUG] routes -
checkForValidCookie - good session so allowing
next route handler to be called
[2020-06-07 19:56:52.677] [DEBUG] routes -
booking flights
[2020-06-07 19:56:52.678] [DEBUG] routes -
toFlight:adad0c37-08f9-4f28-b0a8-
69c17c3de159,retFlight:undefined
```

Fig. 13. An AcmeAir log sample. Yellow color denotes the timestamp, magenta – the logging level, green - the source of the log entry, and cyan – the message.

- application diversity – we included dedicated business applications, purely technical components, and shared service platforms (e.g., storage or communication services),
- time diversity – applications built in different periods,
- integration diversity – applications communicating using different interfaces and exchange formats.

The bias of the dataset related to logs being collected within the same company is mitigated by the fact that Nordea does not enforce any strict rules for log creation and log content for non-regulatory logging.

The fragment of the architecture of the NDEASYS that contributes to the creation of logs within NDEASET2 is shown in Fig. 14. Applications B and D are both providing data of the same type to A, but using different formats. Application A enriches the received data with data from applications C, E, and F. The final result is aggregated as part of the daily reporting and the aggregated data are provided to application G.

Table I describes the characteristics of the NDEASYS2 dataset. As part of the log preprocessing, we unify all of the logs in the dataset to a common CSV format with timestamp, source, and message columns, which match the format of the AcmeAir logs that we use as a benchmark. Fig. 15 presents an anonymized example of log entries in NDEASET2.

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE I
CHARACTERISTICS OF THE NDEASYS2 DATASET

App	Log size [MB]	Application diversity	Team diversity		Time diversity		Integration diversity	
		Type	Size	No. locations	Dev. period	Dev. duration [months]	Integration style	Format
A	26000	dedicated	3	2	2020-2022	24	Messaging, RPI	Swift (ISO15022, ISO 20022), JSON
B	165	technical	1	2	2020	1	Messaging	Swift (ISO15022)
C	7000	shared service	2	2	2016-2020	48	RPI	JSON
D	18	technical	1	2	2020	6	Messaging	Swift (ISO15022)
E	64000	shared service	3	2	2016-2022	72	RPI	JSON
F	153	shared service	3	2	2016-2022	72	RPI	JSON
G	80	dedicated	1	2	2020	1	Messaging	Swift (ISO15022

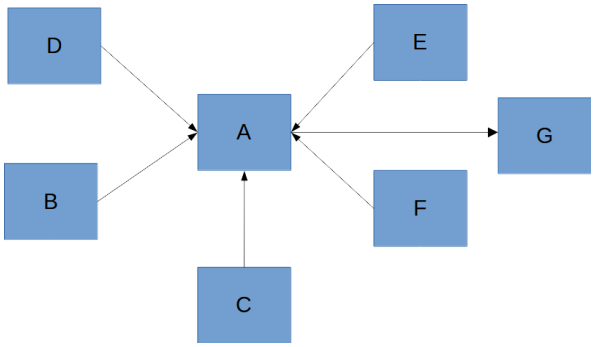

Fig. 14. The architecture of the system used for evaluation. Lines denote pairs of interacting applications, and arrows denote the direction of the data flow.

```
1647330650034,=,"
logger=c.n.t.i.r.q.QueryCallStatsObserver,
operation=QUERY_LATEST_IN_GROUP, clientId=X,
clientLibrary=null, clientVersion=null,
hostName=a01.com, correlationId=969a81d3-8ad8-4a5f-
84e8- 0868bfd65ddb, action=query_start, , domain=Y/Z,
requestCondition={""extracted.id"":
""0122318714085000""}, condition={""extracted.id"":
""0122318714085000""},
groupByFields=[Id], sortFields=[timestamp], limit=0,
payload=true"
```

Fig. 15. Example of one log entry from NDEASET2. Yellow color denotes the timestamp, green –the source application, and cyan –the message.

