Using Clustering to Improve the Structure of Natural Language Requirements Documents

Alessio Ferrari[1], Stefania Gnesi[1], and Gabriele Tolomei[2]

[1] ISTI-CNR, Pisa, Italy
{alessio.ferrari,stefania.gnesi}@isti.cnr.it
[2] DAIS, Università Ca' Foscari Venezia, Italy
gabriele.tolomei@unive.it

Abstract. [**Context and motivation**] System requirements are normally provided in the form of natural language documents. Such documents need to be properly structured, in order to ease the overall uptake of the requirements by the readers of the document. A structure that allows a proper understanding of a requirements document shall satisfy two main quality attributes: (i) *requirements relatedness*: each requirement is conceptually connected with the requirements in the same section; (ii) *sections independence*: each section is conceptually separated from the others. [**Question/Problem**] Automatically identifying the parts of the document that lack requirements relatedness and sections independence may help improve the document structure. [**Principal idea/results**] To this end, we define a novel clustering algorithm named Sliding Head-Tail Component (S-HTC). The algorithm groups together similar requirements that are contiguous in the requirements document. We claim that such algorithm allows discovering the structure of the document in the way it is perceived by the reader. If the structure originally provided by the document does not match the structure discovered by the algorithm, hints are given to identify the parts of the document that lack requirements relatedness and sections independence. [**Contribution**] We evaluate the effectiveness of the algorithm with a pilot test on a requirements standard of the railway domain (583 requirements).

Keywords: Requirements analysis, requirements documents structure, requirements quality, similarity-based clustering, lexical clustering.

1 Introduction

The quality of a natural language requirements document strongly depends on its *structure* [16,10]. A proper document structuring enables a better *understanding* of the requirements, and eases the modifiability of the overall requirements specification [20]. Even though many templates and recommendations are available to guide the structuring of a requirements document [10,14,3], few automatic approaches exist to evaluate the quality of the document structure *after* the document has been written. The current work presents a novel automated analysis method along this little-explored research path.

J. Doerr and A.L. Opdahl (Eds.): REFSQ 2013, LNCS 7830, pp. 34–49, 2013.

Several factors affect the structure of a requirements document. The length of the sections, the number of nested paragraphs in each section, the clarity of the titles and the number of cross-references are only a limited list of those structural aspects that can make a requirements document completely clear or completely unintelligible. In this paper, we focus on two structural quality attributes that we consider relevant to facilitate the understanding of a requirements document, namely the *requirements relatedness* and the *sections independence*.

A requirements document provides proper requirements relatedness if each requirement is conceptually connected with the previous and following requirements of the same section. Furthermore, a requirements document provides proper sections independence when each section is conceptually separated from the others.

In principle, quantitative indexes of the two quality attributes defined above may be derived from a similarity-based lexical analysis of the document. Requirements relatedness could be mapped to an index of *cohesion* among requirements, while sections independence could be mapped to an index of *separation* among sections [18]. However, we argue that such indexes might be hardly usable to improve the structure of the document. Indeed, once we know that a requirements document has a low value of cohesion or separation – even assuming that we are able to define what is a low/high value for these indexes – we do not have any guidance to actually enhance the structure of such document.

Therefore, in our approach, we do not associate quantitative indexes to the quality attributes under analysis. Instead, we define a clustering-based approach that identifies the structure of the document in the way it might be perceived by the reader. Intuitively, if a lexical change occurs in the requirements document, the reader assumes that the discussed concepts are different, and she expects a new section. The document structure is supposed to reflect such conceptual changes.

The clustering algorithm employed in our approach, named Sliding Head-Tail Component (S-HTC), detects the conceptual changes in the requirements, and identifies the *hidden structure* of the document in terms of such changes. In the hidden structure, requirements are grouped according to their lexical relatedness, and sections are partitioned according to their lexical independence. Therefore, requirements relatedness and sections independence for the original document are evaluated in comparison to the hidden structure discovered by the algorithm. If the structure of the original document does not adhere to the hidden structure, hints are given to re-arrange the requirements document.

We present the results of a pilot test on the functional requirements document of the EIRENE (European Integrated Railway radio Enhanced NEtwork) system, a digital radio standard for railways [19] (583 requirements in total).

The paper is structured as follows. In Sect. 2, we present the overview of the method. In Sect. 3, we describe the approach in detail. In Sect. 4, the pilot test is discussed. In Sect. 5, we summarize the most relevant related work. In Sect. 6, we draw conclusions and we discuss the possible evolution of the approach.

2 Overview of the Method

An overview of the method is presented in Fig.1. The method is based on a clustering algorithm, named Sliding Head-Tail Component (S-HTC), which is described in Sect. 3. The input of the algorithm is the ordered list of the requirements (**Requirements List**) in the **Original Document**, without any information about the document structure (i.e., no sections, no indexes).

