Clustering Natural Language Test Case Instructions as Input for Deriving Automotive Testing DSLs

Katharina Juhnke[*], **Alexander Nikic**[**], **and Matthias Tichy**[*]
[*]Institute of Software Engineering and Programming Languages, Ulm University, Germany
[**]Mercedes-Benz AG, Germany

ABSTRACT System testing is an important quality assurance technique in the area of automotive software development where predominantly natural language test cases are used for testing prototype vehicles. To ensure that these test cases are able to identify faults, the test cases themselves must also be of high quality. Testing DSLs can improve the quality of the test cases. To support the smooth introduction of Testing DSLs into the industry, the reuse of system-specific terminology and syntax of existing natural language test case specifications is recommended. Consequently, it is necessary to identify and cluster highly similar domain-specific instructions used in the action and expected result descriptions from those specifications. This is a key activity in automating the development of Testing DSLs to enable the application of grammar inference approaches. However, with an average of $400 - 500$ test cases per specification, this is a time-consuming task when executed manually. We present a clustering approach based on the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm to automatically cluster similar instructions. Due to the special structure of the instructions used in our industrial test case specifications, we further developed a specific distance function required by the DBSCAN algorithm. Additionally, we determined appropriate values for the DBSCAN's parameters *MinPts* and *Eps*. Our evaluation on three industrial test case specifications shows that our approach is suitable for automatically clustering instructions from those test case specifications due to an almost perfect agreement ($\kappa > 0.81$) between clusters, created manually by experts, and the automatically created clusters. Furthermore, we show how to use the Multiple Sequence Alignment (MSA) approach to automatically derive grammar suggestions from the clustered instructions.

KEYWORDS Automotive, Test case templates, DSL, DBSCAN, Distance function, MSA.

1. Introduction

Natural language test case specifications are commonly used in the automotive domain to specify test cases for a specific system, especially for system tests that are manually executed by human testers in a prototype vehicle (Hauptmann et al. 2013; Lachmann & Schaefer 2014; Juhnke et al. 2018). The core of such test cases are the descriptions of *actions* to be performed and the corresponding *expected results* to be checked, each consisting of a sequence of *instructions* (e.g., *"close passenger door"* or *"press unlock key button"*). Similar to the problems known from natural language requirements, the documentation of test cases in natural language often leads to incomprehensible, ambiguous and unmaintainable test cases (cf. Juhnke et al. 2018) identified in particular problems related to the natural language description of automotive test cases, such as translation and spelling errors, inconsistent phrasing, inconsistent use of vocabulary, different description styles for similar test procedures or excessive use of abbreviations. Consequently, test designers and testers demanded mechanisms for a more consistent documentation of test cases of their system to avoid ambiguities and misunderstandings during test execution. To meet this requirement, Juhnke & Tichy (2019) propose the use of *system-specific Automotive Testing DSLs* for test case documentation and a

JOT reference format:
Katharina Juhnke, Alexander Nikic, and Matthias Tichy. *Clustering Natural Language Test Case Instructions as Input for Deriving Automotive Testing DSLs*. Journal of Object Technology. Vol. 20, No. 3, 2021. Licensed under Attribution 4.0 International (CC BY 4.0)
http://dx.doi.org/10.5381/jot.2021.20.3.a5

domain analysis method for developing such Domain Specific Languages (DSLs) for different systems. For instance, over 100 different systems are installed in premium cars today.

In most cases, manually quality-checked test case specifications already exist for a system from another model series or variant, so that they can be used as starting point for the definition of system-specific Testing DSLs. First, the different instructions used in action and expected result descriptions of all test cases (input) are clustered to identify similar instructions. Based on this, so-called *conceptual templates* (output) are derived which represent common parts as well as alternatives and optional parts of all instructions contained in a cluster. These conceptual templates are suitable to be used by domain engineers for the definition of Testing DSLs and to cover 70%–95% of a test case specification (see Juhnke & Tichy 2019).

Our experience in a testing department at Mercedes-Benz has shown that it takes a domain engineer approx. two days to manually derive conceptual templates for actions and expected results from a smaller test case specification with about 250 test steps. Very large specifications can contain up to $16,000$ test cases, which can be very time-consuming. Hence, it is reasonable and necessary to automate the activities regarding the aggregation of instructions and derivation of conceptual templates.

The contribution of this paper is an approach to automate those activities based on existing manually quality-checked test specifications. A special focus of our work lies on the clustering of the instructions used in action and expected result descriptions, which forms the basis for the derivation of conceptual templates. Any existing quality issues like the aforementioned translation errors or spelling errors in the input test case specifications would still be in the conceptual templates. However, they are now more easily to spot in the templates by domain experts. The following two research questions guide our work:

RQ1: *How can the similarity between instructions be determined automatically so that similar instructions can be assigned to the same cluster?*

RQ2: *How good is the result of the automated clustering of instructions compared to groups created manually by domain experts?*

We compare different similarity measures for determining similarity between instructions, including Levenshtein (Levenshtein 1966), Jaro-Winkler (Winkler 1990), Trigram similarity (based on n-grams (Shannon 1948)), and a measure for determining similarities from plagiarism detection called Sherlock similarity (Joy & Luck 1999; White & Joy 2004). In addition, we present our own developed similarity measure called Run Length and its results. For clustering instructions, we use the DBSCAN algorithm (Ester et al. 1996) and determine a suitable *Eps* parameter by a heuristic approach based on the determination of the k-th nearest neighbor. We compare the results of the automated clustering with the results of manual grouping for three real automotive test case specifications. To determine the agreement between the automated and manual results, we calculated *Cohen's Kappa* (κ) (Cohen 1960). We obtain $\kappa > 0.81$ for all three test case specifications, which corresponds to an almost perfect agreement. In addition, we show how a Multiple Sequence Alignment (MSA) approach can be used to derive conceptual templates from our clustering results, which then have to be manually refined into a system-specific Testing DSL.

The remainder of this paper is structured as follows: Section 2 discusses related work regarding grammatical inference, clustering, and appropriate distance functions for texts. Section 3 presents our developed automated approach for clustering instructions used in action and expected result descriptions. Section 4 shows the evaluation results of our approach. Finally, Section 5 provides the conclusion and future work.

2. Related work

In this section we discuss, with regard to our action and expected result descriptions as input, approaches to grammatical inference, common methods for clustering data, and types of similarity measures that are needed to calculate the distance.

2.1. Grammatical inference

Grammatical Inference (GI) is a subarea of machine learning and is about the process of learning a grammar based on positive and/or negative samples. Applications in software engineering are the inference of general purpose programming languages (GPLs), DSLs, graph grammars, and visual languages (Stevenson & Cordy 2013).

The inference of DSLs is intended to support domain experts who may not be familiar with language design or implementation, which is also our intention. For this purpose, the syntactic structure of the underlying grammar is generated based on example DSL programms (Bryant et al. 2010) or sample sentences of the language. This means that the input are formal languages rather than natural languages, which does not apply to our natural language test case specifications. Our action and expected result descriptions are available as unstructured data, i.e., they are not based on an already existing DSL or example DSL program, they contain ambiguities due to their natural language elements, and they are not classified as positive or negative samples in any way by domain experts.

However, most GI approaches generate grammars from positive samples, such as GenInc (Javed et al. 2008), PACS (M. Li & Vitanyi 2008), MAGICe (Mernik et al. 2009), or the MDL approach from (Sapkota et al. 2012). For example, the GenInc learning algorithm requires sorted positive samples to incrementally derive a grammar. The difference between two consecutive samples should be small, which limits the use of GenInc for real problems. Furthermore, a different order of the samples can lead to a wrong grammar (Sapkota et al. 2012). The example shows that in order to apply these methods to our actions and expected result descriptions, positive samples from a test case specification would first have to be identified by a domain expert. Otherwise, applying GI approaches to a non-preprocessed test case specification, which contains constructs that are faulty and would not be used in the future, would lead to useless results.

Moreover, our industry experience reveals that domain experts have trouble defining good examples from scratch and it works better if we show them typically used instructions from previous test case specifications. But as already mentioned, it takes time to extract these instructions and aggregate similar variants. By clustering similar instructions, we obtain a set of instructions that have similar linguistic patterns (e.g., terminology and syntax) and are therefore better suited as input for a GI approach than the set of all instructions (including unlabeled faulty ones) of a test case specification. Additionally, the clustered instructions can be evaluated by domain experts, so that we get positive (and negative) sample sentences as well as the necessary labeling of our data, to apply GI approaches based on positive and/or negative samples, much faster. Therefore, our approach is based on an automated clustering of the contained instructions, which goes beyond the recognition of repetitions as for example used by the Sequiture algorithm (Nevill-Manning & Witten 1997) to directly derive grammars. For this purpose, we first have to adequately recognize similarities between instructions and cluster them. Subsequently, a meaningful derivation of conceptual templates by using GI algorithms is possible. In the following two sections we will therefore consider clustering approaches and distance functions.

2.2. Clustering

To the best of our knowledge, there is no work that clusters natural language test case descriptions, especially not from the automotive domain. Therefore, we consider general cluster approaches in the following.

Clustering is an unsupervised learning technique that groups data elements into clusters. Data elements in the same cluster are similar while data elements between clusters differ significantly. There are different clustering approaches, which can be divided into the following (Xu & Wunsch 2005):

- **Partitioning-based clustering**: divides the data elements into k partitions, where one partition corresponds to a cluster. The most common representative is the k-Means algorithm (MacQueen 1967).
- **Hierarchical-based clustering**: structures data in hierarchies based on a neighborhood matrix. Well-known representatives are agglomerative and divisive methods (Maimon & Rokach 2005). Agglomerative methods operate according to the bottom-up principle, in which each data element initially represents a unique cluster and is then merged with the nearest data element to form a new cluster. This step repeats until all data elements form a cluster. Divisive methods act in exactly the opposite way.
- **Fuzzy-based clustering**: also called soft clustering, where the data elements can belong to several clusters. Fuzzy-c-Means (Bezdek 1981) is a widely known example.
- **Density-based clustering**: groups data elements to clusters together according to a certain density and a minimum number of members. One of the most popular algorithms is the DBSCAN algorithm from Ester et al. (1996).

