2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom) | 978-0-7381-4380-4/20/$31.00 ©2020 IEEE | DOI: 10.1109/TRUSTCOM50675.2020.00261

Assessing the Similarity of Smart Contracts by Clustering their Interfaces

Monika di Angelo, Gernot Salzer
TU Wien, Vienna, Austria
{monika.di.angelo,gernot.salzer}@tuwien.ac.at

***Abstract*—Like most programs, smart contracts offer their functionality via entry points that constitute the interface. Interface standards, e.g. for tokens contracts, foster interoperability. Ethereum is the most prominent platform for smart contracts. The number of contract deployments approaches 30 million, corresponding to roughly 300 000 distinct contract codes. In view of these numbers, it is necessary to develop automated methods for classifying contracts regarding their purpose, if one aims at a qualitative and quantitative understanding of what blockchain applications are used for at large.**

We approach the task by considering contracts as similar if their interfaces are. We encode interfaces and their interrelationships as graphs and explore several algorithms regarding their ability to find clusters of functionally similar contracts. Our evaluation of the quality of clustering relies on a ground truth of token and wallet contracts identified in earlier work. Our analysis is based on the bytecodes deployed on the main chain of Ethereum up to block 10.5 million, mined on July 21, 2020.

***Index Terms*—blockchain analysis, clustering, EVM bytecode, semantics, code similarity, smart contract**

I. Introduction

Numerous ideas have been put forward regarding how blockchains in general and smart contracts in particular may enhance services or lead to entirely new ones. At the same time, it is difficult to assess to which extent these ideas have been realized and how widely they have been adopted. Websites, white papers, and blog posts are easily accessible but biased, as they are often intended to attract investors and customers to an idea or product.

A more reliable source is the data of public blockchains. The recorded transactions are a truthful account of the events and allow us, at least in principle, to draw conclusions about blockchain applications and their activities. This transaction data, though abundant, is in a low-level format that cannot be interpreted directly. Therefore, the field of blockchain analysis is developing methods and tools to make the raw blockchain data amenable to inspection, e.g. by aggregating data, relating events, or checking the properties of chain code.

This work aims to relate functionally similar smart contracts on a large scale, which helps to understand the smart contract landscape in terms of their purpose. We concentrate on Ethereum as the main platform for smart contracts, with the largest data set available.

There, smart contracts are deployed and stored as EVM bytecode, whose purpose is not readily accessible. Regarding source code in high-level programming languages, the website `etherscan.io` offers verified source code for roughly 100 000 contracts: Contract authors may submit source code and the service verifies that it corresponds to the deployed bytecode. While source code may be easy to understand on an individual basis, it is not feasible to check all verified contracts manually for their purpose and functionality. Moreover, the motivation for providing a verified source introduces a bias, and the available source code covers only a fraction of all deployments.

Approach. We choose an abstraction layer between byte- and source code: the interface of contracts, i.e. the set of entry points. The underlying assumption: If the same function header appears in multiple contracts, then the corresponding function bodies implement similar functionality. While this heuristic may fail in single cases, it becomes reliable when contracts share larger parts of the interface. We thus assume that contracts with many functions in common serve a similar purpose. This is also supported by the fact that groups of wallet contracts can be identified by interface fuzzing [1]. Moreover, interface standards (like those for various types of tokens) prescribe certain functions to foster interoperability.

Research questions. We attempt to structure the body of deployed contracts with regard to functional similarity in order to determine

- functionally discernible groups,
- the amount of contracts in these groups,
- how similar contracts are beyond clones, and
- what smart contracts are intended for.

To this aim, we address the following questions. Which methods are suited for clustering interfaces with respect to functional similarity? How well do they achieve the task? Which structures can be uncovered in the interrelationship of the interfaces?

Requirements. A good clustering method should place interfaces of contracts with the same purpose (i.e. similar interfaces modulo our assumption) into the same cluster. The clustering should yield sufficiently many groups, as we want to differentiate purposes. Interfaces of contracts already known to serve the same purpose should end up in the same cluster.

To evaluate our methods, we rely on the following criteria.

- Resolution: Having few, but large clusters tends to place interfaces into a cluster, where they do not belong (false positives), while having many small clusters distributes

2324-9013/20/$31.00 ©2020 IEEE
DOI 10.1109/TrustCom50675.2020.00261

interfaces with similar functionality to several groups (false negatives).
- Ground truth: Check the known contracts for reasonable grouping. In earlier work, we have identified groups of wallet and token contracts that should be clustered accordingly.
- Diversity: At least for wallet contracts, there should be hardly any other interfaces in the same cluster, as the wallets were selected manually and checked via interface fuzzing.
- Cohesiveness: In a cluster, each interface should be sufficiently similar to each other interface.

Contribution. With this work, we investigate approaches to cluster smart contracts functionally, based on the similarity of the interfaces restored from bytecode. Moreover, we add to the discussion on how smart contracts can be grouped into clusters of similar functionality. Furthermore, we structure the body of bytecode regarding the implemented functionality. Our findings may be relevant for people interested in the functional diversity of smart contracts at large.

Roadmap. We characterize interfaces in section II, and detail our methods in section IV. Results, interpretations, and discussions are presented for the different approaches separately, in section V for the bipartite graph, in sections VI, VII, and VIII for the projected graph with different similarity measures. We compare our approach to related work in section III and conclude in section IX.

II. Interfaces

Most smart contracts on Ethereum adhere to an interface standard for interaction (ABI), which we introduce in this section. Due to this standard, we are able to extract the interfaces of virtually all bytecodes deployed on the main chain. Moreover, the community established several standard interfaces [2] for interaction between contracts.

Function Signatures and ABI. The Abstract Binary Interface (ABI) standard [3] identifies functions by signatures that consist of the first four bytes of the Keccak-256 hash of the function name together with the parameter types. Thus, the bytecode of a contract contains instructions to compare the first four bytes of the call data to the signatures of its functions. The presence of a particular function in a contract can be checked by locating the corresponding 4-bytes hash in its deployed bytecode.

Interface. One way of determining the functionality of a contract (in part) is to compare its interface with known functions, interfaces, or even interface standards. We use a heuristic procedure from previous work [4] to locate all exposed function signatures of a contract, thereby reconstructing its ABI as a set of 4-bytes hashes.

Standard Interfaces. With the Ethereum Improvement Proposals (EIPs), also standards for contracts [2] are proposed, discussed, and eventually established. Especially for token interfaces, there exist four accepted standards (ERC-20, ERC.721, ERC-777, ERC-1155), and several proposed ones.