D. Evaluation Criteria

For each dataset, we use evaluation criteria suitable to the level of knowledge about the system that the dataset is based on. For AcmeAir, we can assume full knowledge about the system and its data model due to its simplicity and open-source nature. Based on the data model presented in Fig. 9, we construct the reference data relationships graph shown in Fig. 16, which serves as our ground truth. Arrows are introduced for clarity only. They denote the direction of the relationship and are not taken into consideration during method evaluation. Vertical lanes represent different services

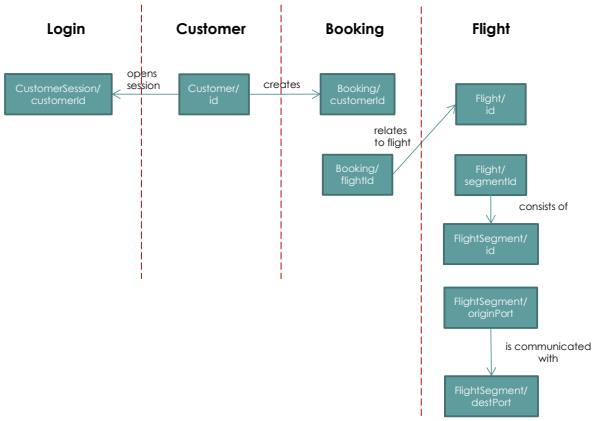

Fig. 16. Graph of data relationships in AcmeAir system, serving as our ground truth.

depicted in Fig. 10. Nodes within a lane represent data attributes managed by a given service. Each node is described by an attribute and entity it belongs to (e.g. *Customer* is an entity and *id* – is its attribute). Edges of the graph represent relationships between data attributes. Only the attributes that are related to one another are presented in the graph.

We use the F1 score of the set of graph edges to determine the accuracy of our method for the AcmeAir dataset.

For the NDEASET2 dataset, we are unable to construct a reference graph. However, our knowledge of the NDEASYS system allows us to assess, whether or not the discovered relationship is correct. Therefore, for this dataset, we use only the precision of the discovered set of edges as the measure of our method's accuracy.

E. Benchmark Results

To be able to compare the outcome of our method with the ground truth presented in Fig. 16, we need to perform two types of mapping:

- mapping of discovered data attributes to the data attributes from Fig. 11 (details of mapping are presented

MOST WIEDZY Downloaded from mostwiedzy.pl

TABLE II
MAPPING OF DATA ATTRIBUTES BETWEEN THE ACMEAIR LOG AND THE REFERENCE DATA RELATIONSHIP GRAPH

Log		**Reference data relationship graph**	
Service	**Discovered** attribute	**Entity**	**Attribute**
Booking	returnbookingid	Booking	id
	departbookingid	Booking	id
	number	Booking	id
	_id	Booking	id
	customerid	Booking	customerId
	userid	Booking	customerId
	user	Booking	customerId
	byuser	Booking	customerId
	flightid	Booking	flightId
	retflightid	Booking	flightId
	retflight	Booking	flightId
	toflight	Booking	flightId
	toflightid	Booking	flightId
	retflightsegid	Flight	segmentId
	toflightsegid	Flight	segmentId
Flights	scheduledarrival time	Flight	scheduledArrival Time
	scheduleddeparture time	Flight	scheduledDeparture Time
	_id	Flight	id
	flightsegmentid	Flight	flightSegmentId
	toairport	FlightSegment	destPort
	fromairport	FlightSegment	originPort
	originport	FlightSegment	originPort
	destport	FlightSegment	destPort
	miles	FlightSegment	miles
Customer	_id	Customer	id
	user	Customer	id
	byid	Customer	id
Login	login	Customer	id
Web	user	Customer	id
	miles	FlightSegment	miles
	_id	FlightSegment	id
	destport	FlightSegment	destPort
	originport	FlightSegment	originPort

in Table II),
- mapping of the denormalized data model present in the logs to a normalized model present in Fig. 11.