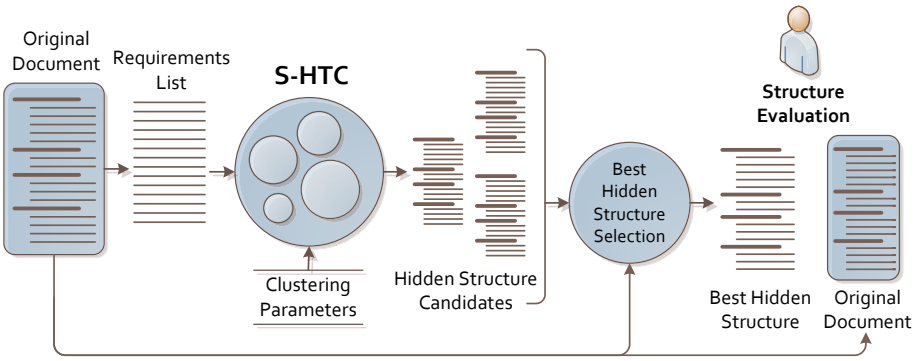

Fig. 1. Method Overview

The **S-HTC** algorithm is able to identify possible *hidden structures* of the requirements document. A hidden structure is a partitioning of the document into possible sections of requirements. The partitioning is performed according to the similarity among neighbouring requirements in the document. The algorithm produces hidden structures where requirements are grouped according to their lexical relatedness, and sections are partitioned according to their lexical independence.

The algorithm is executed several times, varying different clustering parameters that influence its behaviour. The output of each execution is a **Hidden Structure Candidate**. Intuitively, a hidden structure candidate represents the structure of the document perceived by a possible reader, who reads the document without having the original structure information (i.e., the document is made of a single, long sequence of requirements without sections).

Among the hidden structure candidates, we select the one that more closely matches the structure originally provided by the document (**Best Hidden Structure Selection**). We call this structure candidate the **Best Hidden Structure**. Intuitively, the best hidden structure is the structure of the document perceived by a reader who is provided with the original structure information. The assumption here is that the original structure is reasonable, but might require some improvements.

To this end, we finally observe the differences between the best hidden structure and the original structure (**Structure Evaluation**). The detected differences allows identifying the defects in the original structure related to: (1) requirements that are not sufficiently related with the other requirements of the same section; (2) sections that are not sufficiently independent from the others.

3 Requirements Clustering

The core of our method is the Sliding Head-Tail Component (S-HTC) clustering algorithm. The algorithm is a variant of the Head-Tail Component (HTC) algorithm defined by the third author for retrieving groups of task-related queries in Web-search engine query logs [12].

The S-HTC algorithm groups together those requirements that are lexically related, and that are contiguous in the document. The goal of this clustering approach is emulating the process of reading a requirements document: if two contiguous requirements are lexically related, they are likely to speak about the same content. Therefore, we assume that the reader is likely to interpret them as part of the same conceptual unit of the document. If there is a change in the lexical content (i.e., the reader reads a requirement that is not lexically related to the previous ones), the requirement is likely to speak about other concepts, and is likely to be part of another conceptual unit.

The S-HTC algorithm is designed to identify such conceptual units. From the ordered list of requirements in a document, the algorithm derives a *sequence* of clusters (i.e., ordered groups of requirements), each one representing a conceptual unit of the document. We call this sequence of clusters the *hidden structure* of the document. More formally, given the list of requirements in a document, i.e., $D = \langle R_1, R_2, \ldots, R_m \rangle$, the hidden structure is a ordered partitioning of D, i.e., $\pi(D)$, into a non-empty disjoint sequence of clusters C^i that completely cover D, namely $\pi(D) = \langle C^1, C^2, \ldots, C^{|\pi(D)|} \rangle$ [1].

3.1 Requirements Representation

In order for the requirements to be processed by the clustering algorithm, a proper representation is required that describes the lexical content of the requirement. To this end, we choose a representation of a requirement R_j that takes into account the set of lexical terms, which it is composed of.

In order to focus only on the *relevant* lexical aspects of a requirement, it is recommended to remove from its representation all those terms that are common in the language, which the requirements are written in. These common terms, such as articles and pronouns, are known as *stop-words*.

Furthermore, it is useful to reduce the remaining terms to their morphological root (e.g., the terms "equipment" and "equipped" are both reduced to "equip"). This procedure is called *stemming*.

[1] It should be $|\pi(D)| \ll m$ in order for the partitioning to be effective.

More formally, let $T = \{t_1, t_2, \ldots, t_n\}$ be the set of unique terms in D, i.e., the *vocabulary* of terms. Therefore, R_j may be represented as a subset of distinct terms, i.e., $\hat{R}_j \subseteq T$. Let $\overline{T} = T \setminus S$, where S is the set of stop-words. Moreover, let M be the set morphological roots, and let $\phi : T \to M$ be the stemming function that maps each term into its morphological root. Therefore, R_j may be represented as a set of distinct morphological roots of those terms that are not stop-words, i.e., $R'_j = \{\phi(t_j) \mid t_j \in \hat{R}_j \cap \overline{T}\}$.

Consider, for example, the requirement 5.4.6 extracted from the EIRENE dataset [19]: $R =$ *"The driver shall be able to adjust the contrast of the display"*. Its lexical representation is the set $R' = \{$driv, abl, adjust, contrast, display$\}$.