Interfaces as Graphs. For our clustering of similar contracts, we regard contracts as equivalent if their interfaces are the same. Our analysis starts from the interfaces and the function signatures they consist of. The interfaces become the nodes of a graph, whereas the signatures either become a second type of nodes or give rise to weights on the edges between the interface nodes.

III. Related Work

A. Graph and Clustering Approaches

In a graph analysis, the authors of [5] study Ether transfer, contract creation, and contract calls. They compute metrics like degree distribution, clustering, degree correlation, node importance, assortativity, and strongly/weakly connected components, based on which they list 35 major observations. Beside several power law observations, they state that exchanges are important regarding money flow, while token contracts are responsible for a high transaction volume.

The authors of [6] find groups of similar contracts employing the unsupervised clustering techniques *affinity propagation* and *k-medoids*. Using the program *ssdeep*, they compute fuzzy hashes of the bytecodes of a set of verified contracts and determine their similarity by taking the mean of the Levenshtein, Jaccard, and Sorenson distance. After clustering, the authors identify the purpose of a cluster by the associated names (and a high transaction volume). With k-medoids, they are able to identify token presale, DAO withdrawal, some gambling, and empty contracts.

The authors of [7] construct four interaction graphs based on Ethereum transactions, one graph each for user-to-user, user-to-contract, contract-to-user, and contract-to-contract. Then they invesetigate local and global graph properties and discuss similarities with social networks.

Difference to our approach. On the one hand, the approaches [5], [7] that apply similar methods (like graph algorithms, properties, and metrics as well as clustering techniques) focus on different aspects. On the other hand, the appoach with a related focus [6] employs different methods.

We investigate the semantic similarity of the bytecode of smart contracts in order to uncover the purpose. The mentioned graph approaches do not focus on semantic groups of smart contracts like token or wallet contracts. Neither do they analyze the similarity of contracts, especially not on a semantic level. The clustering approaches are based on semantic-free representations of bytecode like hashes [6] or do not aim at finding semantic similarities within sets of contracts.

B. Code Analysis

Code clones. To detect code clones, the authors of [8] first deduplicate contracts by "removing function unrelated code (e.g., creation code and Swarm code), and tokenizing the code to keep opcodes only". Then they generate fingerprints of the deduplicated contracts by a customized version of fuzzy hashing and compute pair-wise similarity scores.

In their tool for clone detection, the authors of [9] characterize each smart contract by a set of critical high-level semantic

properties. A two-step approach of symbolic transaction sketch and syntactic feature extraction yields a numeric vector as representative of a contract and its semantic properties, which are based on the control flow graph, path conditions as well as storage and call operations. Then they detect clones by computing the statistical similarity between the respective vectors.

The authors of [10] investigate clones based on Solidity source code applying a tree-based clone detector. They find that "9 out of the top-10 largest clone clusters are token managers" and that the most frequently cloned code is "token contracts that comply with the ERC20 standard".

Interfaces. To detect token systems automatically, the authors of [11] compare the effectiveness of a behavior-based method combining symbolic execution and taint analysis, to a signature-based approach limited to ERC20-compliant tokens.

Similarly, the authors of [4] employ a signature-based approach to identify ERC-compliant and non-compliant tokens. Their approach to detect similarities in bytecode is comparable to the first step by [8]. Instead of fuzzy hashing as a second step though, the latter authors restore the interface as a set of function signatures extracted from the bytecode in order to identify token contracts reliably.

Difference to our approach. We investigate the semantic similarity of the bytecode of smart contracts in order to uncover the purpose. The mentioned approaches for code analysis obviously intend to find similarities, be it in form of near identity in the case of clones, or by means of interface standard compliance. While [10] detects code clones on Solidity source code level (without focus on semantics), the other approaches analyze the EVM bytecode. Regarding semantics, clone detection is achieved by [9]. However, they do not tackle similarity beyond clones. When determining the standard compliance of EVM bytecode, the semantic similarity of contracts is considered only implicitly via the standard. Similarity beyond clones or standard compliance is tackled by [4] for non-compliant tokens or for distinguishing the possible types of tokens, partially on a semantic level. We complement the latter study by a clustering approach.

IV. Methods for Clustering

In this section, we first describe the methods for the bipartite interface-signature graph. Then, we discuss three ways of measuring the similarity of interfaces, to be used as weights in the projected graph of interfaces. Furthermore, we detail the visualization of groups of interfaces with similar functionality. Finally, we present the methods to evaluate the quality of the clustering.

A. Communities in the Unweighted Interface-Signature Graph

One way to represent the interfaces is as a bipartite graph, with the interfaces and signatures as the nodes and unweighted edges indicating the membership of the signatures in the interfaces. This is a lossless representation: Given the graph, we can reconstruct the interfaces. Another advantage is that the bipartite graph and the original problem are of the same size. To get a first impression of the data set and of the applicability of our approach we decompose the graph into its weakly connected components (WCC) and apply the Louvain algorithm [12] to find communities.

Louvain is a two-step algorithm for detecting hierarchies of communities in a network. The algorithm repeatedly applies two steps: step 1 greedily maximizes the (local) modularity, while step 2 aggregates the found communities. It stops, when a predefined limit for the rounds is reached or the modularity can no longer be increased. In step 1, nodes are randomly placed in communities and subsequently moved if the gained increase in modularity is above a threshold. In step 2, nodes of a community are merged to a single next-level node with self-loops representing the edges within the community, and correspondingly weighted edges to the other merged nodes.

B. Similarity-weighted Projected Graph of Interfaces

We consider the similarity measures *Jaccard* [13] and *Overlap* [14] for each pair of interfaces. These measures relate the number of shared signatures to the additional signatures in the compared interfaces. Additionally, we use *Card*, the absolute number of shared signatures.

Let I and J be interfaces, viewed as sets of signatures. The three measures are defined as $\mathit{Jaccard}(I,J) = |I \cap J|/|I \cup J|$, $\mathit{Overlap}(I,J) = |I \cap J|/\min(|I|,|J|)$ and $\mathit{Card}(I,J) = |I \cap J|$. The first two similarities are rational numbers in $[0,1]$, whereas *Card* similarities are non-negative integers. By definition, we have $\mathit{Jaccard}(I,J) \leq \mathit{Overlap}(I,J) \leq \mathit{Card}(I,J)$. In all three cases, the minimal value of 0 means that the interfaces share no signature. The maximal *Jaccard* similarity of 1 is only attained if the two interfaces are identical, while the maximal *Overlap* of 1 means that one interface is fully contained in the other, $I \subseteq J$ or $J \subseteq I$. *Card* favors contracts with numerous function headers, while *Overlap* treats interfaces as similar if one is contained in the other, no matter how many additional signatures are involved. Therefore, we expect *Jaccard* to be suited best for our purpose.