Each of these clustering approaches has its strengths and weaknesses. Partitioning-based approaches expect a given k for the number of clusters to be determined, which is difficult to predict for unknown data. Hierarchical-based clustering offers different linkage methods, each leading to different results. Additionally, the determination of the clusters has to be done manually using dendrograms. When using fuzzy-based clustering approaches, it happens that a data element is assigned to several clusters. This requires also a manual intervention at the end in order to clearly assign the respective data elements to a cluster. Moreover, none of these three approaches considers outliers in the data. Density-based clustering algorithms, like the DBSCAN algorithm, can deal with outliers. If an instruction cannot be assigned to a cluster, it remains as a non-clustered instruction. This may apply in particular to individual instructions which are, for example, unique and have no similarities with other instructions. In addition, usually it is not known how many clusters can be formed from instructions of a test case specification, which is why the DBSCAN algorithm is suitable in this case. Furthermore, the DBSCAN algorithm does not limit the cluster size (number of instructions within a cluster) and an instruction can only be assigned to exactly one cluster. Therefore, we use the DBSCAN algorithm for clustering instructions as a representative for density-based clustering approaches.

2.3. Distance functions

In order to assign instructions to clusters using the DBSCAN algorithm, a distance function is required which indicates how different the instructions are. The value 0 usually represents a perfect match. To calculate the distance, text similarity measures are used, which can be distinguished as follows:

- **Edit-distance-based**: e.g., Levenshtein (Levenshtein 1966) and Jaro-Winkler (Winkler 1990) similarity calculate the minimum number of operations needed for transforming a string s_1 into string s_2. The smaller the value, i.e., the less editing operations are necessary, the more similar the strings s_1 and s_2 are.
- **Token-based**: e.g., Jaccard Index (Jaccard 1912) of two sets is calculated by the number of common elements divided by the size of the union set. Two strings can be compared based on the contained tokens (e.g., single words or n-grams (Shannon 1948)).
- **Sequence-based**: e.g., Ratcliff-Obershelp (Ratcliff & Metzener 1988) searches between two strings for the longest common substring. Parts that do not match are trimmed off and more common substrings are searched until the end of a string is reached.

The different text similarity measures each have their advantages and disadvantages, which also emerges from various studies, such as (Huang 2008; Strehl et al. 2000). In sequence-based approaches, it may happen that words are trimmed in the middle and assigned with other words as “equal”, even though they are semantically independent from each other. This can lead to supposedly good results with semantically incorrect correlations and is therefore not used for our instructions. Consequently, we consider edit-distance-based and token-based similarity measures as suitable, because edit-distance-based similarity measures are particularly well suited for short instructions or single words and token-based similarity measures are more appropriate for longer instructions. Both may be applicable for instructions in automotive test case specifications.

Furthermore, typical natural language processing techniques for comparing texts are Word2Vec (Mikolov et al. 2013), Doc2Vec (Le & Mikolov 2014), or TF-IDF (Rajaraman & Ullman 2011). These techniques require the transformation of the strings into vectors so that based on these vectors so-called "true" metrics like the euclidean distance or the cosine similarity can be used to compare these strings (Huang 2008). However, vectorizing the strings may result in high dimensions, where investigations (Aggarwal et al. 2001) have shown that the distances between the data elements become more uncertain by means of the euclidean distance.

Moreover, Veni (2009) and Taghva & Veni (2010) show that different distance functions have different effects on the clustering algorithms used, depending on the type of text to be compared. There is no global solution for every type of text. Hence, existing distance functions are often modified for special applications (Eldesoky et al. 2009; Morichetta et al. 2016; L. Li & Li 2017). Karypis et. al (Karypis et al. 2000) compared different document loss techniques and show that token-based methods, such as *n*-grams (Shannon 1948), predominantly achieve better results in text clustering.

In summary, the similarity measure used must be aligned with the underlying data (i.e., automotive test case specifications) and the effects on the clustering algorithm must be investigated. For this we consider the Jaro-Winkler and Levenshtein distance as representatives of the edit-distance-based similarity measures and a trigram similarity measure based on the Jaccard Index as representative of the token-based similarity measure to be promising. In case this is not successful, we have to consider the modification of existing distance functions for our specific applications.

3. Automation of domain analysis activities

The implementation of our automated approach is strongly oriented to the manual approach introduced by Juhnke & Tichy (2019). The core activities of this approach are shown in Figure 1 together with an annotation (green highlighting) of which activities have been (semi-) automated by our approach and what the respective inputs and outputs are.

In the following we explain the preparation of the test case descriptions for clustering (semi-automated activities within Step 1, Sec. 3.1), our developed clustering approach for instructions (Sec. 3.2), and how the clustering results are used to automatically derive conceptual templates (Step 2, Sec. 3.3).

3.1. Preparations for clustering

First, we split the action and expected result descriptions which each contain several instructions (e.g., *'close passenger door'* or *'press unlock key button'*), into *instructions*, depending on enumeration characters and line breaks. After that, we automated the rectification of instructions in Step 1 only to a limited extent, which is why this activity is shown in Figure 1 as semi-automated (dashed line). The recognition and automated correction of spelling mistakes is not possible with the aid of conventional dictionaries, since the instructions contain domain-specific technical terms, automotive specific abbreviations or signal names. However, such domain-specific dictionaries do not currently exist, not even at Mercedes-Benz. Hence, in preparation for the clustering of instructions, we only normalize multiple spaces to one. Then, we decompose the instructions into

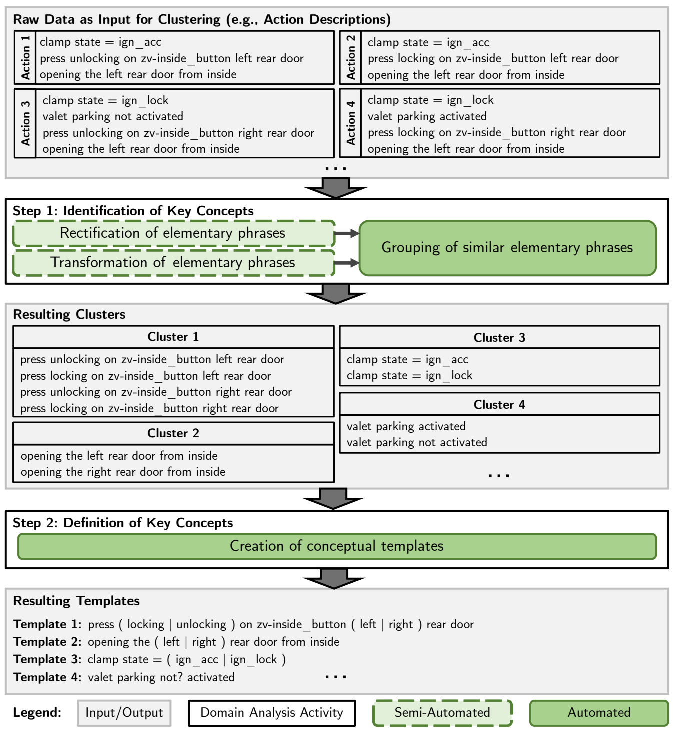
Raw Data as Input for Clustering (e.g., Action Descriptions)
Action 1
clamp state = ign_acc
press unlocking on zv-inside_button left rear door
opening the left rear door from inside
Action 2
clamp state = ign_acc
press locking on zv-inside_button left rear door
opening the left rear door from inside
Action 3
clamp state = ign_lock
valet parking not activated
press unlocking on zv-inside_button right rear door
opening the left rear door from inside
Action 4
clamp state = ign_lock
valet parking activated
press locking on zv-inside_button right rear door
opening the left rear door from inside
...
Step 1: Identification of Key Concepts
Rectification of elementary phrases
Transformation of elementary phrases
Grouping of similar elementary phrases
Resulting Clusters
Cluster 1
press unlocking on zv-inside_button left rear door
press locking on zv-inside_button left rear door
press unlocking on zv-inside_button right rear door
press locking on zv-inside_button right rear door
Cluster 2
opening the left rear door from inside
opening the right rear door from inside
Cluster 3
clamp state = ign_acc
clamp state = ign_lock
Cluster 4
valet parking activated
valet parking not activated
...
Step 2: Definition of Key Concepts
Creation of conceptual templates
Resulting Templates
Template 1: press (locking | unlocking) on zv-inside_button (left | right) rear door
Template 2: opening the (left | right) rear door from inside
Template 3: clamp state = (ign_acc | ign_lock)
Template 4: valet parking not? activated
...
Legend:
Input/Output
Domain Analysis Activity
Semi-Automated
Automated

Figure 1 Overall process with completely automated (green boxes with solid line) and semi-automated (green boxes with dashed line) activities of the domain analysis method

individual tokens using a string tokenizer. A token is a sequence of characters or a word. The space character is used as delimiter between the tokens. Afterwards we completely convert each token to lower case letters. This facilitates the later comparison of instructions and can also eliminate typing errors, such as *"I͟sw_Stat"* instead of *"IS͟w_Stat"*. For the performed instruction analyses, we found no cases in which the unification in lower case caused errors or unintentionally merged instructions in the used industrial test case specifications.

Furthermore, there are also restrictions for automating the transformation of instructions, since expert knowledge is required for this. In order to support this activity, our developed algorithm offers the possibility to define a placeholder and assign to it synonymously used terms, signal names or values. For this the domain engineer has to create a .txt file, which first contains a freely selectable name for the placeholder (e.g., __IGNITION_STATES__). After that, the file contains a list of terms separated by line breaks which are to be replaced by the defined placeholder, e.g., ignition positions IGN_LOCK, IGN_ACC, IGN_ON, or IGN_START. This procedure allows the domain engineer and domain experts to take already known inconsistencies in test case specifications (e.g., different technical terms) into account before clustering instructions.