The latter mapping is necessary because the information in the logs of individual services represents some part of the view of the overall schema of the application. Additionally, the information in logs overlaps between the logs of different services.

It is expected that logs contain more information than only related to data schemas, thus the resulting data relationship graph will be richer (having more nodes and edges) than the reference graph from Fig. 16. To provide a fair assessment of our method, we first present the results on a subgraph limited to the set of nodes from Fig. 16. Then we discuss separately

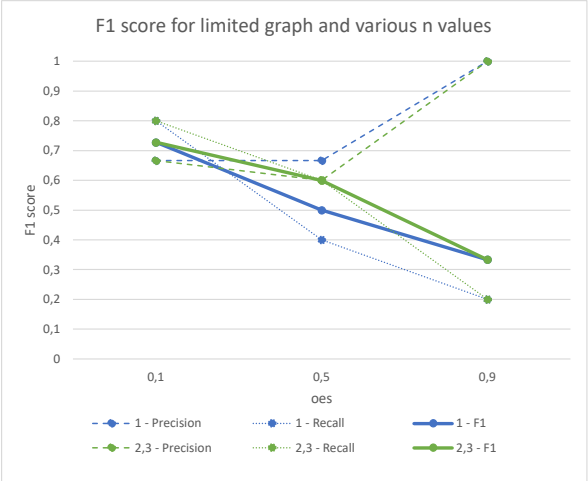

Fig. 17. The results of our method were measured by the F1 score. Colors denote the outcome for different values of n. Dashed lines represent precision, dotted – recall, and solid – F1 measure. The horizontal axis represents different values of the oes meta-parameter.

the rest of the graph.

Fig. 17 Shows the results of our method for the limited graph under different values of meta-parameters. It can be observed that for $n = 2$ and $n = 3$, the results are the same, and the n meta-parameter, in general, does not have a key influence on the overall score. A much more significant parameter is oes, which determines the perception of similarity between data attributes. The best F1 score of 0,72 was reached for the oes value of 0.1. We interpret such a low value as our method needing an argument to exclude a relationship (low level of similarity). This follows the intuition that, even though two attributes are closely corresponding to one another, they do not have to share the same values across the whole log. It is more probable that the set of shared values would be rather small because of a different set of services being triggered depending on the processes executed in the system.

An example of a graph generated by our method is shown in Fig. 18. The thick edges denote data relationships that match the reference data relationship graph from Fig. 16. We can see that the generated graph is much richer in information and our method gives a couple of interesting insights into the underlying data schemas. Firstly, it shows the potential of detecting unknown relationships based on data. An example is a relationship between *CustomerSession/customerId* and *Booking/customerId* which was detected based on matching values that both attributes take. An interesting part of the graph is a complete subgraph under Web service, whose nodes represent airport names. The fact that these attributes were extracted and are connected means that these values must frequently occur next to one another in logs. It is the case since when a flight is booked, origin and destination airports are logged in a single log entry. The fact that these attributes form a complete subgraph is the effect of the low cardinality of the set of airports. Combined with the automated

MOST WIEDZY Downloaded from mostwiedzy.pl

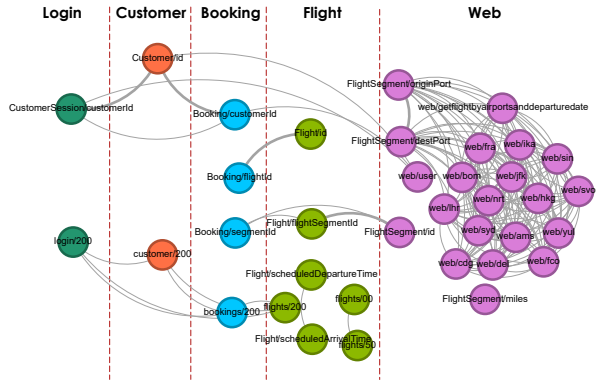

Fig. 18. A data relationship graph was generated for n=3, ies=0.9, and oes=0.1. Vertical lanes denote services depicted in Fig. 12. Nodes within a lane represent data attributes discovered in the log of a given service. Thick edges represent data relationships that match the reference relationship graph in Fig. 16.