Given a similarity measure, we can represent our problem as a projected graph with the interfaces as nodes and weighted edges expressing the similarity. As most interfaces share some signature, the number of edges is quadratic in the number of nodes, which amounts to roughly a billion edges in our case. With standard computational resources, such a graph is too big for any non-linear algorithm to finish within reasonable time bounds. This can be solved by retaining only edges whose weight exceed a fixed cutoff value. While necessary for practical reasons, this step may distort the original problem. However, using cutoffs seems reasonable when aiming at clusters where any two interfaces are supposed to be sufficiently similar to each other. By lowering the cutoff, we allow increasingly less similar interfaces into a component, which in return yields fewer, but larger components.

For *Overlap* similarity, we consider the special case of cutoff 1.0. This leaves us with a graph representing the hierarchy of interfaces induced by the subset relation. To be precise, we consider the transitive reduction of this graph,

where there is a directed edge between interfaces I and J if I is a subset of J and there is no other interface in between.

C. Visualizing Clusters in the Similarity Graphs

To depict the clusters resulting from various cutoffs, we use the graph visualization tool *gephi*. In *gephi*, we apply ForceAtlas2 [15], a force-directed 2D layout algorithm to spatialize a graph based on attraction over the edges (with similarity values as weights) and repulsion between the nodes (interfaces). The parameters change the simulation of the forces that apply continuously until stopped. The placing of a node depends on the other nodes, and especially on their connections. "The result varies depending on the initial state. The process can get stuck in a local minimum. It is not deterministic, and the coordinates of each point do not reflect any specific variable. The result cannot be read as a Cartesian projection. The position of a node cannot be interpreted on its own, it has to be compared to the others." [15] We used default parameter settings, except for scaling and overlap prevention. In some cases, we also use the multi-gravity version of ForceAtlas2 where each node can have its own center of gravity.

D. Evaluation

For evaluating the quality of the clustering, we use metrics and labeled interfaces.

Labels. After extracting the interfaces from the bytecode as described in [4], we check for compliance with an ERC-standard [4] or a wallet blueprint [1]. This yields three labels on level 1: *ercToken* indicates an ERC-compliant interface, *Token* a partially compliant token, and *Wallet* a wallet interface. Labels on level 2 subdivide the level 1 categories by 7 labels for tokens and 29 for wallets, listed in table III.

TABLE I
DATASET FOR INTERFACES

category	# interfaces	# bytecodes	# contracts
total	85 573	298 025	28 110 974
w/o interface	–	31 233	18 468 311
with interface	85 573	266 792	9 642 663
unlabeled	60 114	139 188	4 660 369
labeled	25 459	127 604	4 982 294
ERC-compliant	21 170	114 654	221 232
partially compliant	4 060	11 252	19 488
Wallet	229	1 698	4 741 574

Table I gives an overview of the dataset. Until July 21, 2020, about 28 million successful contract creations took place resulting in almost 300 000 unique bytecodes. Two thirds of the contracts resulted in bytecode without entry points. The other third is responsible for the diversity of interfaces: 266 792 bytecodes offer 85 573 different interfaces. We label 25 459 interfaces (corresponding to 5 million deployments) as being related to tokens or wallets. The majority of deployments are wallets, but they account only for 229 distinct interfaces, with an average of 20 706 deployments per interface. As for compliant and non-compliant tokens, the 240 720 labeled contracts correspond to 25 231 unique interfaces, with an average of 9.5 deployments per interface.

Metrics. For the assessment and comparison of the clustering, we consider the resolution in terms of number and size of the clusters, and we assess the diversity within a cluster using the percentage of labeled interfaces and the number of different labels.

In the connected components for a certain cutoff, each interface is sufficiently similar to *some* other interface in the component, while there may be pairs of interfaces not sharing any signature at all. In a good cluster, we expect every interface to be sufficiently similar to all the others, not just to a few. Therefore we evaluate the *cohesiveness* of a cluster by determining the minimal Jaccard similarity between any two interfaces.

V. CLUSTERS IN THE BIPARTITE GRAPH

In this section, we apply community algorithms to the bipartite graph of interfaces and signatures. Moreover, we evaluate the communities based on our labels.

A. Communities of Interfaces

The bipartite graph described in section IV contains 85 573 interfaces and 312 158 signatures, in total 397 731 nodes and 1 490 110 edges between interfaces and signatures. WCC decomposes the graph into 3 549 components, with the largest one containing 81 253 (95 %) of the 85 573 interfaces, all the labeled ones among them. The second largest component consists of 5 interfaces only. We conclude that the achieved clustering of the interfaces is too coarse with respect to the intended semantic distinction.

TABLE II
INTERFACES AND LABELS IN LOUVAIN COMMUNITIES

community	interfaces	labeled	different labels
1	21 720	18 615	7
2	5 581	401	5
3	3 255	1 927	4
4	2 046	25	2
5	2 042	116	6
6	2 037	26	4
7	2 015	152	2
8	1 923	789	2
9	1 646	115	3
10	1 512	60	2
11	1 382	42	2
12	1 317	246	7
13	1 101	102	3
14	1 070	16	2

For finding groups of interfaces with similar functionality, we therefore apply the probabilistic community algorithm Louvain [12] with a default granularity of 1.0. We obtain about 4 000 communities, roughly 450 more than WCC. Louvain usually does not place disconnected nodes in the same community, but may subdivide connected components. For the data in this study, only the largest WCC component is split. Table II lists, for each of the largest Louvain communities,