3.2 Similarity Metrics

In order to establish the degree of relationship among requirements, the clustering algorithm requires the definition of a similarity metric. Since requirements are natural language sentences, typical string similarity metrics can be used to establish the degree of relationship between pairs of requirements [1].

If we consider the lexical representation described in Sect. 3.1, an immediate way to compute the text similarity between two requirements R_i, R_j is to consider their corresponding term-set representations, i.e., R'_i and R'_j, respectively. An interesting measure is given by the *Jaccard index*, also known as *Jaccard similarity metric*, which is defined as follows:

$$\sigma_{jac}(R_i, R_j) = \frac{|R'_i \cap R'_j|}{|R'_i \cup R'_j|}. \tag{1}$$

This similarity metric measures the proportion of terms that two text strings have in common. It assumes that the greater is the number of shared terms the higher is the chance of the two strings to be similar.

Another popular lexical-based similarity metric is the *edit distance*, which counts the number of edit operations required to transform one text string into another. There are several different ways to define an edit distance, depending on which edit operations are allowed, e.g., *replace*, *delete*, *insert*, *transpose*, etc. In this work, we resort to using one of the most common metric for measuring the edit distance between two strings, namely the *Levenshtein distance* [11], which we properly transform in a similarity metric σ_{lev} normalized between 0 and 1. The Levenshtein distance allows capturing the character-level similarity among the sentences, while the Jaccard metric captures the term-level similarity.

Finally, we consider another similarity metric, which is obtained by a convex combination between σ_{jac} and σ_{lev} as follows:

$$\sigma_{jac-lev}(R_i, R_j) = \alpha \cdot \sigma_{jac}(R_i, R_j) + \beta \cdot \sigma_{lev}(R_i, R_j). \tag{2}$$

In our experiments, which are described in Section 4, we set $\alpha = \beta = 0.5$. This choice equally balances the impact of the two metrics, namely $\sigma_{jac}(R_i, R_j)$ and $\sigma_{lev}(R_i, R_j)$, in the computation of the final measure.

3.3 S-HTC Clustering Algorithm

The S-HTC algorithm comprises two steps. In the first step a preliminary list of clusters is built. In the second step, similar clusters are merged to create the hidden structure.

First Step: Preliminary List of Clusters. The first step aims at creating an approximate fine-grained clustering of the given requirement document D, represented as a *sequence*, i.e., $D = \langle R_1, R_2, \ldots, R_m \rangle$. S-HTC exploits the sequentiality of contract-style requirement lists, and tries to detect sequential clusters of requirements, denoted with $\tilde{C}^i$, namely groups of requirements that occur in a row in the document. Each requirement in the same sequential cluster is similar enough to the immediate next one. The behaviour of the algorithm relies on a threshold parameter τ, which defines how much similar two requirements are supposed to be, in order to be placed in the same cluster.

Each requirement is compared to the following one through a similarity function $\sigma : D \times D \longmapsto [0, 1]$. The similarity function can be computed in terms of σ_{jac}, σ_{lev}, or $\sigma_{jac-lev}$.

The first preliminary cluster $\tilde{C}^1$ is initialized with the requirement R_1. If $\sigma(R_1, R_2) > \tau$, R_2 is added to the $\tilde{C}^1$ cluster. Otherwise, a new cluster $\tilde{C}^2$ is created, and R_2 is added to this cluster. Then, R_2 is compared with R_3. If $\sigma(R_2, R_3) > \tau$, R_3 is added to the cluster where R_2 resides, otherwise a new cluster is created. The algorithm iterates this procedure until no more requirements are left.

At the end of the first step we have a sequence of preliminary clusters, namely $\tilde{\pi}(D) = \langle \tilde{C}^1 \ldots \tilde{C}^{|\tilde{\pi}(D)|} \rangle$.

Second Step: Hidden Structure. In this step we build the possible conceptual units of the document as they might be perceived by the reader, i.e., we identify a possible *hidden structure* of the document. To this end, we merge together related preliminary clusters.

The rationale of this step is as follows. In a requirements document, we might have a requirement that, though not being similar to the exact previous one, is similar to one of the requirements occurring slightly before in the document. For example, take the case of the three following requirements for a generic radio communication system for trains (adtapted from [19]):

- R_1 : *If the driver receives an incoming call, the system shall enable the loudspeaker.*
- R_2 : *If the driver receives an incoming call, the system shall visualize the identity of the caller through the driver visual interface.*
- R_3 : *If the loudspeaker is enabled, an audible indication shall be provided to the driver.*

A reader would naturally interpret these requirements as part of the same conceptual unit. We notice that R_1 and R_2 are lexically similar. Hence, after the first step of the algorithm, we expect to have a cluster $\tilde{C}^1 = \langle R_1, R_2 \rangle$. On the

other hand, R_2 is not lexically similar to R_3. Therefore, we would have a separate cluster for R_3, i.e., a cluster $\tilde{C}^2 = \langle R_3 \rangle$. Nevertheless, we notice that R_3 is similar to R_1, and it is natural to interpret R_3 in relation to R_1. Therefore, in the final hidden structure we want a cluster to include all the three requirements.