TABLE III
CHARACTERISTICS OF THE NDEASYS2 DATASET

n	Discovered data attribute (n-gram embedding)
2	booking, byuser
2	flightsegment, _id
3	customer, by, user
3	flights, by, user

generation of the AcmeAir log sample, this resulted in all the origin/destination airport combinations being exhausted. It shows that our method is susceptible to the identification of false relationships for data of low cardinality. This is normally addressed in our method by choosing only long enough tokens (ml meta-parameter of our method). For the AcmeAir dataset, however, the minimum token length was set to 3 to utilize the airport data as a means of detecting other relationships.

In Fig. 18 it can be observed that the graph contains nodes representing data attributes that are not present in the original model in Fig. 11, e.g. *web/user* or *web/getflightsbyairportanddeparturedate*. These attributes come from the analysis of log entries that do not directly log the whole data entity but are representants of typical observation-point logging used by developers [10]. Such attributes are valuable since they provide an additional semantic context to interpret other attributes. In the case of *web/user*, we can interpret that a customer is a user of the system.

The graph presented in Fig. 18 is created after performing the attribute mapping described at the beginning of the section. The full set of results achieved by our method for various meta-parameters is available in [9] in the GEXF format. However, it is interesting to analyze raw data attributes. Table III presents a sample extract of raw attributes represented by n-grams for $n \in \{2, 3\}$. The set of tokens constituting the n-gram can also be viewed as a semantic context, which can be used for the refinement of the generated graph in the future.

Our method has also identified some false positives – numeric nodes that represent an accidental correlation of data that should be treated as noise. Filtering out this noise is one of the challenges for our future work.

F. Real-life System Results

Our method has been executed against a log of the real-life system running at Nordea Bank - NDEASYS. We used the following meta-parameters to get the best results:

- $N = 4$,
- $ies = 0.9$,
- $oes = 0.1$,
- $mt = 10$,
- $ml = 5$.

Since, due to the size of the domain, it is hard to obtain the full ground truth, we focused on identifying falsely discovered data relationships. The edges of the resulting graph were validated using our domain knowledge. Our method achieved a precision of 0.98 on NDEASET2.

We performed an additional validation of the achieved results based on the analysis of data flows within NDEASYS. Based on Fig. 14, it is expected that data relationships are found between applications B, G, A, and F, which represent the main data flow. The discovered data relationships are aligned with the diagram in Fig. 14. For every edge of the architecture graph, there is at least one discovered data attribute relationship between the two applications.

Also, other properties of our method revealed on the AcmeAir dataset were confirmed:

- larger values of N tend to introduce an additional semantic context to the data relationship graph,
- observation-point logging entries introduce a domain nomenclature that puts other discovered attributes into a concrete context.

While promising, these results still need to be validated especially in terms of the recall and F1 measures. The size of the data and the big number of discovered relationships do not allow us to present an anonymized visualization of the retrieved graph, analogous to Fig. 18.

G. Threats to Validity

Internal threats to validity include the construction of the NDEASET2 dataset and the level of detail of logs. We tried to mitigate both threats by choosing applications that provided a decent level of diversity. We find team diversity especially important, as it ensures log entries are created by developers from different teams, potentially following different standards. Also, the application diversity (different types of applications included in the dataset) is meant to increase the overall representativeness of the dataset. As for the quality of logs, we will cover its influence on our method's results in our future work.

The biggest external threat to validity is the dataset from Nordea Bank, which might not be representative of the whole bank and enterprise systems in general. This threat might come from the specific dataset that was chosen for research but also from potential formal or informal rules regarding logging practices enforced by Nordea. We tried to mitigate this threat

MOST WIEDZY Downloaded from mostwiedzy.pl

by applying our method to a benchmark system, which is however much simpler than an average enterprise application, both business and technical-wise. In the future, we will try to find other datasets to further mitigate these threats. It is worth noting that although publicly available datasets are very beneficial for the validation of our method and the general reproducibility of research, they will never be close to the size and complexity of real-life enterprise systems. Therefore, apart from selecting open-source datasets, we will seek further datasets inside Nordea Bank to retain real-world validation for our research.