TABLE III
DISTRIBUTION OF LABELS OVER COMMUNITIES

Label	# interfaces	# WCC	# Louvain
ERC-20	19 787	1	106
Token (non-compliant)	4 060	1	92
ERC-721	1 259	1	14
ERC-777	65	1	3
ERC-1155	50	1	3
ERC-1644	6	1	2
ERC-1462	3	1	1
multisig Gnosis	99	1	2
multisig Ethereum	23	1	1
multisig BitGo	22	1	3
multisig BitGo forwarder	17	1	1
multisig Lundkvist	13	1	2
smart GnosisSafe	8	1	2
consumer wallet	7	1	1
timelocked wallet	6	1	2
multisig Teambrella	4	1	1
dapper	4	1	1
smartwallet	3	1	1
eidoo wallet	2	1	1
simple wallet	2	1	1
spendable wallet	2	1	1
intermediatewallet	2	1	2
controlled	2	1	1
ether wallet 2	1	1	1
ether wallet 1	1	1	1
multisig Unch. Capital	1	1	1
basicwallet	1	1	1
logicproxywallet	1	1	1
simple wallet 2	1	1	1
simple wallet 3	1	1	1
smart Argent	1	1	1
autowallet	1	1	1
wallet1	1	1	1
multisig Ivt	1	1	1
multisig Argent	1	1	1
multisig NiftyWallet	1	1	1
total number	25 459	3 549	$\sim$ 4 000

the number of comprised interfaces, labeled interfaces and different labels.

Louvain clustering shows a more balanced distribution of interfaces over the communities. The largest community comprises still 22 000 interfaces, but there are 13 additional communities with more than 1 000 interfaces. Moreover, every community contains at most 7 different labels.

Table III shows the distribution of labels over communities. For each label, it gives the number of interfaces tagged with it and the number of communities it appears in. The upper part of the table shows the token labels, the lower part the wallet labels. With the deterministic WCC decomposition, all labeled interfaces end up in the largest component, while the other 3 548 components do not contain any labeled interfaces. Regarding the probabilistic Louvain algorithm at a default resolution of 1.0, the labels occur in more than 100 communities. For the wallets, the default resolution nicely distributes each label to its own community. For the tokens, it is not evident, which resolution would be preferable.

VI. PROJECTED GRAPH WITH JACCARD SIMILARITY

In this section, we construct the projected graph using *Jaccard* similarity for the edge weights. Then, we cluster and visualize the interfaces.

A. Jaccard Clusters – All Labels

First, we take the labels from section IV and look at their distribution over clusters. Then, we color the interfaces in the similarity graph based on the labels. As a metric, we calculate the proportions of labels and unlabeled interfaces within the clusters.

TABLE IV
THE 20 LARGEST COMPONENTS WITH JACCARD CUTOFF 0.9

component	interfaces	labeled	labels	% labeled
1	2 627	2 620	2	99.7
2	852	852	2	100
3	359	359	1	100
4	221	221	2	100
5	208	208	1	100
6	110	110	1	100
7	67	67	2	100
8	62	0	0	0
9	60	0	0	0
10	54	54	1	100
11	51	50	1	98
12	48	**34**	1	70.8
13	47	47	1	100
14	44	44	2	100
15	42	42	1	100
16	41	41	2	100
17	41	41	2	100
18	38	38	1	100
19	38	38	1	100
20	37	37	1	100
singletons	61 825	12 918	34	20.9
non-singletons	23 748	12 541	16	52.8
all components	85 573	25 459	36	29.7

TABLE V
THE 20 LARGEST COMPONENTS WITH JACCARD CUTOFF 0.8

component	interfaces	labeled	labels	% labeled
1	9 274	9 181	2	99
2	412	412	2	100
3	248	248	1	100
4	216	216	1	100
5	208	203	2	97.6
6	131	0	0	0
7	108	0	0	0
8	104	0	0	0
9	101	101	1	100
10	89	**63**	1	70.8
11	82	0	0	0
12	80	0	0	0
13	65	65	2	100
14	57	57	1	100
15	50	50	1	100
16	49	48	2	98
17	48	0	0	0
18	47	47	1	100
19	47	0	0	0
20	44	0	0	0
singletons	45 824	7 003	28	15.3
non-singletons	39 749	18 456	24	46.4

1) Cutoff 0.9: Table IV and the upper plot in figure 1 show that the clustered components (except for component 12) have either most interfaces labeled (the number of interfaces is almost equal to the number of labeled interfaces) or none, while the number of different labels is at most 2. Component 12 has 34 of 48 interfaces labeled as wallets (multisig Gnosis). This indicates that the clustering is reasonable.

Regarding the largest component in both plots of figure 1, we notice an inhomogeneous connection pattern, as several dense areas with only little interconnection stand out.

TABLE VI
THE 20 LARGEST COMPONENTS WITH JACCARD CUTOFF 0.7

component	interfaces	labeled	labels	% labeled
1	14 028	13 860	2	98.9
2	654	654	2	100
3	343	0	0	0
4	260	252	2	96.9
5	217	0	0	0
6	176	0	0	0
7	155	0	0	0
8	140	0	0	0
9	131	0	0	0
10	122	0	0	0
11	106	**71**	1	67.0
12	71	0	0	0
13	67	67	1	100
14	59	**22**	1	37.3
15	57	0	0	0
16	55	0	0	0
17	54	54	1	100
18	54	0	0	0
19	51	0	0	0
20	49	49	2	100
singletons	36 881	4 180	24	11.3
non-singletons	48 692	21 279	27	43.7

TABLE VII
THE 20 LARGEST COMPONENTS WITH JACCARD CUTOFF 0.6

component	interfaces	labeled	labels	% labeled
1	19683	16893	5	85.8
2	889	881	2	99.1
3	316	305	2	96.5
4	208	0	0	0.0
5	196	0	0	0.0
6	187	0	0	0.0
7	142	0	0	0.0
8	133	**88**	2	66.2
9	129	0	0	0.0
10	104	0	0	0.0
11	93	0	0	0.0
12	88	0	0	0.0
13	77	77	1	100.0
14	73	**22**	1	30.1
15	73	0	0	0.0
16	72	0	0	0.0
17	71	0	0	0.0
18	71	0	0	0.0
19	69	0	0	0.0
20	67	67	2	100.0
singletons	29 191	2 450	20	8.4
non-singletons	56 382	23 009	29	40.8

2) Cutoff 0.8: When we include less similar interfaces by using a cutoff of 0.8, the components become larger, as expected. Still, we see in table V and the right plot of figure 1 that components have either most interfaces labeled or none. Component 10 comprises 63 wallets (again all multisig Gnosis) and 26 unlabeled interfaces. Also the number of different labels per component remains below 3. Obviously some of the earlier components have been merged as some of the previously separated interfaces are now connected, thus combining the affected components.