The second step of the algorithm aims at systematically resolving situations such as the one described above. To this end, clusters are merged if the last requirements of a cluster (*tail*) are sufficiently similar to the first requirements (*head*) of a following cluster.

More formally, given a *window size* parameter $w \in \mathbb{N}$, we define the head of a cluster as the set of its first w requirements, while we define the tail of a cluster as the set of its last w requirements. If a preliminary cluster $\tilde{C}$ is composed by $|\tilde{C}|$ requirements $R_0 \dots R_{|\tilde{C}|-1}$, its head is: $h(\tilde{C}) = \{R_i \in \tilde{C} : 0 \leq i < w\}$, while its tail is: $t(\tilde{C}) = \{R_i \in \tilde{C} : 0 < |\tilde{C}| - w \leq i \leq |\tilde{C}| - 1\}$.

The comparison among two preliminary clusters $\tilde{C}^i$, $\tilde{C}^j$ (with $i < j$) is performed considering the tail of $\tilde{C}^i$ and the head of $\tilde{C}^j$ according to the following similarity function:

$$s(\tilde{C}^i, \tilde{C}^j) \;=\; \max_{\substack{R' \in \{t(\tilde{C}^i)\} \\ R'' \in \{h(\tilde{C}^j)\}}} \sigma(R', R'').$$

Again, σ may be computed in terms of σ_{jac}, σ_{lev}, or $\sigma_{jac-lev}$.

In order to merge preliminary clusters, we compare head and tail of each cluster $\tilde{C}^i$ with the l following ones, from $\tilde{C}^{i+1}$ to $\tilde{C}^{i+l}$. The parameter l defines how many subsequent clusters shall be compared with $\tilde{C}^i$. We call this parameter *lookahead*. Clusters are merged if their similarity is higher than the previously defined threshold τ (i.e., if $s(\tilde{C}^i, \tilde{C}^j) > \tau$).

The goal of this comparison with the l clusters that follow $\tilde{C}^i$ – and not solely with $\tilde{C}^{i+1}$ – is to isolate requirements, or group of requirements, that are not sufficiently related with the neighbouring ones. Consider for example the following requirement, R_u : *"The system shall support recovery from loss of communication"*.

Let us now consider again the requirements R_1, R_2, R_3. If we have an original requirement document such as $D = \langle R_1, R_2, R_u, R_3 \rangle$, requirement R_u might look misplaced. Indeed, it is related to the recovery part of the system, while the other requirements are speaking about procedural aspects. Therefore, we would like the hidden structure to include a $\langle R_1, R_2, R_3 \rangle$ cluster, and isolate R_u in a separate cluster. This goal is addressed by comparing the cluster $\tilde{C}^1 = \langle R_1, R_2 \rangle$ with both the cluster $\tilde{C}^2 = \langle R_u \rangle$ and $\tilde{C}^3 = \langle R_3 \rangle$. Since $\tilde{C}^1$ is similar to $\tilde{C}^3$, these will be merged, and $\tilde{C}^2 = \langle R_u \rangle$ will appear as an isolate cluster.

Summarizing, the final hidden structure $\pi(D)$ is computed as follows. The first cluster C^1 is initialized with the first preliminary cluster $\tilde{C}^1$. Then, C^1 is compared with with any other following preliminary cluster $\tilde{C}^j$, with $j = 2 \dots 1 + l$, by computing the similarity $s(C^1, \tilde{C}^j)$. If $s(C^1, \tilde{C}^j) > \tau$, then $\tilde{C}^j$ is merged into C^1, the *head* and *tail* requirements of C^1 are updated consequently, and $\tilde{C}^j$ is removed from the set of preliminary clusters. The algorithm continues comparing the new cluster C^1 with the remaining preliminary clusters until

$j = 1 + l$. When all the remaining preliminary clusters within the lookahead interval have been considered, the oldest preliminary cluster available is used to build a new cluster C^2. The algorithm iterates this procedure until no more preliminary clusters are left.

The final output of the algorithm is the sequence of clusters that represents the *hidden structure*, namely $\pi(D) = \langle C^1 \dots C^{|\pi(D)|} \rangle$.

3.4 Best Hidden Structure

The output of the algorithm depends on the similarity function σ adopted, and on three parameters, namely, the similarity threshold τ, the window size w, and the lookahead l. By executing the algorithm with different similarity functions and different combinations of values for the parameters, different hidden structures are generated. We call these structures *hidden structure candidates*. These structures represent possible partitioning of the document perceived by a reader who reads the document without having the original structure information. Intuitively, the variation of the parameters allows the identification of possible variants in the interpretation of the document structure.

Given the hidden structure candidates, we select the one that more closely matches the structure originally provided by the document. We call this candidate the *best hidden structure*. In our view, the best hidden structure represents the structure that is the closest to the conceptual partitioning that might be perceived by a reader who is provided with the original structure information. The reasonable assumption here is that the original structure is not completely flawed, but has been defined with the purpose of providing a useful guidance for the reader.