The threat to construct validity is an insufficient measure of the quality of our method for the NDEASET2 dataset (we are using only precision, not using recall, and not being able to construct the reference graph). Construction of the reference graph for NDEASYS is one of the goals of our future work.

Conclusion validity is threatened by data attributes with low cardinality of values (e.g. currency symbols in the banking industry or airport codes in the flight booking domain). Such data attributes can be falsely considered as related just because of a high chance of them getting the same values. Another threat to conclusion validity is the low level of detail of logs can result in relationships between data attributes not being detected.

V. Related Work

Discovering relationships between data is a topic studied by schema matching discipline [11]. This domain is very broad and uses several techniques, including matching strings, matching words in certain languages, matching graphs, or using ontologies representing knowledge in certain fields. The goal of traditional schema matching is to match elements between two schemas given as input. The authors of [12] and [13] study a more general case where the number of schemas to be matched is greater than two. Finding similarities in attribute values (called duplicates) is the basis of the method proposed in [14]. The authors perform schema matching based on a small number of matching values in two schemas. [15] proposes a method for matching knowledge graphs. The authors split the process into schema-level and instance-level matching. In the first phase, matching is performed using only schema information, while in the second phase, the result is further refined by matching the values that schema attributes take. All of the approaches in the schema matching domain assume full knowledge of the individual schemas being matched. What is in the area of interest is only finding the correspondence between the attributes of the schemas. This is a significant difference compared to our method, which needs to discover both the schemas and their corresponding attributes.

Semantic data type discovery is a field of research that focuses on assigning types that have well-defined meanings to schema attributes. As compared to regular type detection (e.g. whether an attribute is a string, int, or boolean), semantic types hold much more information (e.g. postal code, surname, country) and as such could be used to match attributes of different schemas. [16] introduces *Sherlock*, a supervised-learning approach to semantic type detection. It uses the VisNet dataset to train a classifier that detects one of the 78 semantic types defined in the *T2Dv2Gold Standard* dataset. The authors of [17] propose the *SATO* algorithm, which extends *Sherlock* by incorporating the concept of context. Data attributes are matched not only based on the values they take but also based on the neighbor attributes in the same schema. Such an approach allows to properly classify semantic types for attributes with a low number of samples. Both [17] and [18] rely on model training for a given dataset which contrasts with the unsupervised approach we take to derive data relationships. [18] describes *RaF-STD*, an unsupervised learning approach to semantic type detection. The authors exploit triples of schema attributes that share common values and iteratively introduce higher-level virtual attributes representing the notion of similarity. This method does not require any prior knowledge of the source schemas but requires the existence of such schemas, which we do not assume in our method.

Automated log analysis is a field of research that focuses on the extraction of data from logs in an automated fashion. The authors of [2] split this field based on the type of knowledge being extracted. According to this classification, domain model extraction is a field that is somewhat relevant to data attribute discovery. [19] presents a method for discovering an ontology based on event logs and process mining techniques. The method is validated using a dataset of stack overflow posts and proved to generate a valid ontology in a computer science domain. The use of an event log requires intensive log preprocessing, which is not the case with our method, which operates on raw application logs, with very little preprocessing to unify log format across applications.

Log template generation is a field of research that aims in finding patterns in lines of log files as part of a log analysis task. Typically such patterns split a log line into static and variable parts, which could be used to discover data attributes in our method. [20] and [21] are two common methods for discovering log patterns. According to [1], both of these methods do not cope well with log lines of variable length (e.g. XML document being logged), which can be a typical case of logging entry/exit data. The authors of [10] classify such log entries under the observation-point logging category and show it is one of the most common types of log entries. [1] proposes an *SLT* method for log template generation, which is well-suited for handling log entries of variable length but the result is coarse-grained – it does not provide any information on the position of the variable parts within a log line. Our method for data relationship discovery uses the context of neighbor tokens to detect data attributes, so the position of the variable tokens within a log entry is essential.