3.2. Clustering of instructions

Similar instructions are grouped together in a cluster if they have similarities in terms of the tokens they contain. For this we use the DBSCAN algorithm described by Ester et al. (1996) that, applied to our data material (i.e., the instructions), requires the following two parameters:

(1) Eps, maximum distance between two neighboring instructions

(2) $MinPts$, minimum number of instructions in an Eps-neighborhood of an instruction to form a cluster with these instructions

Accordingly, two instructions p_X and p_Y from a test case specification T are considered neighbors, if p_Y is in the Eps-neighborhood N_{Eps} of p_X. For this, the distance $dist(p_X, p_Y)$ between the instructions p_X and p_Y must be less than or equal to Eps (Ester et al. 1996):

$$N_{Eps}(p_X) = \{p_Y \in T | dist(p_X, p_Y) \leq Eps\} \qquad (1)$$

In this case, p_Y is also called *directly density-reachable* from p_X. In addition, two instructions p_A and p_B are assigned to the same cluster if a chain of instructions $p_1, ..., p_n$ with $p_1 = p_A$ and $p_n = p_B$ exists, such that p_{i+1} is directly density-reachable from p_i, i.e., $p_{i+1} \in N_{Eps}(p_i)$. In this case, p_A and p_B are called *density-connected* to each other. Instructions that are neither *directly density-reachable* nor *density-connected* or inside a N_{Eps} neighbourhood are defined as outliers. For clustering instructions appropriately, we require a suitable distance function $dist(p_A, p_B)$ to determine the similarity between instructions. In addition, the parameters Eps and $MinPts$ of the DBSCAN algorithm have to be defined appropriately. This will be discussed in the following two subsections.

3.2.1. Similarity measures for comparing instructions

To determine the similarity between instructions, we applied edit-distance-based and token-based distance functions (cf. Sec. 2) to automotive test case descriptions. Table 1 shows computed results using the example of a German test case specification for the edit-distance-based similarity measures *Levenshtein similarity* sim_l (Levenshtein 1966) and *Jaro-Winkler similarity* sim_{jw} (Winkler 1990) as well as for the token-based similarity measures *Trigram similarity* sim_t (Shannon 1948; Ukkonen 1992) and *Sherlock similarity* sim_s (Joy & Luck 1999; White & Joy 2004). Usually the similarity measures between two instructions p_A and p_B in Table 1 take a value between 0 (maximum difference) and 1 (perfect match) with the exception of sim_t.

Results for the comparison of edit-distance-based and token-based similarity measures As can be seen from Table 1, the Levenshtein similarity is usually less than the Jaro-Winkler similarity. This is due to the prefix scale that Jaro-Winkler similarity uses, which gives higher ratings to strings with the same beginning. Thus, even instructions that have only few similarities receive quite high similarity values, such as $sim_{jw} = 0.87$ in row number 4 in Table 1. The Levenshtein similarity measure is unsuitable if words within a string are transposed or occur in different order, such as $sim_l = 0.09$ in row number 2 in Table 1.

Furthermore, it can be seen from Table 1 that the Trigram similarity is strongly dependent on the length of the instructions. Instructions smaller than three words do not contain trigrams, which is indicated by a negative value ($sim_t = -1.00$). Hence, this similarity measure is particularly suitable for instructions with more than three tokens or words, which is not always the case in automotive test case descriptions (e.g., parameter-value assignments usually consist of only three words).

Development of the Run Length similarity measure The similarity measures sim_l and sim_t do not provide satisfactory results. Only the Jaro-Winkler similarity measure sim_{jw} shows convincing values for different kinds of instructions, sometimes a bit too optimistic, but quite promising enough to be examined in conjunction with the clustering algorithm (see Sec. 3.2.2). Therefore, we additionally considered similarity measures that are used in plagiarism detection algorithms especially for the detection of source code plagiarism, since source code has a certain similarity to parameter assignments in automotive test case descriptions and the instructions per line are rather short. For instance, the algorithms Ferret (Lyon et al. 2004) and Sherlock (Joy & Luck 1999; White & Joy 2004) are based on token-based distance functions, where Ferret resembles the Trigram similarity sim_t. In contrast, the Sherlock similarity originates from sentence-based natural language plagiarism detection (Joy & Luck 1999; White & Joy 2004) and is based on the assumption that two documents can be considered plagiarism, i.e., similar, if they have a limited number of structural changes. According to the Sherlock algorithm defined by the University of Warwick (University of Warwick 2014), the similarity of two sentences s_1 and s_2 is calculated based on the sentence lengths (e.g., $len(s_1)$), which is defined by the number of words. In addition, the number of common words $com(s_1, s_2)$ of two sentences s_1 and s_2 is calculated. Hence, the Sherlock similarity sim_s for sentences s_1 and s_2 is calculated as follows:

$$sim_s(s_1, s_2) = \frac{1}{2}\left(\frac{com(s_1, s_2)}{len(s_1)} + \frac{com(s_1, s_2)}{len(s_2)}\right) \qquad (2)$$

The Sherlock *similarity score* is given by $100 \cdot sim_s$ with $0 \leq sim_s \leq 1$. White & Joy (2004) define a *similarity threshold* of 80 and a *common threshold* of 6 words, i.e., two sentences are similar if they have a similarity of $sim_s > 80$ or have more than six words in common ($com(s_1, s_2) > 6$). However, these thresholds result from experience gained from the comparison of source code comments and may not be transferable to automotive test case specifications. Furthermore, White & Joy (2004) perform an extensive preprocessing of the sentences to be compared, e.g., words that occur too often and do not appear useful for comparison and duplicate words are eliminated. Especially the removal of duplicates in an instruction entails the risk of introducing considerable changes in the meaning of the instructions. Therefore, the preparation of instructions in Step 1 does not include such changes, but only the adjustments described in Section 3.1. Nevertheless, the good similarity results (cf. Tab. 1) led to the idea to adjust the Sherlock similarity measure, which resulted in the creation of the Run Length similarity sim_{rl}.

Our developed Run Length similarity measure is based on *runs*. Joy et al. (Joy & Luck 1999) define a *run*, as sequence of sentences that two documents have in common. The sequence of these common sentences does not have to be coherent. In the context of comparing two instructions, a run is a sequence of common words that occur in the same order in both instructions and form a coherent sequence. Between two instructions p_A

#	**Instruction p_A**	**Instruction p_B**	sim_l	sim_{jw}	sim_t	sim_s[a]	sim_{rl}
1	beifahrertür schließen	beifahrertür schließen	1.00	1.00	-1.00[b]	1.00	1.00
2	beifahrertür schließen	schließen __ARTICLE__ beifahrertür	0.09	0.54	0.00	0.83	0.33
3	schließen __ARTICLE__ drehfalle heckdeckel	schließen __ARTICLE__ heckdeckeldrehfalle	0.60	0.95	0.00	0.58	0.50
4	abwarten von busruhe	abwarten von powerdown	0.64	0.87	0.00	0.67	0.67
5	aktivierung kindersicherung fondtür rechts	aktivierung kindersicherung fondtür links	0.88	0.96	0.33	0.75	0.75
6	taste entriegeln auf __ARTICLE__ zv-innentaster hinten links betätigen	taste entriegeln auf __ARTICLE__ zv-innentaster vorne links betätigen	0.93	0.95	0.33	0.88	0.88
7	schlüsseltaste entriegeln betätigen	zv-innentaster fahrertür entriegeln betätigen	0.58	0.68	0.00	0.58	0.50
8	abwarten von powerdown __ARTICLE__ ezs	öffnen __ARTICLE__ tankdeckels	0.26	0.64	0.00	0.27	0.20
9	taste entriegeln auf __ARTICLE__ zv-innentaster vorne links betätigen für zvi_tastenbetätigung_halten	mehrfach taste entriegeln auf __ARTICLE__ zv-innentaster vorne links betätigen	0.60	0.73	0.67	0.84	0.80
10	taste entriegeln auf __ARTICLE__ elektronischen fahrzeugschlüssel für schlüsselbetätigung_kurz innerhalb von __NUMBER__ sec nach erreichen __ARTICLE__ spielschutzes betätigen	taste verriegeln auf __ARTICLE__ elektronischen fahrzeugschlüssel für schlüsselbetätigung_kurz betätigen innerhalb von __NUMBER__ sec nach erreichen __ARTICLE__ spielschutzes	0.98	0.90	0.59	0.94	0.94
11	zeitgleich taste verriegeln auf __ARTICLE__ zv-innentaster vorne links und taste verriegeln auf __ARTICLE__ zv-innentaster hinten rechts betätigen	taste entriegeln auf __ARTICLE__ zv-innentaster hinten links betätigen	0.45	0.66	0.06	0.73	0.35

a The Sherlock algorithm calculates a similarity score between 0 and 100. For better readability the calculated score was divided by 100 to get a value between 0 and 1 that is comparable to the other similarity values.
b Total length of both instructions is less than 3, therefore no trigram can be formed. To indicate this, the resulting similarity is negative, i.e., -1.00.
Note: *We evaluated the different similarity measures on industrial test cases from Mercedes-Benz which are predominantly written in German. Hence, we chose to report the original measures on the original German instructions.*

Table 1 Results of various tested similarity measures

and p_B there can be several runs, so that the sum of the length of all runs is defined by the total run length $trl(p_A, p_B)$. Figure 2 shows an example for calculating the total run length between two instructions p_1 and p_2.