The comparison among the hidden structure candidates and the original structure is performed according to the three following indexes: (1) *number of clusters*, (2) *average* number of requirements per cluster, and (3) *standard deviation* of the number of requirements per cluster. This last index is useful to discard hidden structures that, for example, have regular clusters, while in the original document the sections largely vary in length.

The hidden structure candidate that more closely matches the original structure according to these indexes is chosen as the best hidden structure.

Finally, the best hidden structure is analysed and compared with the original structure by a human operator to assess the differences, and identify parts of the document that lack *requirements relatedness*, and *sections independence*. Possible cases are the following:

- If there is a section in the document that includes several lexically independent sub-parts (i.e., a section corresponds different clusters of the best hidden structure), there is probably a lack of section independence. Such section should be probably partitioned into more sections.
- If two separate sections of the document are lexically related (i.e., the sections appears in a single cluster of the best hidden structure), there is too much dependency among sections. It should be probably preferable to merge the two sections into a single one.

– If some requirements are not lexically related to the neighbouring requirements, like in the case of R_u of the example in Sect. 3.3, there is likely to be a lack of requirements relatedness. The requirement should be placed in another section.

4 Pilot Test: EIRENE

We evaluate the effectiveness of our proposed method on a publicly available requirements document of the railway domain, namely the *UIC EIRENE Functional Requirements Specification version 7* [19], issued in 2006. The requirements specified by the document refer to the EIRENE (European Integrated Railway radio Enhanced NEtwork) system, which is the digital radio standard for the European railways. The standard gives prescriptions concerning the network services that are required to support communications that involve trains, stations and railway personnel. Furthermore, requirements are given also concerning the interface of the system with the driver of the train.

The document comprises 583 requirements partitioned into 14 sections. The introductory section of the document is not considered in our dataset. Other relevant statistics on the document content are reported in Table 1.

Table 1. The EIRENE dataset

Requirements	
Number of requirements	583
Total number of distinct terms (stemmed and without stop-words)	879
Average number of terms per requirement	16
Standard deviation of the number of terms per requirement	14
Sections	
Number of sections	14
Average number of requirements per section	42
Standard deviation of the number of requirements per section	46

The document has been processed through the S-HTC algorithm to produce the hidden structure candidates. Each run of the algorithm, and therefore each produced candidate, depends on the similarity metric adopted, and on the specific values of τ, w and l. The similarity metrics are the σ_{jac}, σ_{lev} and $\sigma_{jac-lev}$. Different combination of values of the parameters τ, w, and l have been used. In particular, the threshold τ varies between 0 and 1, with increasing steps of 0.1. For each value of τ, we vary both w and l. The window size is $w \in \{1, 2\}$, which implies that we consider the head/tail of a preliminary cluster as a single requirement, or two requirements. The lookahead is $l \in \{1, 2, 3\}$. Therefore, at most three preliminary clusters are compared each time with the current cluster, to evaluate possible merge operations in the second step of the S-HTC algorithm. In the context of the document under analysis, we considered that a maximum lookahead of three was sufficient to isolate misplaced requirements according to

the rationale discussed in Sect. 3.3. Larger ranges could be used to properly discover misplaced requirements in documents with different writing styles.

Considering all the combinations of the parameters for each one of the similarity metric, we derive 198 hidden structure candidates in total (66 for each similarity metric). Fig. 2 gives a comparative view of the number of clusters produced with the different similarity metrics. In the x axis we order the hidden structure candidates considering increasing values of τ, and increasing variation of w and l within their ranges.

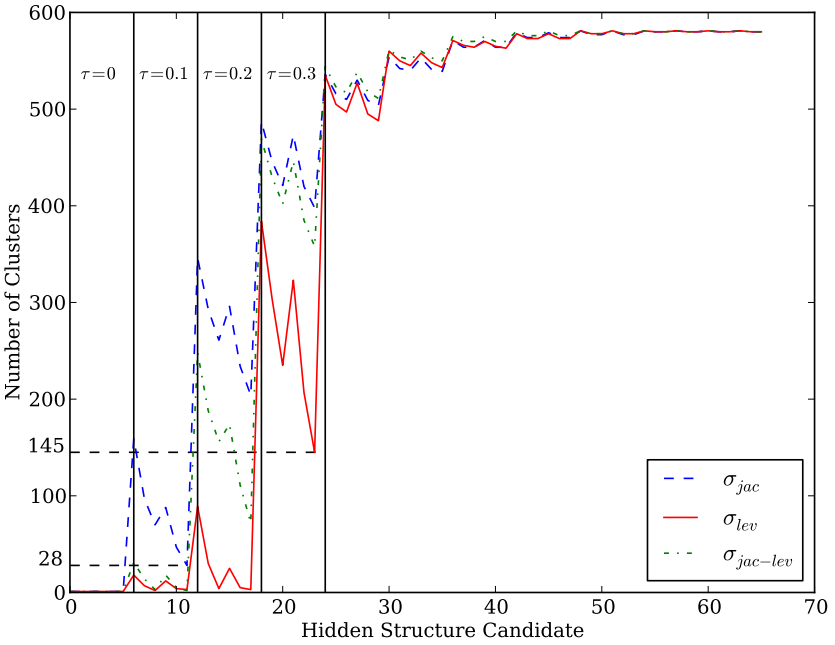

Fig. 2. Number of clusters for each hidden structure candidate

Hidden Structure Candidates. We notice that, regardless of the similarity metric considered, the number of clusters of the hidden structure candidates rapidly increases towards values that are not comparable with the number of sections of the original document. Indeed, already with $\tau = 0.3$ (hidden structure candidates from 18 to 23, according to the numbering of the x axis), the minimum number of clusters is 145, obtained for $\sigma = \sigma_{lev}$, $\tau = 0.3, w = 1, l = 3$.