Regarding the largest component in the center of the right plot in figure 1, we notice several dense areas (over 20 green blobs and a few blue ones) with more pronounced interconnections than in the left plot.

3) Cutoff 0.7: Table VI lists the 20 largest clusters with cutoff 0.7 for the *Jaccard* similarity. We see that the clusters become even bigger, with more of them only partially labeled. More strikingly, the left plot of figure 2 depicts the largest component that shows many dense areas (plenty of green and several blue blobs) with strong interconnections in the center and plenty of loosely connected legs and annexes. Visually, sub-clusters are clearly discernible, but they end up in the same component because they are not sufficiently dissimilar.

4) Cutoff 0.6: The right plot in figure 2 depicts the largest cluster obtained with cutoff 0.6. By having more and more not so similar sub-clusters merged into one large component, the picture worsens with a decreased cutoff.

B. Jaccard Clusters – Token Labels

TABLE VIII
CLUSTERS WITH TOKEN LABELS FOR JACCARD CUTOFFS

label	total	0.6	0.7	0.8	0.9
erc20	19 787	2 563	4 318	6 881	11 503
erc721	1 259	297	440	610	892
erc777	65	10	16	34	50
erc1155	50	29	35	37	41
erc1644	6	5	5	5	5
erc1462	3	1	1	1	2
Token	4 060	1 008	1 396	1 997	3 085

Table VIII shows that the numerous interfaces labeled as token are distributed over many clusters. Even though the number of clusters with the most common label, erc20, decreases from 11 503 for cutoff 0.9, it is still at 2 563 for cutoff 0.6. With cutoff 0.6, more than half of the token labels are in the largest partition (cf. table VII), but also a loosely related wallet (multisig Gnosis, pink blob on the upper right border of the right plot in figure 2).

C. Jaccard Clusters – Wallet Labels

As we demonstrated before, clusters with wallet interfaces are the main candidates for partially labeled clusters. Figure 3 depicts all clusters containing wallet interfaces, except the largest one with two single wallets. The unlabeled interfaces appear in grey, while the 29 different wallet types are colored. We notice a distribution of wallet interfaces into well over

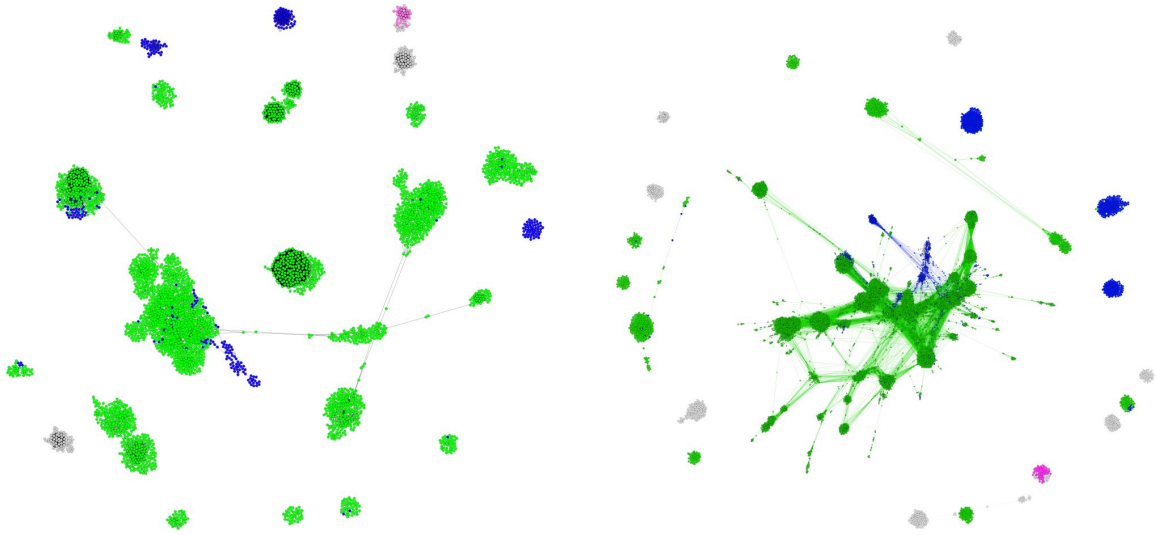

Fig. 1. The 20 largest clusters with Jaccard similarity cutoff 0.9 (left) and 0.8 (right). The color green depicts ERC-compliant tokens, blue non-compliant tokens, purple wallets, and grey unlabeled interfaces.

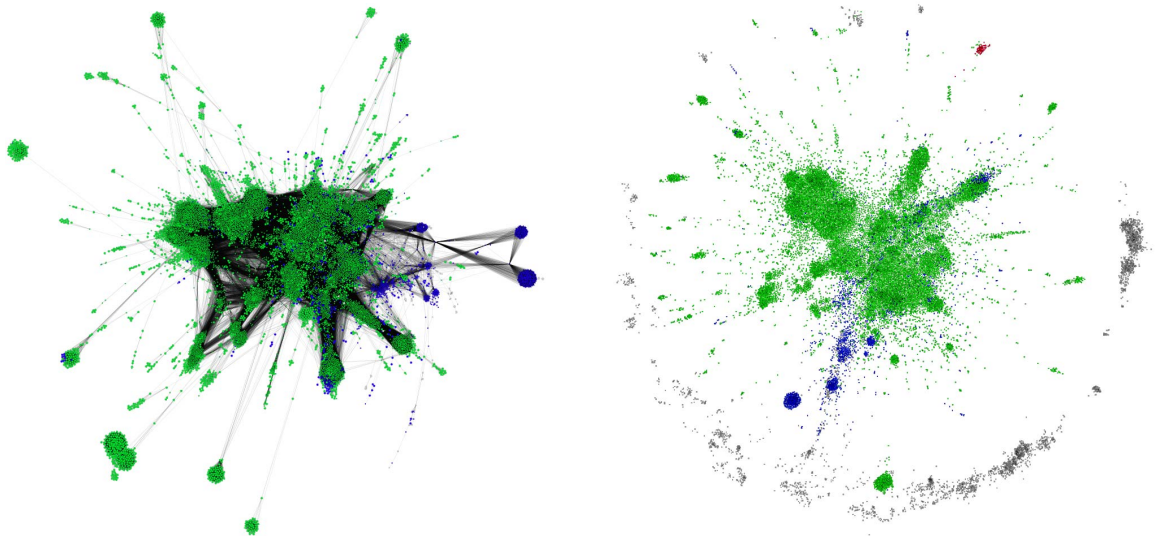

Fig. 2. The largest cluster with Jaccard similarity cutoff 0.7 (left) and 0.6 (right). The color green depicts ERC-compliant tokens, blue non-compliant tokens, pink wallets, and grey unlabeled interfaces.