To normalize this similarity measure to values between 0 and 1, we divided the total run length by the maximum length of both instructions, which is the number of words in the longest instruction, i.e., $len(p_A)$ or $len(p_B)$. Hence, the Run Length similarity sim_{rl} is calculated as follows:

$$sim_{rl}(p_A, p_B) = \frac{trl(p_A, p_B)}{max\,(len(p_A), len(p_B))} \tag{3}$$

The Run Length similarity measure represents a downward estimate compared to the original Sherlock similarity measure, i.e., $sim_s \geq sim_{rl}$. Hence, the Run Length similarity is more stringent. This is shown by the following mathematical derivation based on sim_s (cf. Equation 2):

1. It applies $len(p_A), len(p_B) \leq max(len(p_A), len(p_B))$:

$$\frac{1}{2}\left(\frac{com(p_A,p_B)}{len(p_A)} + \frac{com(p_A,p_B)}{len(p_B)}\right) \geq \frac{1}{2}\left(\frac{com(p_A,p_B) + com(p_A,p_B)}{max\,(len(p_A), len(p_B))}\right)$$

2. It applies $com(p_A, p_B) \geq trl(p_A, p_B)$:

$$\frac{1}{2}\left(\frac{com(p_A,p_B)}{len(p_A)} + \frac{com(p_A,p_B)}{len(p_B)}\right) \geq \frac{1}{2}\left(\frac{2 \cdot trl(p_A,p_B)}{max\,(len(p_A), len(p_B))}\right)$$

3. Summarize equation:

$$\frac{1}{2}\left(\frac{com(p_A,p_B)}{len(p_A)} + \frac{com(p_A,p_B)}{len(p_B)}\right) \geq \frac{trl(p_A,p_B)}{max\,(len(p_A), len(p_B))}$$

4. Hence it applies: $sim_s \geq sim_{rl}$

Step two of the estimation downwards is based on the fact that for the calculation of the total run length $trl(p_A, p_B)$ the order of the tokens is taken into account, while $com(p_A, p_B)$ only checks if words from instruction p_A are also contained in instruction p_B, regardless of their position. Instructions that contain the same token more than once are a special case. In this case $com(p_A, p_B) \leq trl(p_A, p_B)$ would apply. However, based on the examined automotive test case specifications in the context of this work, test case descriptions rarely contain the same tokens more than once, since they are usually written in note form or in short sentences.

As can be seen from Table 1, sim_{rl} returns in most cases somewhat lower similarity values than sim_{jw}, but this is desirable, especially for $sim_{rl} = 0.20$ in row number 8.

Selected distance functions In conclusion, for clustering using the DBSCAN algorithm, we considered the Jaro-Winkler and the Run Length similarity to be promising. The DBSCAN algorithm requires a distance measure $dist(p_A, p_B)$, which provides values between 0 and 1, where 0 represents a perfect match and 1 the maximum difference. Therefore, the *Jaro-Winkler distance* $dist_{jw}(p_A, p_B)$ and the *Run Length distance* $dist_{rl}(p_A, p_B)$ are defined based on the presented similarity measures as follows:

$$dist_{jw}(p_A, p_B) = 1 - sim_{jw}(p_A, p_B) \tag{4}$$

$$dist_{rl}(p_A, p_B) = 1 - sim_{rl}(p_A, p_B) \tag{5}$$

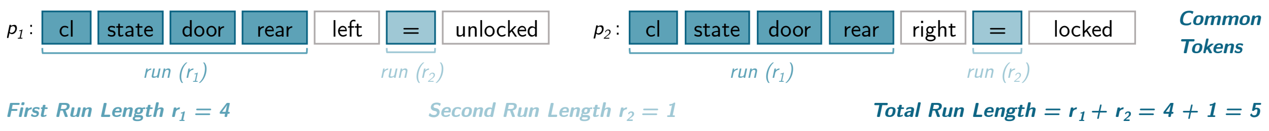

Figure 2 Example for calculating the total run length between two instructions.

Based on this, we considered the determination of suitable values for the parameters *Eps* and *MinPts* of the DBSCAN algorithm in the following.

3.2.2. Determining suitable parameters for DBSCAN algorithm

As mentioned at the beginning of Section 3.2, the DBSCAN algorithm requires the two parameters *MinPts* and *Eps*, whose determination is explained below.

Parameter *MinPts* Ester et al. (1996) suggest the default value 4 for the parameter $MinPts$. However, this recommendation refers to two-dimensional data. In the case of the instructions to be clustered, however, these are only one-dimensional data. According to the recommendation of Sander et al. (1998), the value for $MinPts$ should be as small as possible and a suitable value should be calculated as $MinPts = 2 \cdot dim - 1$, where dim represents the dimensionality of the data. However, it should apply $MinPts > 1$, otherwise the *"single-link effect"* may occur (Sander et al. 1998). It may happen that objects are clustered in one large cluster, although it makes more sense to split them into two clusters. This is exactly the case if there is a chain of objects between these two clusters, where the distance of each object to the neighboring object is smaller than Eps. Based on the suggestions of Sander et al. (1998), the parameter $MinPts$ for clustering instructions ($dim = 1$), would be one, which is not recommended. Therefore, we chose $\boldsymbol{MinPts = 2}$ as a suitable value. In principle, this also allows smaller clusters.

Parameter *Eps* To determine a suitable value for the parameter *Eps*, we used the heuristic described by Ester et al. (1996) that is based on the determination of the k-th nearest neighbor of an instructions. Therefore, the *k-dist* value, i.e., the distance to the k-th nearest neighbor of an instruction p_X, is calculated for all instructions. The resulting *k-dist* values of all instructions are then sorted and plotted in descending order. The resulting graph is called a *sorted k-dist graph* and gives information about the density distribution (Ester et al. 1996). Figures 3 and 4 show the *sorted k-dist graphs* for $k \in [1..6]$ based on the Jaro-Winkler distance and the Run Length distance using the example of an automotive test case specification that is considered by experts to be of high quality.

The goal is to choose the smallest possible *k-dist* value of an instruction p_X based on which the parameter *Eps* is set to *k-dist*(p_X). All instructions with a smaller or equal *k-dist* value are grouped in one cluster, therefore a smaller value should be chosen. This value is called the *threshold point* and can be identified by a *"plateau"*[1] in the *sorted k-dist graph*.

Basically, it can be seen from Figure 4 that the Run Length

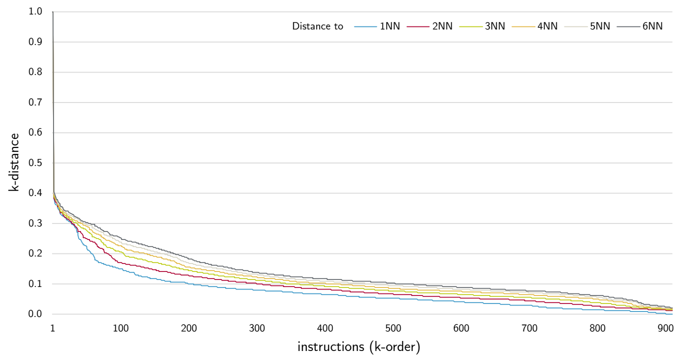

Figure 3 Sorted k-dist plots regarding Jaro-Winkler distances.

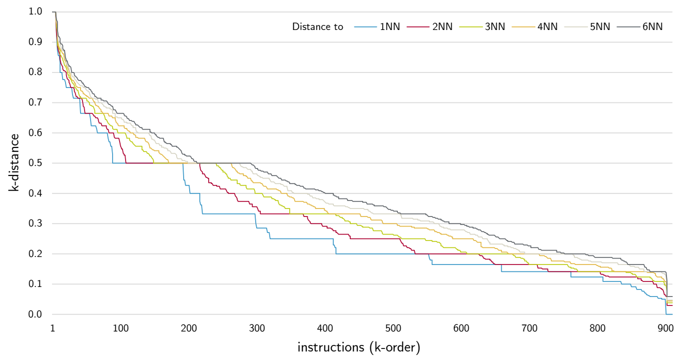

Figure 4 Sorted k-dist plots regarding Run Length distances.

distance seems to be more suitable for determining a *k-dist* value, since the graph shows more distinctive *"plateaus"*. In addition, the *"plateaus"* are much wider, i.e., there are more frequent instructions with identical *k-dist* values to their neighboring instructions. For example, the *k-dist* value 0.500 is applicable for 109 instructions or the value 0.333 is applicable for 63 instructions (cf. Figure 6):

In contrast, the *sorted k-dist graph* of the Jaro-Winkler distance (cf. Figure 3) shows a very flat curve and only a few distinctive *"plateaus"*. Some examples of potentially suitable *k-dist* values are shown in Figure 5 but they are not as clearly identifiable as in the *sorted k-dist graphs* regarding the Run Length distance. The flat curves of the *sorted k-dist graphs* shown in Figure 3 also confirm that many of the instructions are rated as relatively similar by the Jaro-Winkler distance measure, which is reflected in a lower dispersion of the *k-dist* values, which are mostly between 0.003 and 0.406. The same observation has already been made in Table 1. Based on the heuristic of Ester et al. (1996) a suitable value for the parameter *Eps* was determined using the sorted *k-dist* plots of different automotive test case specifications. After analysis of different cluster results for different *k-dist* values and both distance measures, we found the Run Length distance to be appropriate. We observed that when

[1] In the original paper, Ester et al. (1996) refer to such noticeable points in the sorted *k-dist* graph as *"valley"*. However, since the graph is sorted in descending order, there can be no *"valley"*. Hence, we prefer the term *"plateau"*.

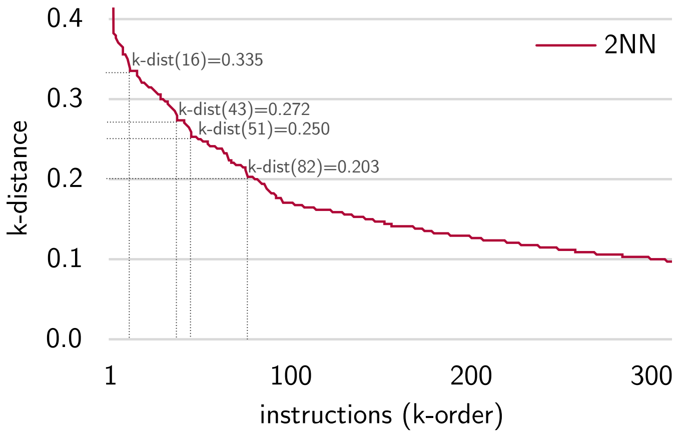

Figure 5 Zoomed extract from the sorted k-dist graphs regarding Jaro-Winkler distances.