We also notice that, for values of $\tau \leq 0.3$, the number of clusters produced with the σ_{jac} similarity metric is always higher than the number of clusters produced with σ_{lev} and $\sigma_{jac-lev}$. In particular, the minimum number of clusters produced with σ_{jac} is 28 (obtained for $\tau = 0.1, w = 2, l = 3$), exactly twice the number of sections in the original document[2]. Therefore, we argue that the σ_{jac}

[2] We do not consider the cases when $\tau = 0$, for which the algorithm produces always one cluster, regardless of the similarity metric employed.

similarity metric is basically not representative in the context of the requirements document under analysis.

According to these observations, the only candidates that can be reasonably evaluated to select the best hidden structure are those generated with $\sigma \in \{\sigma_{lev}, \sigma_{jac-lev}\}$, and $\tau \in [0, 0.2]$. Fig 3 reports an excerpt of the previous plot, centred on these hidden structure candidates.

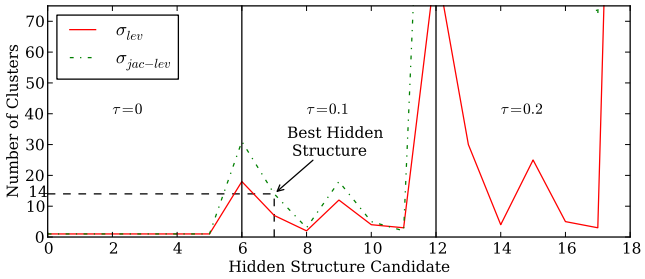

Fig. 3. Number of clusters produced when $\tau \in [0, 0.2]$

Best Hidden Structure. The best hidden structure, highlighted in Fig 3, is found for $\sigma_{jac-lev}$, $\tau = 0.1$, $w = 1$ and $l = 2$. In Table 2 we list the best candidates, together with the values for the indexes that have been used to select the best hidden structure (highlighted in bold).

Table 2. Best hidden structure candidates

σ	τ	w	l	Clusters	Average	Std Deviation
$\sigma_{jac-lev}$	**0.1**	**1**	**2**	**14**	**42**	**35**
σ_{lev}	0.1	2	1	12	49	58
$\sigma_{jac-lev}$	0.1	2	1	18	32	47
σ_{lev}	0.1	1	1	18	32	37
Expected				**14**	**42**	**46**

In the last row of the table we recall the expected values for such indexes, namely the number of sections of the original document, and the average and standard deviation of the number of requirements per section. More accurate approaches for the evaluation of clusterings could be employed (see, for instance, the similarity-oriented measures of cluster validity in [18]). However, in the presented experiment, the indexes adopted resulted sufficient to identify the best hidden structure.

We evaluate the original structure of the document and we compare it with the best hidden structure selected. Furthermore, we consider the most recent version of the requirements document (version 7.3, issued in 2012). The updates found in such document are used to give hints to evaluate the effectiveness of the approach.

Table 3. Comparison among the *original structure* and the *best hidden structure*

Section	Title	Cluster
2	Network Requirements	C^1
3	Network Configuration	C^2
4	Mobile equipment core specification	C^2
5	Cab radio	C^3, C^4, C^5
6	General purpose radio	C^6, C^7
7	Operational radio	C^8
8	Controller equipment specifications	C^9
9	Numbering plan	C^9
10	Subscriber management	C^{10}
11	Functional numbering and location dependent addressing	C^{10}
12	Text messaging	C^{11}
13	Railway emergency calls	C^{11}
14	Shunting mode	C^{11}
15	Direct mode	C^{12}
-	-	C^{13}, C^{14}

Table 3 compares the section of the original document with the clusters of the best hidden structure. In general, we observe that clusters in the hidden structure can be mapped to corresponding sections, or groups of sections. The two clusters that cannot be mapped (C^{13} and C^{14}) include requirements that are deleted from the specification, and which, in the original document, are formed solely by the term "deleted". These are basically clusters of noisy items.

Let us give a closer look to Table 3. The first section after the introduction, named "Network Requirements", has a corresponding cluster in the hidden structure that includes all its requirements. We conclude that such section is sufficiently independent from the others. The same observation holds for sections 7 and 15, named "Operational Radio" and "Direct Mode", respectively.