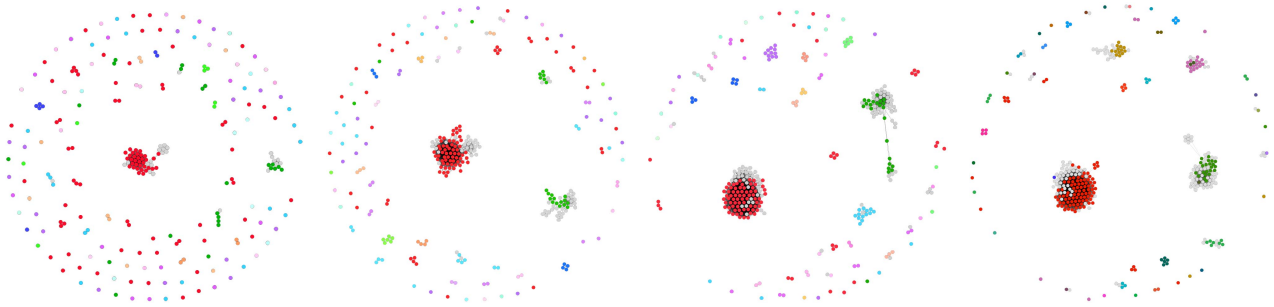

Fig. 3. Clusters with different types of wallets. The four plots show the effect of decreasing the Jaccard similarity cutoff from 0.9 (left) to 0.6 (right). Unlabeled interfaces are depicted in grey, the most variable interfaces pertaining to Gnosis multi-sig in red.

29 partitions, meaning that not all wallets of one type are clustered into the same partition. Moreover, we arrive at a more concise clustering with decreasing cutoff values (from left to right). Still, only the wallet types that have just a few different interfaces (and thus little variability) are clustered into a single partition, while the other wallet types are distributed over more than one cluster. Table IX lists the distribution of labels for wallet types into clusters with more than one interface.

TABLE IX
DISTRIBUTION OF WALLET LABELS PER JACCARD CUTOFF

label	total	0.6	0.7	0.8	0.9
multisig Gnosis	99	7	17	27	51
multisig Ethereum	23	2	2	5	12
multisig BitGo	22	5	8	13	18
multisig forwarder	17	4	6	15	17
multisig Lundkvist	13	4	8	9	13
smart GnosisSafe	8	3	3	3	5
consumer wallet	7	2	2	2	3
timelocked wallet	6	2	2	3	5
dapper	4	1	2	2	3
multisig Teambrella	4	1	1	1	2
smartwallet	3	2	2	2	3
intermediatewallet	2	2	2	2	2
controlled	2	2	2	2	2
simple wallet	2	1	1	1	2
spendable wallet	2	1	1	1	2
eidoo wallet	2	1	1	1	1

Also, the number of unlabeled interfaces in the wallet clusters increases with a lower cutoff value. However, we do not see any clusters containing two different wallet types. Thus, the different wallet types seem to be sufficiently dissimilar so that they do not get mixed up in the clusters. Also, token and wallet interfaces do not mingle, even though wallets handle tokens and sometimes contain function signatures that are named as the token counterparts they call (e.g. transfer(address, amount)).

D. Jaccard Clusters – Discussion

1) Distribution of Labels: All *Jaccard*-based clustering shows a multitude of partitions with only token-related labels (green and blue). Most larger clusters are predominantly tokens, but we also see a few partitions with only unlabeled interfaces. Most notably, partially labeled clusters (cf. table X) are rare for all examined cutoff values, with less than 0.2 % of the non-singleton clusters. For clusters with wallet labels, the percentage of partially labeled clusters is higher, up to 16 %. The distribution of labels from the ground truth indicates that using *Jaccard* similarity as the weights results in a reasonable clustering.

TABLE X
PARTIALLY LABELED CLUSTERS PER JACCARD CUTOFF

number of	0.6	0.7	0.8	0.9
partially labeled clusters	93	82	60	36
non-singleton clusters	56 382	48 692	39 749	23 748

2) Cutoff Values: A higher cutoff yields a stricter separation, which seems to be reflected in the label distribution. With a lower cutoff, clusters become larger and fewer, while the number of fully labeled clusters decreases. Likewise, the percentage of labeled interfaces decreases for both, singleton and non-singleton clusters. When decreasing the cutoff towards 0, the percentage of labeled interfaces among the singletons approaches 0, while the percentage of labeled interfaces in non-singletons approaches the overall value of 29.7.

In the upper plot of figure 1, we notice some loosely connected annexes and legs. These structures are even more pronounced for lower cutoffs (cf. figures 1 and 2). Interfaces at opposite ends are connected by a path with weights above the cutoff, but tend to have a similarity much lower than the cutoff (and therefore are not directly connected). This implies that with decreasing cutoff, we have an increasing number of unwanted members in the clusters.

It is not obvious, which cutoff values are suited best. For the tokens, it seems that 0.9 may be already too low (cf. the legs and annexes in figure 1), while a cutoff value of 0.6 yields better results for the wallets (cf. figure 3) since the single wallet types are distributed over fewer clusters. However, we can conclude that the fairly small group of 229 wallet interfaces is markedly more diverse regarding their interfaces than the much larger group of 25 231 token interfaces.

3) Minimal Jaccard similarity within a cluster: To judge the quality of a clustering, we consider the distribution of *minimal* similarities per cluster (figure 4). The higher the proportion of clusters with high cohesiveness, the better the clustering, since it indicates that no two interfaces in a cluster are too dissimilar. We see a decreasing quality of the clustering with decreasing cutoff values. For comparison, Louvain clustering in the original bipartite graph performs even worse.

VII. PROJECTED GRAPH WITH OVERLAP SIMILARITY

In this section, we construct, examine and visualize the projected graph using *Overlap* similarity for the edge weights.