Word embedding is a sub-field of natural language processing that aims in representing words in some text corpus as multi-dimensional vectors. This corresponds to the initial phases of our method, where we perform the embedding of tokens of a log line. [22] and [23] are the two most popular methods for word embedding. For both, there is a large set

MOST WIEDZY Downloaded from mostwiedzy.pl

of pre-trained models over text corpora in various languages. However, their usability to log analysis is limited. Firstly, they require training of the model on a particular text corpus, as the model is specific to the logs being analyzed. Secondly, the set of words in log corpora is infinite due to information like unique identifiers being commonly logged.

The method for discovering interactions between applications described in [1] shares similar ideas to the method described in this paper. It looks for rare tokens in various applications' logs and uses the findings to justify the hypothesis of the existence of a relationship between applications. The main difference between [1] and this paper is that [1] is focused on detecting the relationships between applications, while this paper focuses on detecting relationships between data attributes. For this purpose, we are looking not only at the rare but also the more frequent tokens. Discovery of a relationship of more frequent tokens (e.g. currency codes) in two applications cannot be treated as a good justification for the existence of a dependency (data exchange) between the applications. Therefore the method presented in this paper cannot be considered a generalization of the method described in [1]. These methods are rather complementary and the identification of relationships in data can be used to refine the set of interactions discovered by [1].

VI. Conclusions and Future Work

In this paper, we proposed an unsupervised method for the automated discovery of data relationships in an enterprise system. We validated our method on a synthetic and real-world dataset. In both cases, it proved to be a useful tool for obtaining an overview of the data processed by applications within an enterprise system. For the real-world dataset, the method has successfully detected correspondences between data attributes in different applications, which can have multiple uses - failure diagnosis, schema discovery, or schema mapping, to name a few.

In the future, we will extend the method to take advantage of additional information – locality of log entries, data attributes described by n-grams with higher values of n, and semantics hidden in the observation-point logging entries. We will further analyze the retrieved relationship graphs for the NDEASET2 dataset and work on establishing the ground truth for NDEASYS to validate the recall and F1 measures. We will also combine the proposed method for data relationship discovery with the method of interaction discovery [1] to form a comprehensive approach for discovering knowledge about an enterprise system based on automated log analysis.

Acknowledgment

This paper was written in cooperation with the Nordea Bank which provided the log dataset and an overview of the systems that were subject to this study.

References

[1] L. Korzeniowski and K. Goczyla, "Discovering interactions between applications with log analysis," 2022. doi: 10.15439/2022F172 p. 861 – 869.

[2] L. Korzeniowski and K. Goczyla, "Landscape of automated log analysis: A systematic literature review and mapping study," *IEEE Access*, vol. 10, pp. 21 892–21 913, 2022. doi: 10.1109/ACCESS.2022.3152549

[3] D. Barua, N. T. Rumpa, S. Hossen, and M. M. Ali, "Ontology based log analysis of web servers using process mining techniques," 2019, Conference paper. doi: 10.1109/ICECE.2018.8636791 p. 341 – 344.

[4] S.-L. Chuang and L.-F. Chien, "Enriching web taxonomies through subject categorization of query terms from search engine logs," *Decision Support Systems*, vol. 35, no. 1, pp. 113–127, 2003. doi: https://doi.org/10.1016/S0167-9236(02)00099-4 Web Retrieval and Mining.