Different conclusion can be drawn from the evaluation of the other clusters. Most of the requirements of Sections 3 ("Network Configuration") and 4 ("Mobile equipment core specification") are included in the same cluster. Looking at the content, we see that the reason why the two sections are merged into the same cluster are the last requirements of section 3, namely requirements 3.5.7 and 3.5.8. The requirements are as follows:

- 3.5.7: *Cab Radios configured for reception of a call to all drivers in the same area entering an area where a call to all drivers in the same area is ongoing shall automatically join this call unless involved in a higher priority call or involved in a call of the same priority.*
- 3.5.8: *Requirement 3.5.7 needs further technical specification changes before field implementation can be achieved.*

The first requirement is evidently unreadable, and, most of all, anticipates some content discussed in section 4. This is the reason why the two sections appear in the same cluster, and it is an evident lack of *sections independence*. Instead, the second requirement is included in another cluster, since it is not lexically related with the previous one. The relation with the previous requirement is given by

the cross-reference, and our algorithm is not thought to deal with such semantic issues.

It is worth noticing that, in the most recent version of the specification, both the requirements have been deleted.

Section 5, named "Cab Radio", is partitioned into 3 different clusters, which is an indicator of a lack of *section independence*. This section is the longest of the document (188 requirements in total, 32% of the entire document), and could be rearranged into smaller sections. In particular, there are requirements that are contiguous in the document, but that are not strictly related (i.e., there is a lack of *requirements relatedness*). Consider the following requirements:

- 5.4.3: *All call related functions shall be possible with the handset on or off the hook.*
- 5.4.4: *Note: there is no requirement for hands free operation.*
- 5.4.5: *The driver shall be able to adjust the brightness of buttons, indicator lights and displays according to the ambient lighting in the cab.*
- 5.4.6: *The driver shall be able to adjust the contrast of the display.*

In the best hidden structure, requirements 5.4.5 and 5.4.6, which concern the interface of the system with the driver of the train, are not clustered with requirements 5.4.3 and 5.4.4. Instead, they are clustered with other requirements that give prescriptions concerning the system-driver interface (5.2.3.18-19). The re-arrangement suggested by the best hidden structure is as follows:

- 5.2.3.18 *It shall be possible for the driver to increase and decrease the loudspeaker volume within the adjustment range selected.*
- 5.2.3.19: *When the handset is picked up, the loudspeaker shall continue to operate, but at a reduced volume level.*
- 5.4.5: *The driver shall be able to adjust the brightness of buttons, indicator lights and displays according to the ambient lighting in the cab.*
- 5.4.6: *The driver shall be able to adjust the contrast of the display.*

The best hidden structure suggests to group together requirements related to the audio interface with those related to the visual interface, which is a reasonable solution. Furthermore, we notice that requirement 5.4.4, which is evidently vague, has been largely modified in the most recent version of the document. We argue that such modification is an indicator that the part of the document considered lacked proper requirements relatedness, as pointed out by our algorithm.

Similar observations can be drawn from the analysis of the rest of the best hidden structure. We find that section 6 is partitioned into two clusters, revealing a lack of sections independence. Furthermore, the other remaining sections tend to be grouped together. In particular, we find three clusters that group sections 8-9, 10-11, and 12-13-14, respectively. This is a witness of dependency among sections. However, we have noticed that such dependency is not actually evident while reading the sections grouped in the same clusters. In these cases, the section partitioning given by the original document is probably preferable to the one suggested by the best hidden structure.

5 Related Work

The structure of the requirements document is considered as a central element to evaluate the quality of a requirements specification [20,2,8]. Several international standards provide templates to enforce a proper structuring of the requirements. The main reference in software engineering is the IEEE Standard 830-1998 [10], which defines eight possible templates, each one focused on a different approach for organizing the document (e.g., by functional hierarchy, by feature, etc.). Each template lists the sections and the content that a software requirements specification is supposed to provide. A similar approach is adopted in the MIL 498 standard [14], where the template is tailored for requirements document of the defense sector. In the railway domain, the CENELEC EN 50128 standard [3] does not provide a specific template. Instead, guidelines are given to provide a proper structuring of the requirements document. In this case, companies normally provide internal templates that are used across different projects.

Several studies exist that are focused on the automatic analysis of the structure of a *generic* natural language document. Mao *et al.* [13] provides a comprehensive survey of the different approaches available in the literature. Most of the works rely on a set of rules or templates that enable the automatic identification of the different logical parts of the document, such as titles, abstract and sections. To our knowledge, the only work on automatic structure analysis that is specifically concerned with requirements documents has been recently proposed by Rauf *et al.* [17]. In that paper, a framework is presented to retrieve the logical structure of requirements documents. The logical structure items (e.g., functional requirements or use cases) that the framework is supposed to identify are defined in the form of templates. The framework retrieves instances of such items from the document according to the templates.

Compared to the other works, the main novelty of the current paper is the automatic evaluation of the structure of a requirements document *without* a reference template. On the other hand, the technologies that we have employed, in particular the text similarity metrics, have been widely used in natural language requirements analysis. Such related works use technologies similar to ours to achieve different goals. Besides our previous contribution [7], where a clustering-based approach is used to identify requirements flaws, it is worth citing the comprehensive study of Falessi *et al.* [6]. Here, the authors experiment different similarity metrics to identify equivalent requirements. Other relevant contribution are those of Natt och Dag *et al.* [5], focused on the identification of conceptual links between newly-arrived and previously stored customer requirements, and Park *et al.* [15], where text similarity metrics are employed to support the requirements analyst in finding inconsistencies between high-level and low-level requirements.