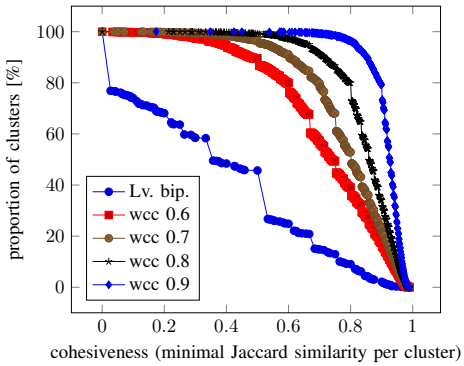

Fig. 4. Quality of clustering with Jaccard similarity cutoffs from 0.9 to 0.6. The higher the minimal Jaccard similarity for a cluster, the better the clustering. For cutoff 0.9, we see that virtually all clusters have cohesiveness 0.8, while the percentage of clusters with cohesiveness 0.8 drops to 80, 60, 40 and 10 % for cutoffs 0.8, 0.7, 0.6 and Louvain in the bipartite graph.

A. Reduced Subset Graph

We set the *Overlap* cutoff to 1.0, which coincides with the subset property (cf. section IV). The resulting graph thus reflects which interface is an extension of another one.

The directed subset graph has the 85 573 interfaces as nodes and over 2.5 million edges. Taking into account the transitivity of the subset relation, we consider the transitive reduction of the projected graph with just 151 380 edges. Running the WCC algorithm on the reduced subset graph yields one large partition with 71 950 interfaces, 1 581 non-singleton partitions with another 4 002 interfaces, and 9 621 interfaces as singletons.

Figure 5 depicts the largest partition of the reduced subset graph in the upper plot and the other non-singletons below. We notice pronounced interconnections in the large partition with 84.1% of the interfaces contain each other in some way.

B. Spatialized Subset Relationship

To explore the interconnections visually, we arrange the directed acyclic graph in the plane. We calculate two coordinates for each interface. On the horizontal axis, we order the interfaces regarding their maximal distance from a root node, i.e., we take the length of the longest chain of subset relations ending in a particular interface as its x-coordinate. An x-value of 0 indicates that the interface does not contain any other interface. On the vertical axis, we distribute the interfaces such that densely connected interfaces receive the same y-coordinate. To this end, we calculate Louvain communities, number them consecutively and use the index of the community as the y-coordinate of its interfaces. In *gephi*, we take the coordinates as gravity sources for the MultiGravity ForceAtlas2 layout algorithm.

Figure 6 shows the result of the visualization. The colors green and blue indicate ERC-compliant and non-compliant token interfaces, respectively, whereas wallet interfaces are marked in purple; unlabeled interfaces appear in grey.

In the upper plot, edges have the same color as their source node. We observe that the starting points of subset chains, located to the left, are predominantly unlabeled interfaces in grey. It is worth noting that 3 601 interfaces contain just one signature and another 3 611 interfaces just two signatures,

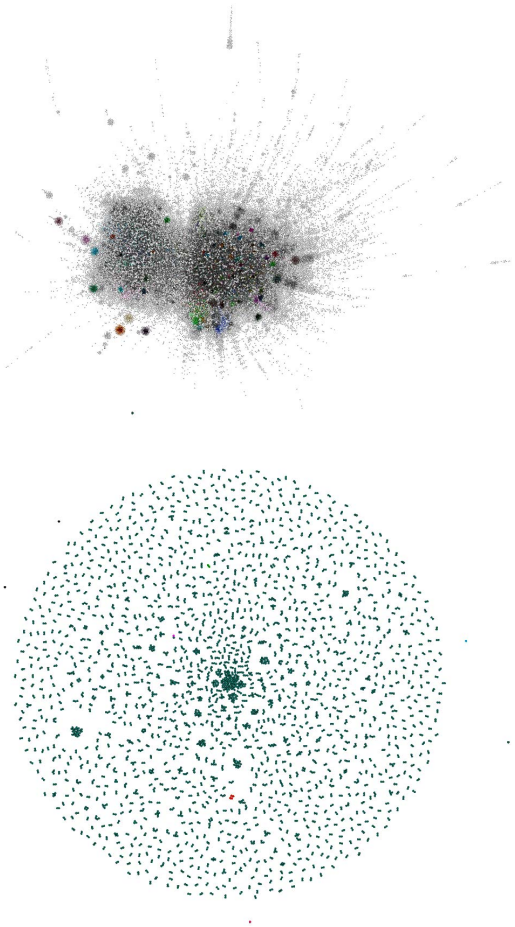

Fig. 5. The reduced subset graph with the densely connected largest partition (top) and the sparsely connected remaining non-singletons (bottom).

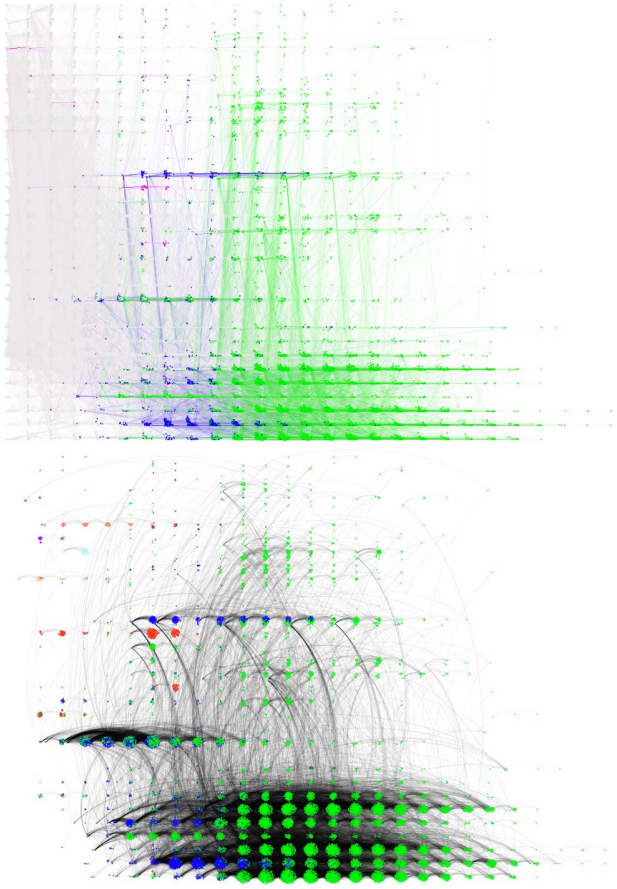

Fig. 6. Subset relationship between interfaces. Spatial arrangement with the maximal root distance on the horizontal axis and 32 Louvain communities vertically. The upper plot shows all interfaces of the largest partition in the reduced subset graph, while the lower plot contains only labeled interfaces.

which makes them natural candidates for being contained in other contracts. We also notice that blue interfaces (non-compliant tokens) mostly appear to the left of green interfaces. This is not surprising, as most non-compliant tokens omit some of the functions mandated by the respective standard.

