applied sciences

MDPI

*Article*

# Genetic Algorithms: A Practical Approach to Generate Textual Patterns for Requirements Authoring

**Jesús Poza *, Valentín Moreno, Anabel Fraga and José María Álvarez-Rodríguez**

Knowledge Reuse Group, University Carlos III, 28911 Madrid, Spain; vmpelayo@inf.uc3m.es (V.M.); afraga@inf.uc3m.es (A.F.); josemaria.alvarez@uc3m.es (J.M.Á.-R.)
* Correspondence: jepozac@inf.uc3m.es

**Abstract:** The writing of accurate requirements is a critical factor in assuring the success of a project. Text patterns are knowledge artifacts that are used as templates to guide engineers in the requirements authoring process. However, generating a text pattern set for a particular domain is a time-consuming and costly activity that must be carried out by specialists. This research proposes a method of automatically generating text patterns from an initial corpus of high-quality requirements, using genetic algorithms and a separate-and-conquer strategy to create a complete set of patterns. Our results show this method can generate a valid pattern set suitable for requirements authoring, outperforming existing methods by 233%, with requirements ratio values of 2.87 matched per pattern found; as opposed to 1.23 using alternative methods.

**Keywords:** text patterns; genetic algorithm; knowledge reuse; requirements authoring; requirements engineering



**Citation:** Poza, J.; Moreno, V.; Fraga, A.; Álvarez-Rodríguez, J.M. Genetic Algorithms: A Practical Approach to Generate Textual Patterns for Requirements Authoring. *Appl. Sci.* **2021**, *11*, 11378. https://doi.org/10.3390/app112311378

Academic Editor: Giancarlo Mauri

Received: 27 September 2021
Accepted: 29 November 2021
Published: 1 December 2021



## 1. Introduction

The emergence of digitalization in the engineering discipline has led to the improvement of the engineering process of safety-critical systems. According to the report on the "Future of Systems Engineering" [1] by the International Council of Systems Engineering (INCOSE), there are five major goals ("Model use for decision making", "Authoritative source of truth", "Tech innovation", "Collaborative environment" and "Digital engineering workforce and culture") and several stages to reach the major objective of digital engineering. Interoperability, standardization, digital twins, semantics, simulation, and Model-based Systems Engineering [2] (MBSE) are key technologies to bring digitalization to the Systems Engineering discipline. In this context, the notion of "augmented engineering" will be reached once data can semantically link together and exploited through different techniques such as AI/ML to automate some existing, complex and, in many cases, manual tasks, e.g., recovery traceability links between system artifacts [3], generate documentation or check quality (consistency) of the system under development.

On the other hand, the proper specification of a system has been demonstrated to be one of the cornerstones to timely and safely deliver complex products and services. Requirements Engineering [4] is a traditional engineering method used to specify complex systems. The Requirements Engineering Process comprises several activities to elicit, analyze, specify, verify, and validate requirements at different levels of abstractness (business, user/stakeholder, system, software, telecommunications, system operational concept, etc.). Commonly, these activities are implemented using different techniques to finally generate a specification document. More specifically, requirements authoring is usually done through text-based statements to be understandable for both business and technical users. Other more formal techniques such as logical models (e.g., linear temporal logic) may exist but, in the end, they are usually verbalized to ease the communication between the agents involved in the specification process. In this sense, semi-formal techniques like boilerplates

or patterns are also used to restrict how requirement statements can be written (syntax) and what they can specify (semantics).

In the latter, the use of ontologies including the main domain entities and relationships of the system under specification represents the ground truth to properly specify requirements and being able to verify [5] the quality properties of consistency, correctness, and completeness (CCC). In this sense, it is possible to assess, for example, that a system element is assigned to the proper operations and restrictions making use of the required magnitudes and units of measurement. Technological solutions integrated with the main requirements management systems such as the Requirements Authoring Tool (RAT) and methodologies like Volere can be also found in the market. In fact, RAT is a specific tool, part of the Reuse SE Suite [6], oriented to ease the writing of requirements following the approach of predefining a terminology, a set of semantic relationships, and a set of patterns to write requirement statements, see Figure 1. This tool has been successfully deployed in different safety-critical systems such as aerospace and automotive and, it also includes libraries to measure quality ensuring that every requirement and the specification by itself accomplish with the CCC quality properties.

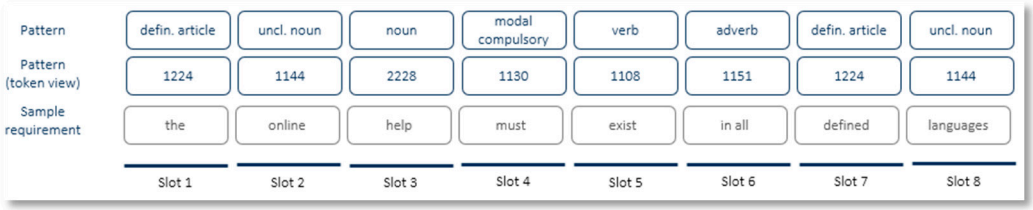


**Figure 1.** Example of requirement pattern within the RAT tool. Tokens are integers representing syntactical and semantic terms.