Textual similarity analysis has been widely used also for automated requirements tracing. The main reference works have been published, among others, by Hayes (see, e.g., [9]) and Cleland-Huang (see, e.g., [4]). However, in the traceability domain, the majority of the works employs information retrieval and statistical machine learning methods, while, in our work, a clustering-based approach is adopted.

6 Conclusions

In this paper, a clustering-based approach to evaluate the structure of a natural language requirements document is presented. We focus on the identification of those parts of the document that lack proper requirements relatedness and proper sections independence. The approach identifies the *hidden structure* of the document (i.e., the structure possibly perceived by the reader) and compares it with the original document structure to assess inconsistencies.

The method has been employed to evaluate a requirements document standard of the railway domain. To fully evaluate the approach, further experiments on other documents, and with the support of domain experts, are required. Nevertheless, our pilot study has shown that the proposed algorithm is effective in identifying parts of the documents with poor structuring, sections that shall be re-arranged, and requirements that are misplaced.

On the other hand, the algorithm behaviour shall be tuned to better deal with issues related to sections dependency. We have seen that, in some cases, the algorithm reveals dependency among sections that, from a reader's perspective, are not actual conceptual dependencies. We argue that taking into account also semantic aspects when performing requirements clustering might resolve these situations. To this end, we plan to explore latent semantic analysis and term frequency analysis approaches, as well as exploiting external knowledge sources (e.g., Word-Net[3], Wikipedia[4]) for capturing the semantic relatedness between requirement pairs.

References

1. Achananuparp, P., Hu, X., Shen, X.: The evaluation of sentence similarity measures. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2008. LNCS, vol. 5182, pp. 305–316. Springer, Heidelberg (2008)
2. Berry, D.M., Bucchiarone, A., Gnesi, S., Lami, G., Trentanni, G.: A new quality model for natural language requirements specifications. In: Proc. of REFSQ 2006, pp. 115–128 (2006)
3. CENELEC: EN 50128, Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems (2011)
4. Cleland-Huang, J., Czauderna, A., Gibiec, M., Emenecker, J.: A machine learning approach for tracing regulatory codes to product specific requirements. In: Proc. of ICSE 2010, vol. 1, pp. 155–164. ACM, New York (2010)
5. Natt och Dag, J., Gervasi, V., Brinkkemper, S., Regnell, B.: A linguistic-engineering approach to large-scale requirements management. IEEE Software 22, 32–39 (2005)
6. Falessi, D., Cantone, G., Canfora, G.: Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. IEEE Transactions on Software Engineering PP(99) (2011)
7. Ferrari, A., Gnesi, S., Tolomei, G.: A clustering-based approach for discovering flaws in requirements specifications. In: Proceedings of ACM SAC 2012, pp. 1043–1050 (2012)

[3] `http://wordnet.princeton.edu/`
[4] `http://www.wikipedia.org`

8. Gervasi, V., Nuseibeh, B.: Lightweight validation of natural language requirements. Software: Practice and Experience 32(2), 113–133 (2002)
9. Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: The study of methods. IEEE Trans. Software Eng. 32(1), 4–19 (2006)
10. IEEE: Std 830-1998 - Recommended Practice for Software Requirements Specifications (1998)
11. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10(8), 707–710 (1966)
12. Lucchese, C., Orlando, S., Perego, R., Silvestri, F., Tolomei, G.: Identifying task-based sessions in search engine query logs. In: Proc. of WSDM 2011, pp. 277–286. ACM, New York City (2011)
13. Mao, S., Rosenfeld, A., Kanungo, T.: Document structure analysis algorithms: a literature survey. In: Proc. of DRR 2003, pp. 197–207 (2003)
14. MIL: Std 498 - Software Development and Documentation (1994)
15. Park, S., Kim, H., Ko, Y., Seo, J.: Implementation of an efficient requirements-analysis supporting system using similarity measure techniques. IST 42, 429–438 (2000)
16. Pohl, K.: Requirements Engineering: Fundamentals, Principles, and Techniques. Springer (2010)
17. Rauf, R., Antkiewicz, M., Czarnecki, K.: Logical structure extraction from software requirements documents. In: Proc. of IEEE RE 2011, pp. 101–110. IEEE Computer Society, Washington, DC (2011)
18. Tan, P., Steinbach, M., Kumar, V.: Introduction to Data Mining. Addison-Wesley, Boston (2005)
19. UIC - International Union of Railways: EIRENE Functional Requirements Specification v.7 (2006), `http://www.uic.org/IMG/pdf/EIRENE_FRS_v7.pdf`
20. Wilson, W.M., Rosenberg, L.H., Hyatt, L.E.: Automated analysis of requirement specifications. In: Proc. of ICSE 1997, pp. 161–171. ACM Press, New York (1997)