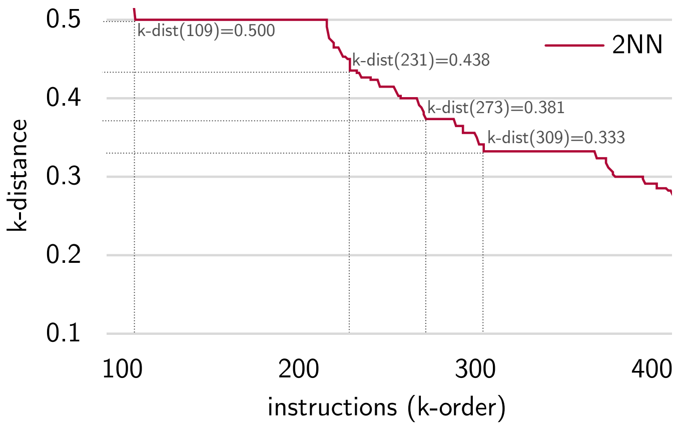

Figure 6 Zoomed extract from the sorted k-dist graphs regarding Run Length distances.

clustering is based on the Jaro-Winkler distance, similar instructions are usually distributed over several clusters, resulting in a larger number of clusters. In addition, a larger proportion of instructions remain that could not be assigned to a cluster. In contrast, using the Run Length distance for clustering leads to cluster results that are very similar to the results of manual clustering as described in Juhnke & Tichy (2019). In the end, we chose $\boldsymbol{Eps} = \mathbf{0.333}$ as a suitable value for the clustering of instructions, which is also evident from the results of the clusterings performed in the evaluation (cf. Section 4). It should be noted that for this *k-dist* value a wider *"plateau"* could also be identified in different sorted *k-dist* plots and for different test case specifications. Therefore, $Eps = 0.333$ is a suitable starting value for clustering other automotive test case specifications. However, if test case specifications deviate from the conventional way of describing test cases or if the cluster results are not satisfactory, it is advisable to repeat the determination of Eps based on the heuristic described by Ester et al. (1996).

3.2.3. Handling unclustered instructions Due to the presented approach, a large number of instructions can be clustered depending on the test case descriptions contained in a test case specification. The remaining non-clustered instructions are mainly short instructions, such as shown in Table 2.

Action Instructions	Expected Result Instructions
closing the fuel filler flap	light_switch_position = auto
closing the wiper water flap	light_switch_position = manual
closing the tank cap	light_switch_position = parking light
execute bs_adl_off	closing passive = active
execute bs_adl_on	closing passive = not active

Table 2 Examples of non-clustered instructions

Example Instructions: *"Open the door rear left"*

n-Gram Name	n	n-Grams
Unigram	1	"Open", "the", "door", "rear", "left"
Bigram	2	"Open the", "the door", "door rear", "rear left"
Trigram	3	"Open the door", "the door rear", "door rear left"
Tetragram	4	"Open the door rear", "the door rear left"
Pentagram	5	"Open the door rear left"

Table 3 Examples for n-grams

This is quite plausible, since the Run Length distance used is a token-based similarity measure, which is known to be less suitable for single words or shorter strings. In particular, this affects descriptions from expected results, which often involve parameters and value assignments (cf. Table 2).

For this reason, a subsequent clustering of remaining instructions is necessary. This clustering is performed using a developed greedy algorithm that is based on n-grams (Shannon 1948). The term n-grams is used when a instruction p is split into sub-instructions p_{sub} of length $len(p_{sub}) = n$. Examples for the decomposition of an instruction into n-grams of different lengths and their names are shown in Table 3.

Our developed algorithm initially groups all remaining instructions that have common trigrams. This is then repeated for still non-clustered instructions with digrams and monograms. Thus, even short instructions can be clustered that were not previously clustered by the DBSCAN algorithm due to a low similarity. The heuristic offers particular advantages if the remaining instructions are signal or value assignments and therefore contain the same technical terms, signal names or parameters. Thus, they are typically assigned to the same cluster.

In summary, the results of the comparison of similarity measures (cf. Tab. 1) and the insights gained from their application together with the DBSCAN algorithm (cf. Sec. 3.2.2) allow us to answer **RQ1** as follows:

The similarity between instructions can be determined automatically by using a suitable similarity measure, which optimally interacts with the clustering algorithm used. This also corresponds to the statement of Veni (2009) that different distance measures have effects on clustering algorithms. In the case of instructions from automotive test case descriptions, we have shown that a mixture of the Run Length distance developed by us and the DBSCAN algorithm with the parameters $MinPts = 2$ and $Eps = 0.333$ is suitable. In addition, the presented automation approach seems to be appropriate for automotive test case specifications, since the clustering results are similar to the results of the manual grouping. This is substantiated by the evaluation results presented in Section 4, which also allows us to answer **RQ2**.

3.3. Using the clustered instructions

After Step 1, the clustered instructions are used in Step 2 to automatically derive suggestions for conceptual templates (cf. Figure 1). These suggestions serve as a preliminary stage for the definition of a final Testing DSL for specifying test cases. Ideally, the resulting conceptual templates can be used directly for the definition of a grammar for the respective testing DSL. However, if an initial test case specification of low quality was provided with incorrect formulations or massive spelling errors (which go beyond a correction of upper and lower case as described in our minimalistic pre-processing), these deficiencies will also be reflected in the generated conceptual templates. To demonstrate how the clustered instructions can be further used to derive conceptual templates, we make use of an established approach from bioinformatics that is typically used for the analysis of structures in RNA and DNA sequences – Multiple Sequence Alignment (MSA). Multiple sequences are compared methodically and with respect to the order of the contained elements. The results are multiple alignments between the sequences, which describe sequences of editing operations that describe a transformation from one sequence to another. We used a global alignment, in which all elements of a sequence are taken into account and not only subsequences, as would be the case with a local alignment (Chao & Zhang 2009). We combine the MSA approach together with a developed heuristic to automatically derive conceptual templates for instructions that are clustered in a group. The algorithm we developed for this purpose is described below.

First, the instructions clustered in a group are converted into a string that encodes an instruction as a protein sequence. For this purpose, all words of the instructions contained in a cluster are first recorded and assigned to a unique protein coding. For example, this results in the protein sequence "I Q R K M H P N E" from the instruction *"press unlock button several times on zv-inside_button front left"*. MSA is then performed based on these protein sequences. For this, similarities between the sequences are first calculated pairwise using the Needleman-Wunsch algorithm (Needleman & Wunsch 1970). This results in a matrix which contains the calculated distances between the sequences. These are used to create a *guide tree* (Chao & Zhang 2009), also known as a phylogenetic tree. This is used to perform a progressive alignment (Feng & Doolittle 1987), the result of which is the MSA. This method for performing MSA is implemented in the program Clustal (Higgins & Sharp 1988) or ClustalW (Thompson et al. 1994), which is also part of the open source project BioJava (BioJava 2020), which we used. Table 4 shows an example of MSA of four instructions.

After that, the obtained MSA serves as a basis for the derivation of a conceptual template. For this purpose, our developed heuristic compares the contained elements column by column. If the elements of different sequences of the respective column are identical, the element is included in the template. If there are several different elements per column, such as the elements

Multiple Sequence Alignment	**Decoded Instructions**
I Q R K M H P N E – – – – –	press unlock button several times on zv-inside_button front left
I F R K M H P N E – – – – –	press lock button several times on zv-inside_button front left
I Q R K M H P N E D C G A L	press unlock button several times on zv-inside_button front left in intervals of 1 sec
I F R K M H P N E D C G A L	press lock button several times on zv-inside_button front left in intervals of 1 sec

Table 4 Example of a MSA based on instructions encoded in protein sequences

Q and *F* in Table 4, options are created in the template (e.g., *Q* | *F*). If there are gaps in one or more of the sequences indicated by "–", this element is included in the template as optional and marked with a "?". Therefore, the template shown in Figure 7 results from the example of the MSA shown in Table 4. This protein sequence (see Figure 7) is then decoded again, resulting in the conceptual template shown in Figure 8.

I (Q | F) R K M H P N E (D C G A L) ?

Figure 7 Example of a template suggestion represented as protein coding.

press (lock | unlock) button several times on zv-inside_button front left (in intervals of 1 sec) ?

Figure 8 Example of a conceptual template decoded from the protein coding in Figure 7.

Further examples of clustered instructions and the conceptual templates derived for them based on the described MSA-based approach are shown in Figure 1. The definition of the grammar of a Testing DSL (e.g. using the Xtext grammar) finally looks similar to these templates.

However, the described MSA-based approach is a heuristic, which means that it is not always possible to obtain a context-free grammar from the conceptual templates for creating an Automotive Testing DSL. Thus, not all automatically generated conceptual templates are suitable. For example, Table 5 shows three clustered instructions and the automatically determined conceptual template. Based on this, the following two instructions could also be derived using the conceptual template: *"request for interior locking from system active protection"* or *"request for interior locking from system special comfort"*. But the systems active protection or special comfort might not exist at all. In this case the conceptual template *"request for interior locking from system (parkpilot|special protection|active comfort)"* would be correct.

Nevertheless, the automatically generated conceptual templates provide insights into the structure and vocabulary used and also show which parts are variable parts (indicated by |) or optional parts (indicated by ?). This supports a language engineer in the development of a system-specific Testing DSL.