However, the intrinsic need of predefining a knowledge base (terminology, concepts, relationships, and patterns) and the effort to reuse existing specifications including standards (in many cases, specifications of the safety-critical system contain the order of thousands of requirement statements), are preventing the proper automation of this first stage within the engineering process in the scope of the major goal of "Authoritative source of truth". Therefore, it is essential to provide new tools that aid in these processes.

From a technical perspective and to avoid defining patterns from scratch, a method to automatically infer patterns from existing documents (e.g., existing requirements specifications, standards, etc.) can allow us to ease the process of requirements authoring by reusing existing text-based statements.

In this paper, we will describe a solution capable of automatically generating requirements patterns, based on the use of genetic algorithms [7] and separate-and-conquer methods [8], in conjunction with a controlled vocabulary. The result patterns may conduct the authoring process, avoiding complex structures, ambiguity, or using a vocabulary not included in the dictionary.

Afterward, the proposed method to infer textual patterns is validated with a real corpus of requirements and compared to previous works [9]. Finally, some conclusions and future works are also outlined.

## 2. State of the Art

After more than 50 years [10], the use of requirements in the Software and Systems Engineering lifecycle is a widely accepted method for analysis purposes [11]. In the classical view, requirements try: (1) to address or answer the question "What?" (More specifically, What the target system is supposed to do?) (2) to serve as a system specification that can be used in further stages such as design or testing.

In this sense, requirements as common practice for understanding and stating both stakeholders' intentions and system functionalities is still far from being fully developed and, in many cases, the use of requirements is isolated in the inception phases of the

development process. However, the current practice in the Systems Engineering discipline strongly relies on building an underlying knowledge graph [12] of the development lifecycle being able to trace any piece of information. For instance, requirements data can be used to boost the use of descriptive and analytical models [13], current practice in the Systems Engineering discipline.

That is why the first step to properly state requirements relies on being able to promote textual statements into concept-based descriptions. Domain entities and relationships can be used then to model the structure and semantics of textual requirements (i.e., patterns) and then, reuse such information to automate more complex tasks such as traceability link recovery, automatic model population, or test case generation to name a few. To do so, it is possible to find a general set of patterns like those in the EARS [14] (Easy approach to requirements syntax) approach or those defined in the context of the CESAR (Cost-efficient methods and processes for safety-relevant embedded systems) European project (deliverable "Requirement Capturing Specification of Improved Methods" D_SP2_R3.3_M3_Vol2). However, they only offer a general syntax structure without paying attention to the semantics of the requirements statements.

In this context, ontologies emerge to provide a common understanding of what is being specified. Formal ontologies have been widely used in multiple domains, e.g., e-Health, Tourism, e-Government, e-Learning, or Information Science, and with multiple objectives: information extraction, information retrieval, or recommendation engines. More specifically, in the Systems Engineering discipline, ontologies have been applied to unify domain knowledge under a common data model and paradigm, e.g., The NASA QUDT Units Ontology or the NASA Knowledge graph built with Stardog, and to reason about data consistency. The European Space Agency (ESA) has also led to the creation of the Space Systems ontology, used to model the underlying knowledge generated during the development of space products and ease the collaboration between the agency and the industry. The ESA has also created the Object-Role Modelling [15] language to define patterns and verbalize them into requirements. However, all these approaches are based on the assumption that patterns must be first generated.

On the other hand, in the field of computational linguistics, it is possible to find pure methods to infer textual patterns like the one presented in [16] to reconcile company names. Foundational language models [17], based on Deep Learning techniques such as transformers [18], have been widely used to perform natural language processing tasks such as text generation and entity recognition. However, these methods require a large corpus and an adequate training time to provide meaningful results that may not apply to a vertical domain like requirements authoring.

As preliminary conclusions, requirements are a first member of the Software and Systems Engineering process. Although methodologies like MBSE are now focusing on models, the reality shows that textual requirements are still valid to specify a system and to serve us as a communication tool. Moreover, model verbalization that has been explored [19] for a long time in other domains, is now a key aspect in the Systems Engineering discipline to improve the explainability of both descriptive and analytical models. In this context, patterns, and semantics (ontologies) emerge to help in the authoring process of, at least, requirements. However, and considering the huge amount of requirements statements already available in different specifications and standards, there is not yet a common approach to properly infer requirements patterns. Some initial attempts using machine learning [20] have been done but approaching the problem from the perspective of quality and learning from a labeled dataset. The Reuse Knowledge Group has also investigated an approach to obtain text patterns automatically [9]. Using a set of 545 requirements, they got 442 text patterns. The ratio of text patterns found versus the number of requirements was 0.81 (This ratio will be used to measure our experiment's performance versus this previous work) not good enough to incorporate the solution as part of the Reuse SE Suite.

That is why, in this work, we propose a new approach based on the use of Genetic Algorithms (GA) [7] in conjunction with separate-and-conquer [8] strategy. This method

will deal with thousands of combinations of terms trying to find quality patterns and checking them against a knowledge base and previous experiments.