The lower plot only retains the labeled interfaces and keeps the edges uniformly in black. We notice that the green interfaces (ERC-compliant tokens) appear in pronounced blobs along the subset axis (horizontal) of several Louvain communities (vertical). The same holds for the blue interfaces (non-compliant tokens). Moreover, a few wallets (pink) show a similar pattern in the plot.

VIII. Projected Graph with Card Similarity

In this section, we briefly discuss the projected graph of interfaces with *Card* similarities as edge weights.

About half of the interfaces share at least 10 signatures with some other interface. A cutoff at this level yields a graph with unmanageable 384 million edges. The high number of overlapping signatures is mainly due to token contracts, as the popular ERC-20 standard mandates six signatures and proposes another three.

The situation changes markedly with a cutoff value of 11, where the number of edges drops to 72 million and then halves with every increment of the cutoff.

A cutoff of 16 shared signatures yields a graph with 30 000 nodes and 3.5 million edges. The larger clusters are dominated by token interfaces, while most wallet interfaces have been excluded.

Our experiments support the assessment in section IV that *Card* similarities probably are less useful in obtaining interesting clusters for a wide range of interfaces.

IX. Conclusions

Regarding interface similarity, we conclude with the following observations. (i) The bipartite graph expresses the high similarity in extensive interconnection patterns. (ii) With *Jaccard* weights, we can uncover details of the similarities that seem reasonable for the labeled interfaces from our ground truth. (iii) *Overlap* similarity shows that surprisingly many interfaces contain each other. This hints at significant functional similarities. (iv) Likewise, *Card* similarity shows that quite many interfaces share a large number of signatures.

With respect to the suitability of the proposed clustering methods, we conclude as follows. (i) In the bipartite graph, both WCC and Louvain do not yield satisfactory results. WCC is too coarse, and Louvain places dissimilar interfaces into the same a cluster while yielding numerous smallish partitions. (ii) In the projected similarity graph, *Jaccard*-based clustering performs best, but tends to form clusters that may contain pairs of dissimilar interfaces, especially with lower cutoffs, while *Overlap* and *Card* disregard relevant details by treating the subset relation as equality or by ignoring smaller interfaces.

For our ground truth of labeled interfaces, we find that (i) wallet interfaces can be grouped in a fine-grained manner with similarity clustering (as shown with all approaches); (ii) token interfaces are more similar than wallet interfaces because the majority of token interfaces tend to assemble in the largest cluster in all approaches and they show high interconnectivity, while the wallets tend to separate into their own clusters.

Further Work. Jaccard similarity can be improved by taking the frequency of signatures (their significance) into account. This may help in subdividing the larger clusters with token interfaces. Another approach is multi-level clustering: After removing common signatures in a cluster, a second round of clustering may reveal more details.

Clusters and their signatures resemble the idea of formal concepts. Formal concept analysis [16] yields a hierarchy of clusters. To single out the interesting ones, we need a better understanding of the criteria a good cluster should meet.

References

[1] M. Di Angelo and G. Salzer, "Characteristics of Wallet Contracts on Ethereum," in *2nd Conference on Blockchain Research and Applications for Innovative Networks and Services (BRAINS'20)*. IEEE, 2020.

[2] "Ethereum improvement proposals: Application-level standards and conventions," accessed 2020-07-16. [Online]. Available: https://eips.ethereum.org

[3] "Contract ABI Specification," 2019, accessed 2019-09-09. [Online]. Available: https://solidity.readthedocs.io/en/latest/abi-spec.html

[4] M. Di Angelo and G. Salzer, "Tokens, Types, and Standards: Identification and Utilization in Ethereum," in *Int. Conf. Decentralized Applications and Infrastructures (DAPPS)*. IEEE, 2020.

[5] T. Chen, Z. Li, Y. Zhu, J. Chen, X. Luo, J. C.-S. Lui, X. Lin, and X. Zhang, "Understanding Ethereum via Graph Analysis," *ACM Transactions on Internet Technology (TOIT)*, vol. 20, no. 2, pp. 1–32, 2020.

[6] R. Norvill, I. U. Awan, B. B. F. Pontiveros, A. J. Cullen *et al.*, "Automated Labeling of Unknown Contracts in Ethereum," in *ICCCN 26th Int. Conf. on Computer Communications and Networks*. IEEE, 2017.

[7] X. T. Lee, A. Khan, S. Sen Gupta, Y. H. Ong, and X. Liu, "Measurements, Analyses, and Insights on the Entire Ethereum Blockchain Network," in *Proceedings of The Web Conference 2020*. New York, NY, USA: ACM, 2020, pp. 155–166.

[8] N. He, L. Wu, H. Wang, Y. Guo, and X. Jiang, "Characterizing Code Clones in the Ethereum Smart Contract Ecosystem," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 654–675.

[9] H. Liu, Z. Yang, Y. Jiang, W. Zhao, and J. Sun, "Enabling Clone Detection For Ethereum Via Smart Contract Birthmarks," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 2019, pp. 105–115.

[10] M. Kondo, G. A. Oliva, Z. M. J. Jiang, A. E. Hassan, and O. Mizuno, "Code Cloning in Smart Contracts: A Case Study on Verified Contracts from the Ethereum Blockchain Platform," *Empirical Software Engineering*, 2020.

[11] M. Fröwis, A. Fuchs, and R. Böhme, "Detecting Token Systems on Ethereum," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019.

[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast Unfolding of Communities in Large Networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, 2008.

[13] P. Jaccard, "Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines," *Bulletin de la Société Vaudoise des Sciences Naturelles*, vol. 37, pp. 241–272, 1901.

[14] M. Vijaymeena and K. Kavitha, "A Survey on Similarity Measures in Text Mining," *Machine Learning and Applications: An International Journal*, vol. 3, no. 1, pp. 19–28, 2016.

[15] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, "ForceAtlas2, a Continuous Graph Layout Algorithm for Handy Network Visualization Designed for the Gephi Software," *PLOS ONE*, vol. 9, no. 6, pp. 1–12, 2014.

[16] B. Ganter and R. Wille, *Formal Concept Analysis - Mathematical Foundations*. Springer, 1999.