Template:	**request for interior locking from system (parkpilot | special | active) (protection | comfort)?**
Cluster:	request for interior locking from system parkpilot
	request for interior locking from system special protection
	request for interior locking from system active comfort

Table 5 Example of an insufficient conceptual template

4. Evaluation

In this section, the automated approach for grouping instructions is evaluated by comparing the results of manual grouping (G) with those of automated clustering (C). Therefore, we examine to what extent manually grouped instructions are also clustered by the DBSCAN algorithm. Furthermore, we investigated in how many clusters the instructions are distributed by the automated approach compared to the manual approach.

4.1. Study Design & Data Collection

Instructions are the basis for manual grouping and automated clustering. To obtain instructions, action and expected result descriptions must first be extracted from test case specifications, split, rectified, and, if necessary, transformed (cf. Juhnke & Tichy 2019). In this evaluation, the same instructions from action and expected result descriptions are used as input for manual grouping and automated clustering. Then, we determine for each instruction from the manual grouping to which automatically created cluster it is assigned. This is shown graphically using *sankey diagrams*. These diagrams visualize which groups respectively clusters exist and which instructions from a group are contained in a certain cluster. In addition, we use *Cohen's Kappa* κ (Cohen 1960) as measure for the agreement between the result of manual grouping and automated clustering. For the interpretation of *Cohen's Kappa*, and thus for the description of the relative strength of agreement, we use the nomenclature of Landis & Koch (1977), where, for example, $0.81 \leq \kappa \leq 1.00$ stands for an almost perfect, $0.61 \leq \kappa \leq 0.80$ for a substantial, $0.41 \leq \kappa \leq 0.60$ for a moderate, $0.21 \leq \kappa \leq 0.40$ for a fair, $0.00 \leq \kappa \leq 0.20$ for a slight, and $\kappa < 0.00$ for a poor agreement.

For the evaluation, we selected three real automotive test case specifications that have already been reviewed by domain experts who assessed them as being of high quality and contained established formulations. For these test case specifications, instructions have already been manually extracted and a manual grouping has been performed. These groups serve as a reference for comparison with the results of our developed clustering approach. The selected test case specifications are of different sizes, i.e., test case specification E1 is a small, E2 is a medium and E3 is a large one (see Tab. 6).

4.2. Results

Figures 9 to 14 illustrate the results of the comparison using *sankey diagrams*, which are discussed below. For each test case specification (TestSpec E1 – E3) there is a *sankey diagram* for the instructions from the action (A) and one for those from the expected result (ER) descriptions. On the left-hand side are the groups (G), whih represent the result of the manual grouping. On the right-hand side are the groups resulting from the automated clustering (C). The number of instructions contained in the groups (G and C) is indicated in parentheses.

Test Case Specifications	E1	E2	E3
# test cases	46	203	750
# test steps	127	353	2845
# action instructions	39	86	131
# expected result instructions	54	77	105

Table 6 Overview of the evaluated test case specifications

For **Test Case Specification E1**, Figure 9 shows for action descriptions that one instruction from group G_A1 and one from G_A5 are not clustered by the automated approach. The instruction *"Outdoor temperature* $< 5°C$*"* was manually grouped together with instructions of the form *"Light switch position = AUTO"* in group G_A1, since both instructions are considered as a kind of parameter and value assignment. In contrast, the instruction for temperature indication was not clustered automatically, as there are no matching tokens with light switch instructions.

Furthermore, the three instructions from group G_A5 have different lengths. The two instructions clustered in cluster C_A8 contain six and eight words. The third instruction from group G_A5, which was not clustered by the DBSCAN algorithm, is only three words long, with two words matching the longer instructions. Therefore, these three instructions from group G_A5 are not clustered in the same cluster. For instructions used in action descriptions of Test Case Specification E1, *Cohen's Kappa* was calculated[2] with $\kappa = 0.932$, which represents an almost perfect agreement between groups (G) and clusters (C).

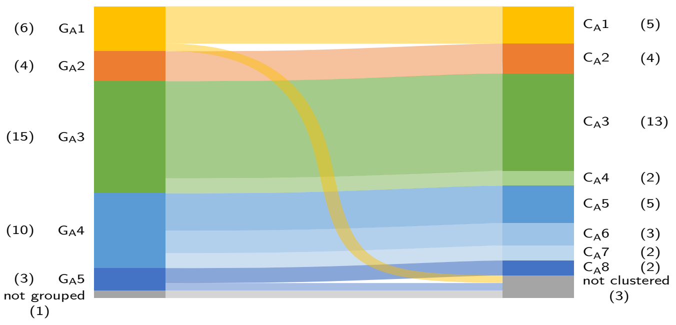

Figure 9 Comparison of the results of manual grouping (G) and automated clustering (C) for TestSpec E1 (A).

For expected result descriptions, only one instruction from group $G_{ER}1$ that is also used to indicate the outside temperature is not clustered (see Figure 10). All other instructions were clustered and were also clustered in similar clusters. The only thing that stands out is that the clusters are more fine-grained

[2] Several clusters resulting from a group and thus representing a more fine-grained division of this group were combined, so that the calculation of *Cohen's Kappa* is based on a *nxn* confidence matrix, where *n* corresponds to the number of groups including the group of not grouped instructions. This procedure was used to calculate *Cohen's Kappa* for TestSpecs E1 to E3.

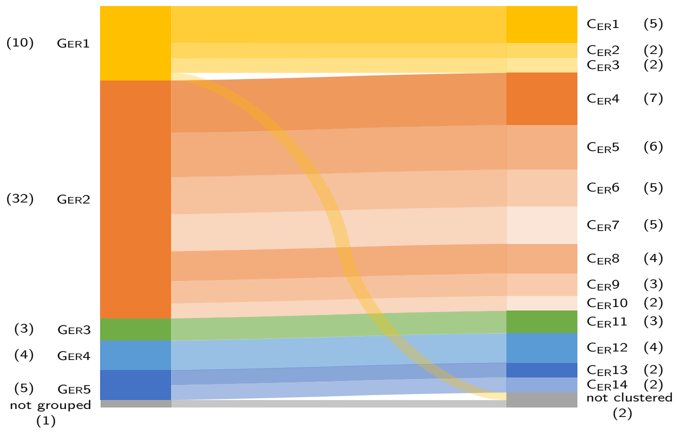

Figure 10 Comparison of the results of manual grouping (G) and automated clustering (C) for TestSpec E1 (ER).

and therefore there are more clusters than groups. In this case this may indicate poor manual grouping. For example, group $G_{ER}2$ contains instructions describing the state of an object, such as *"parking light is active"* or *"rear blind remains open"*. The clustering into several clusters ($C_{ER}4 - C_{ER}10$) is done according to the objects, such as *"high beam"*, *"sunroof"*, *"windscreen wiper"*, *"ambient lights"*, *"trunk lid"*, or *"exterior mirrors"*. For instructions used in expected result descriptions of Test Case Specification E1, *Cohen's Kappa* was calculated with $\kappa = 0.969$, which represents an almost perfect agreement between groups (G) and clusters (C).

Figure 11 shows in the action descriptions of **Test Case Specification E2** three manually created groups (G_A2, G_A3, and G_A4) which group parameters, such as *"isw_stat=ign_acc"*, *"set isw_stat=ign_off"*, *"rdlgt_fl_rq_st3 = 0%"*, or *"[icf_modelparam_write][pilc_ttimeout] = 10s"*. The latter type of parameters form the group G_A3. The parameter names were considered in the evaluation in their original form and not replaced by placeholders (cf. Section 3.1). This also explains the more fine-grained partitioning of the parameter instructions from groups G_A2 and G_A3 into clusters C_A3 to C_A7. Replacing the parameter names with placeholders would result in fewer clusters. From the group of not grouped instructions one instruction was clustered in cluster C_A17 together with instructions from group G_A13 due to a matching word by the subsequent greedy algorithm (see algorithm description on Page 8). Overall, the cluster result is very similar to the manual grouping result. This is also confirmed by *Cohen's Kappa* with $\kappa = 0.842$, which represents an almost perfect match.

In the expected result descriptions in Figure 12, the instructions from groups $G_{ER}1$ to $G_{ER}3$ are similar to the parameter instructions in action descriptions. This also explains the clustering of four instructions from group $G_{ER}1$ and the two instructions from group $G_{ER}2$ into cluster $C_{ER}3$. Group $G_{ER}6$ has an instruction that is twice as long (eight words) as the instructions otherwise contained (three to four words). This instruction is assigned to cluster $C_{ER}13$ by the subsequent clustering. The other instruction contains only three words, and only one word has a

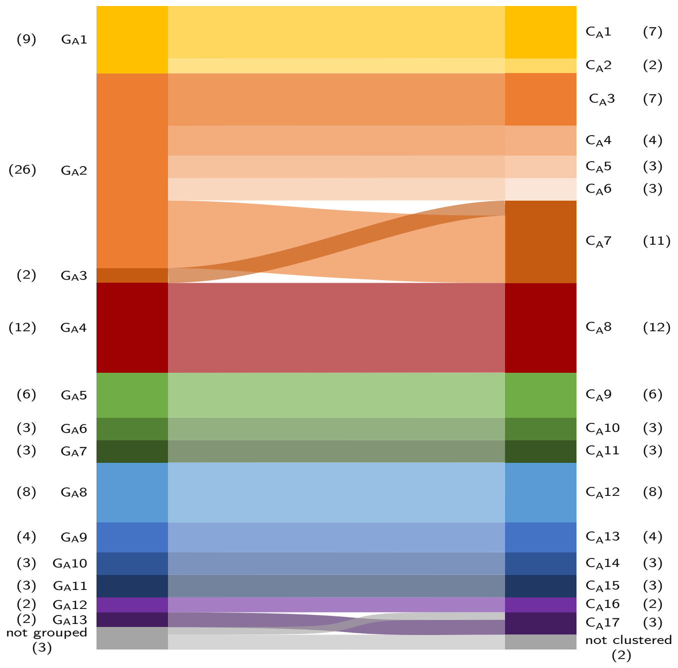

Figure 11 Comparison of the results of manual grouping (G) and automated clustering (C) for TestSpec E2 (A).