[5] S. Khan and S. Parkinson, "Eliciting and utilising knowledge for security event log analysis: An association rule mining and automated planning approach," *Expert Systems with Applications*, vol. 113, p. 116 – 127, 2018. doi: 10.1016/j.eswa.2018.07.006

[6] Acmeair, "A java implementation of the acme air sample application." last accessed: 2023-07-24. [Online]. Available: https://github.com/acmeair/acmeair

[7] C. D. Manning, H. Schütze, and G. Weikurn, "Foundations of statistical natural language processing," *SIGMOD Record*, vol. 31, no. 3, p. 37 – 38, 2002. doi: 10.1145/601858.601867

[8] C. M. Aderaldo, N. C. Mendonça, C. Pahl, and P. Jamshidi, "Benchmark requirements for microservices architecture research," 2017. doi: 10.1109/ECASE.2017.4 p. 8 – 13.

[9] Acmeair, "A nodejs implementation of the acme air sample application with extended logging." last accessed: 2023-07-24, commitId: 59e8545c1e5264107e60706a360e0c8133aa8f9e. [Online]. Available: https://github.com/lkorzeni11/acmeair-nodejs

[10] Q. Fu, J. Zhu, W. Hu, J.-G. Lou, R. Ding, Q. Lin, D. Zhang, and T. Xie, "Where do developers log? an empirical study on logging practices in industry," 2014, Conference paper. doi: 10.1145/2591062.2591175 p. 24 – 33.

[11] P. Shvaiko and J. Euzenat, "A survey of schema-based matching approaches," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3730 LNCS, p. 146 – 171, 2005. doi: 10.1007/11603412_5

[12] E. Rahm and E. Peukert, "Large-scale schema matching," in *Encyclopedia of Big Data Technologies*, 1st ed., S. Sakr and A. Zomaya, Eds. Springer Publishing Company, Incorporated, 2019. ISBN 331977526X

[13] E. Rahm and E. Peukert, "Holistic schema matching," in *Encyclopedia of Big Data Technologies*, 1st ed., S. Sakr and A. Zomaya, Eds. Springer Publishing Company, Incorporated, 2019. ISBN 331977526X

[14] A. Bilke and F. Naumann, "Schema matching using duplicates," 2005. doi: 10.1109/ICDE.2005.126 p. 69 – 80.

[15] X. Xue and H. Zhu, "Matching knowledge graphs with compact niching evolutionary algorithm," *Expert Systems with Applications*, vol. 203, 2022. doi: 10.1016/j.eswa.2022.117371

[16] M. Hulsebos, A. Satyanarayan, K. Hu, T. Kraska, M. Bakker, Demiralp, E. Zgraggen, and C. Hidalgo, "Sherlock: A deep learning approach to semantic data type detection," 2019, Conference paper. doi: 10.1145/3292500.3330993 p. 1500 – 1508.

[17] D. Zhang, Y. Suhara, J. Li, M. Hulsebos, Demiralp, and W. C. Tan, "Sato: Contextual semantic type detection in tables," *Proceedings of the VLDB Endowment*, vol. 13, no. 11, p. 1835 – 1848, 2020. doi: 10.14778/3407790.3407793

[18] F. Piai, P. Atzeni, P. Merialdo, and D. Srivastava, "Fine-grained semantic type discovery for heterogeneous sources using clustering," *VLDB Journal*, vol. 32, no. 2, p. 305 – 324, 2023. doi: 10.1007/s00778-022-00743-3

[19] D. Barua, N. T. Rumpa, S. Hossen, and M. M. Ali, "Ontology based log analysis of web servers using process mining techniques," 2019, Conference paper. doi: 10.1109/ICECE.2018.8636791. ISBN 978-153867482-6 p. 341 – 344.

[20] R. Vaarandi and M. Pihelgas, "Logcluster - a data clustering and pattern mining algorithm for event logs," 2015, Conference paper. doi: 10.1109/CNSM.2015.7367331 p. 1 – 7.

[21] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," 2017, Conference paper. doi: 10.1109/ICWS.2017.13 p. 33 – 40.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, Conference paper.

[23] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," 2014, Conference paper. doi: 10.3115/v1/d14-1162 p. 1532 – 1543.

MOST WIEDZY Downloaded from mostwiedzy.pl