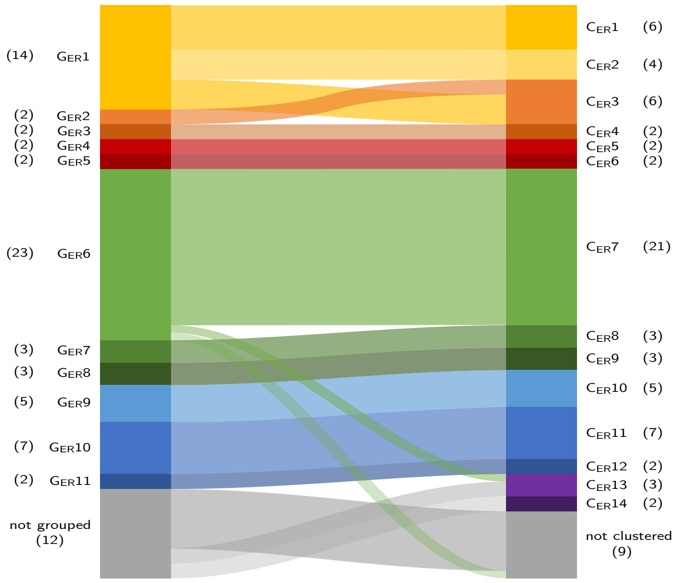

Figure 12 Comparison of the results of manual grouping (G) and automated clustering (C) for TestSpec E2 (ER).

match with other instructions. Therefore this instruction is not clustered. In addition, it is noticeable that the greedy algorithm causes previously not clustered instructions to form new clusters ($C_{ER}13$ and $C_{ER}14$). Overall, the calculated *Cohen's Kappa* with $\kappa = 0.908$ shows an almost perfect agreement between groups (G) and clusters (C).

The third considered **Test Case Specification E3** shows a few more differences between manual grouping and automated clustering in the action descriptions (cf. Figure 13). Two instructions from group G_A8 with a length of seven words were clustered in cluster C_A9, since these seven words are completely contained in an instruction from G_A9. The instructions from G_A8 to G_A10 are very similar, since they describe the operation of a handheld transmitter. They differ only in the beginning, which describes a temporal context, such as *"exactly 20 seconds after the signal is sent ..."* or *"within 60 seconds of pressing the key ..."*. Hence, they are clustered together into cluster C_A9. Group G_A12 contains instructions like *"send standby request"* or *"send wake-up request"*. These two instructions are assigned the previously non-grouped instruction *"send 3 times different security byte"*, which results in cluster C_A13. This can be explained by the subsequent greedy algorithm due to the matching of the word *"send"*. Since there is a match of two words between the third instruction from group G_A12 and an instruction from group G_A13, they form the new cluster C_A14.

The combination of the instructions from groups G_A15, G_A16 and G_A17 (cf. Figure 13) can be explained by the fact that the instructions of the form *"press button 1 for 1 second"* from group G_A15 are completely contained in the instructions from groups G_A16 and G_A17, e.g., *"during engine start press button 1 for 1 second"*. This is an example that manual grouping can also be carried out very fine-granularly. The instruction *"press programming key on garage door operator"* from group G_A15 is not clustered by the automated approach because it does not match the instructions of the form *"press button 2 for 1 second"*. The subsequent greedy algorithm also causes a clustering of previously not clustered instructions, which is mainly due to the combination of instructions that have only one word in common. Altogether, Test Case Specification E3 has 131 instructions for action descriptions, which makes it a larger test case specification. The *sankey diagram* (cf. Figure 13) shows some more differences between manual grouping and automated clustering. Nevertheless, the calculated *Cohen's Kappa* with $\kappa = 0.848$ indicates an almost perfect agreement.

The clustering results for expected result descriptions are quite similar to the results of manual grouping, as shown in Figure 14. The group $G_{ER}7$ contains instructions of the form *"gdo sends on the lin gdo_stuckedbutton_flt_st3 = 1 = fault"*. The group $G_{ER}18$ contains instructions describing a waiting time and a subsequent reaction. The instruction *"after 180 seconds gdo sends on the lin gdo_stuckedbutton_flt_st3 = 1 = fault"* from $G_{ER}18$ is clustered together with instructions from group $G_{ER}7$ into cluster $C_{ER}12$, since 10 of 13 tokens match. This also applies to instructions from groups $G_{ER}8$ and $G_{ER}9$, which are similar to instructions from group $G_{ER}7$. Also for the expected result descriptions, the calculated *Cohen's Kappa* with $\kappa = 0.946$ shows that there is an almost perfect agreement between groups (G) and clusters (C).

Altogether, it is noticeable in all three test case specifications that the instructions are clustered by the automated approach in a partly finer granular way. This is due to the fact that the similarity comparison between instructions in the automated approach is based exclusively on a syntactic comparison. In contrast, semantic similarities were partially taken into account for the manually performed groupings. A variation of the parameter *Eps* of the DBSCAN algorithm, such as $Eps < 0.333$, leads to larger clusters, but also to a mixture of instructions from different groups, which are then combined in a cluster. As a result, instructions that have less in common are also combined. For this reason, a fine-granular result is preferred. In summary, this allows us to answer **RQ2** as follows:

The evaluation shows that automated clustering provides useful results and is comparable to manual grouping. This is particularly clear from the *sankey diagrams* in Figures 9 to 14. In addition, the calculated values for *Cohen's Kappa* are always $\kappa > 0.81$ and thus, according to Landis & Koch (1977), show an almost perfect agreement between groups (G) and clusters (C) for all considered test case specifications.

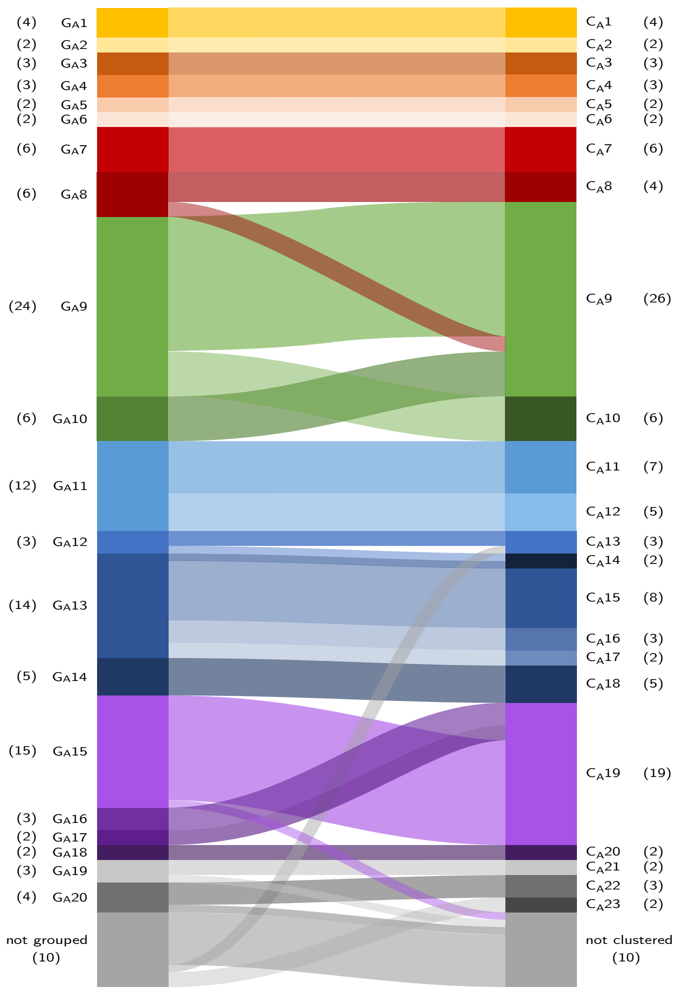

Figure 13 Comparison of the results of manual grouping (G) and automated clustering (C) for TestSpec E3 (A).

5. Conclusion and future work

In this paper we presented an automated clustering approach to support the domain analysis of automotive test case specifications. Therefore, we used the DBSCAN algorithm and developed a similarity measure called Run Length similarity. Furthermore, we showed how to appropriately parameterize the DBSCAN algorithm for the application to automotive test case specifications ($MinPts = 2$, $Eps = 0.333$). In addition, we evaluated our automated clustering approach by showing for three real test case specifications that there is an almost perfect alignment between the result of the automated clustering and

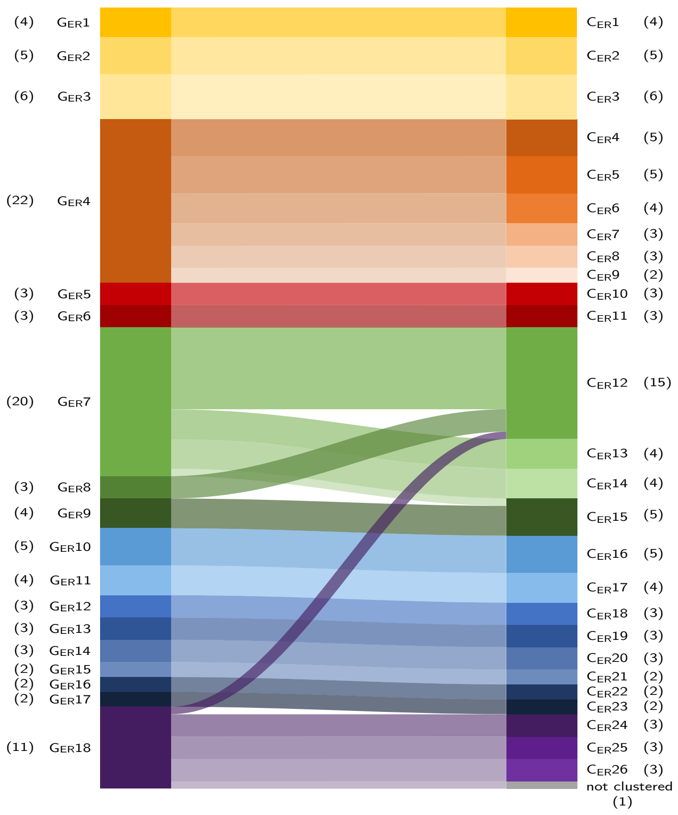

Figure 14 Comparison of the results of manual grouping (G) and automated clustering (C) for TestSpec E3 (ER).

the groups created manually ($\kappa > 0.81$). The advantage of automated clustering is that the results are person-independent and thus easier to reproduce, which is not necessarily the case with manual created groups. Since the instructions used in actions and expected result descriptions are uniquely assigned to a cluster and outliers are allowed, so that an inappropriate assignment of instructions to a cluster does not necessarily occur, our automated approach provides basis for deriving conceptual templates. A special feature of our automation approach is its application to unknown and unlabeled data, i.e., no knowledge of the test case specifications used or special pre-processing is required, which would require, for example, domain-specific dictionaries or a preliminary analysis by domain experts.

In future work, we intend to apply our clustering approach to additional test case specifications (e.g., of varying quality and maturity) to further investigate the suitability of the resulting clusters for deriving DSLs. In this respect, and complementary to the MSA-based approach used, we want to investigate how the input is suitable for other grammatical inference algorithms to derive grammars. In addition, since our approach works well on reviewed test case specifications even without special pre-processing, we also want to identify possible approaches for more extensive pre-processing that would allow our approach to be applied to test case specifications with a lower level of maturity. Furthermore, we plan to investigate how well the presented approach and particularly the clustering and similarity measures apply to requirements documents for automatically inferring requirement boilerplates.

References

Aggarwal, C. C., Hinneburg, A., & Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In *Proc. of ICDT'01* (pp. 420–434). Springer. doi: 10.1007/3-540-44503-X_27

Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms*. Springer. doi: 10.1007/978-1-4757-0450-1

BioJava: The open-source java library for bioinformatics. (2020). Retrieved from http://biojava.org/index.html

Bryant, B. R., Mernik, M., Hrnčič, D., Javed, F., Liu, Q., & Sprague, A. (2010). Grammar inference technology applications in software engineering. In *Grammatical inference: Theoretical results and applications (ICGI'10)* (pp. 276–279). Springer. doi: 10.1007/978-3-642-15488-1_25

Chao, K.-M., & Zhang, L. (2009). *Sequence comparison: Theory and methods*. London: Springer. doi: 10.1007/978-1-84800-320-0

Cohen, J. (1960). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37–46. doi: 10.1177/001316446002000104

Eldesoky, A., Saleh, M., & Sakr, N. (2009). Novel similarity measure for document clustering based on topic phrases. In *Proc. of ICMN'09* (pp. 92–96). IEEE. doi: 10.1109/ICNM.2009.4907196

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD'96* (pp. 226–231). AAAI Press. Retrieved from http://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

Feng, D. F., & Doolittle, R. F. (1987). Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, *25*(4), 351–360. doi: 10.1007/bf02603120

Hauptmann, B., Junker, M., Eder, S., Heinemann, L., Vaas, R., & Braun, P. (2013). Hunting for smells in natural language tests. In *Proc. of ICSE'13* (pp. 1217–1220). IEEE. doi: 10.1109/ICSE.2013.6606682

Higgins, D. G., & Sharp, P. M. (1988). Clustal: A package for performing multiple sequence alignment on a microcomputer. *Gene*, *73*(1), 237–244. doi: 10.1016/0378-1119(88)90330-7

Huang, A. (2008). Similarity measures for text document clustering. In *Proc. of the 6th new zealand computer science research student conference (NZCSRSC'08)* (pp. 49–56).

Jaccard, P. (1912). The distribution of the flora in the alpine zone. *New Phytologist*, *11*(2), 37–50. doi: 10.1111/j.1469-8137.1912.tb05611.x

Javed, F., Mernik, M., Bryant, B. R., & Sprague, A. (2008). An unsupervised incremental learning algorithm for domain-specific language development. *Applied Artificial Intelligence*, *22*(7-8), 707–729. doi: 10.1080/08839510802164127

Joy, M., & Luck, M. (1999). Plagiarism in programming assignments. *IEEE Transactions on Education*, *42*(2), 129–133. doi: 10.1109/13.762946

Juhnke, K., & Tichy, M. (2019). A tailored domain analysis method for developing system-specific testing DSLs enabling their smooth introduction in automotive practice. In *Proc. of SEAA'19* (pp. 10–18). IEEE. doi: 10.1109/SEAA.2019.00011

Juhnke, K., Tichy, M., & Houdek, F. (2018). Challenges concerning test case specifications in automotive software testing. In *Proc. of SEAA'18* (pp. 33–40). IEEE. doi: 10.1109/SEAA.2018.00015

Karypis, M. S. G., Kumar, V., & Steinbach, M. (2000). A comparison of document clustering techniques. In *Workshop on text mining at KDD 2000 (may 2000).*

Lachmann, R., & Schaefer, I. (2014). Towards efficient and effective testing in automotive software development. In *Informatik 2014.* (pp. 2181–2192). Gesellschaft für Informatik e.V. Retrieved from http://dl.gi.de/handle/20.500.12116/2847

Landis, J. R., & Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, *33*(1), 159–174. doi: 10.2307/2529310

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents. In *Proc. of ICML'14* (pp. 1188–1196). JMLR.org.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, *10*(8), 707–710. Note: Translated from Doklady Akademii Nauk SSSR Vol. 163, No. 4, pp. 845–848, 1965.

Li, L., & Li, H. (2017). A novel document distance based on concept vector space. In *Proc. of the 17th international conference on communication technology (ICCT'17)* (pp. 2014–2017). IEEE. doi: 10.1109/ICCT.2017.8359982

Li, M., & Vitanyi, P. M. (2008). *An introduction to kolmogorov complexity and its applications* (3rd ed.). Springer. doi: 10.1007/978-0-387-49820-1

Lyon, C., Barrett, R., & Malcolm, J. (2004). A theoretical basis to the automated detection of copying between texts, and its practical implementation in the ferret plagiarism and collusion detector. *Plagiarism: Prevention, Practice and Policies*, 1–7. Retrieved from http://hdl.handle.net/2299/2114

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proc. of the 5th berkeley symposium on mathematical statistics and probability* (Vol. 1, pp. 281–297). University of California Press.

Maimon, O., & Rokach, L. (2005). *Data mining and knowledge discovery handbook*. Springer. doi: 10.1007/978-0-387-09823-4

Mernik, M., Hrnčič, D., Bryant, B. R., Sprague, A. P., Gray, J., Liu, Q., & Javed, F. (2009). Grammar inference algorithms and applications in software engineering. In *Proc. of ICAT'09* (p. 1-7). doi: 10.1109/ICAT.2009.5348441

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Morichetta, A., Bocchi, E., Metwalley, H., & Mellia, M. (2016). Clue: Clustering for mining web urls. In *Proc. of ITC 28* (pp. 286–294). IEEE.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453. doi: 10.1016/0022-2836(70)90057-4

Nevill-Manning, C. G., & Witten, I. H. (1997). Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, *7*, 67–82. doi: 10.1613/jair.374

Rajaraman, A., & Ullman, J. D. (2011). *Mining of massive datasets*. Cambridge University Press.

Ratcliff, J. W., & Metzener, D. E. (1988). Pattern matching: The gestalt approach. *Dr. Dobb's Journal*, *13*(7), 46.

Sander, J., Ester, M., Kriegel, H.-P., & Xu, X. (1998). Density-based clustering in spatial databases: The algorithm gdbscan and its applications. *Data Mining and Knowledge Discovery*, *2*, 169–194. doi: 10.1023/A:1009745219419

Sapkota, U., Bryant, B. R., & Sprague, A. (2012). Unsupervised grammar inference using the minimum description length principle. In *Proc. of MLDM'12* (pp. 141–153). Springer. doi: 10.1007/978-3-642-31537-4_12

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423. doi: 10.1002/j.1538-7305.1948.tb01338.x

Stevenson, A., & Cordy, J. R. (2013). Grammatical inference in software engineering: An overview of the state of the art. In *Software language engineering (SLE'12)* (pp. 204–223). Springer. doi: 10.1007/978-3-642-36089-3_12

Strehl, A., Ghosh, J., & Mooney, R. (2000). Impact of similarity measures on web-page clustering. In *Workshop on artificial intelligence for web search (AAAI'00)* (pp. 58–64).

Taghva, K., & Veni, R. (2010). Effects of similarity metrics on document clustering. In *Proc. of ITNG'10* (pp. 222–226). IEEE. doi: 10.1109/ITNG.2010.65

Thompson, J. D., Higgins, D. G., & Gibson, T. J. (1994). Clustal W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, *22*(22), 4673–4680. doi: 10.1093/nar/22.22.4673

Ukkonen, E. (1992). Approximate string-matching with q-grams and maximal matches. *Theoretical Computer Science*, *92*(1), 191–211. doi: 10.1016/0304-3975(92)90143-4

University of Warwick. (2014). *Sherlock: Plagiarism detection software.* Department of Computer Science. Retrieved from http://warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock

Veni, R. (2009). *Effects of similarity metrics on document clustering* (PhD Thesis, University of Nevada, Las Vegas). Retrieved from http://digitalscholarship.unlv.edu/thesesdissertations/71

White, D. R., & Joy, M. S. (2004). Sentence-based natural language plagiarism detection. *ACM Journal of Educational Resources in Computing*, *4*(4), 1–20. doi: 10.1145/1086339.1086341

Winkler, W. E. (1990). String comparator metrics and enhanced decision rules in the fellegi-sunter model of record linkage. *Proc. of the Section on Survey Research Methods*, 354–359. Retrieved from http://eric.ed.gov/?id=ED325505

Xu, R., & Wunsch, D. (2005). Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, *16*(3), 645–678. doi: 10.1109/TNN.2005.845141