GA are an approach to solve non-deterministic problems inspired by natural selection [7,21]. The basis is the generation of a variety of possible solutions to a problem (population). Using a selection criterion (fitness function), the algorithm selects the best individuals and generates offspring that inherit the characteristics of the parents. If parents demonstrate good fitness, the breeding process (interchange of parent's genes) will generate better offspring. This is an iterative process that, in the end, may find good solutions to the problem. Since the method is not deterministic and uses randomness as a source of variety, a valid final solution is not guaranteed.

GA have been used in a wide variety of scenarios where no deterministic approaches are feasible: searching optimal ubication of wind turbines in wind farms [22], requirements prioritization [23], generating regular expression patterns [24], and solving the traveling salesman problem [25,26].

These are the activities involved in a generic GA:

1. Solution codification: define genes, individuals, and population.
2. Define a fitness function to evaluate how suitable every individual is as a possible solution to the problem.
3. Randomly generate an initial population.
4. ITERATE
    a. Evaluate population of generation n using the fitness function
    b. Select parents to cross breed (creating an individual pool)
    c. Crossbreed parents. Generate generation n + 1
    d. Mutation of generation n + 1
    e. Replace generation n with generation n + 1
5. UNTIL exit condition.

Separate-and-conquer strategy has been long used in many inductive rule learning algorithms [8]. Basically, the separate-and-conquer algorithm consists of a global loop that searches for a particular solution that partially solves the problem; the instances of the problem solved are separated (conquest), the partial solution is stored, and the loop continues with the remaining instances not solved.

A similar approach, the combination of genetic algorithms and separate-and-conquer strategy, has been used by Bartoli in an experiment to find text patterns [24] for information extracting in regular expressions.

## 3. Methodology

### *3.1. Basic Definitions*

#### 3.1.1. Pattern, Tokens, and Slots

A textual pattern is a template, formal representation of a model or boilerplate that can be used to represent several real-world sentences (Figure 1). They are formed by cells containers called slots representing word positions. There have syntactic and semantic components called tokens filling the slots. Textual patterns allow the recognition of similar sentences throughout a document or sets of documents of a similar context. In Reuse SE tools, Tokens are represented by an integer (see Table 1).

#### 3.1.2. Requirement and Requirements Sets

For our purpose requirements are just textual sentences. They are processed by the Knowledge Manager Indexer component of the Reuse SE Suite using the basic NLP process (sentence segmentation, word tokenization, lemmatization, stemming, and syntactic and semantic analysis) [27] and transformed into slots and tokens, as shown in Figure 2.

**Table 1.** Sample table of tokens from Reuse SE tools.

| Token | Term | Token | Term |
|---|---|---|---|
| 1134 | acronyms (ok) | 2228 | noun |
| 1103 | adjective | 1123 | number |
| 1151 | adverb | 1117 | opening exclamation mark |
| 1106 | adverbial phrase | 1115 | opening question mark |
| 1130 | article | 1170 | phrasal verb |
| 1210 | aspectual verb | 1153 | place adverb |
| 1118 | closing angle brackets | 1199 | possessive pronoun |
| 1224 | definite article | 1105 | proposed adjective |
| 1111 | left slash | 1152 | time adverb |
| 1171 | modal phrase | 1241 | unclassified adjective |
| 1130 | modal compulsory | 1144 | unclassified noun |
| 1128 | month | 1108 | verb |

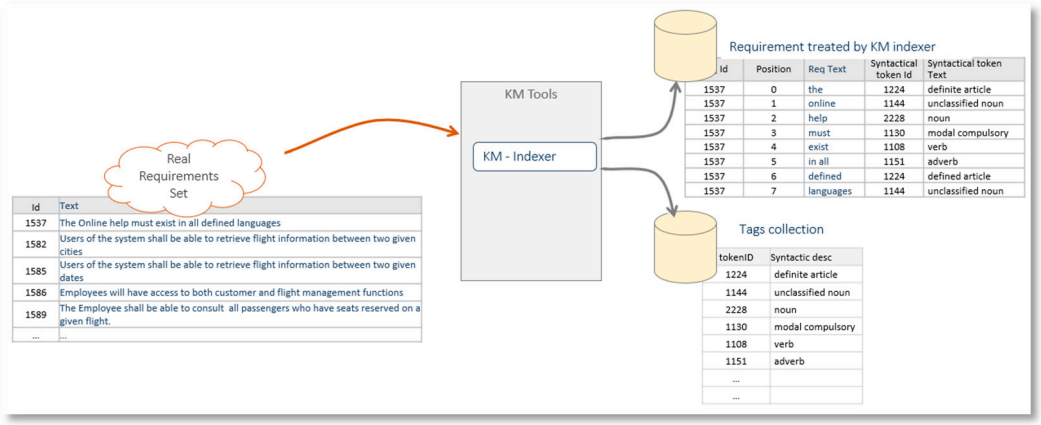


**Figure 2.** How KM Indexer process Requirements (tables, attributes, and relationships have been simplified to show relevant attributes for the experiment).

Another view of a requirement, similar to the Pattern view, is shown in Figure 3.


**Figure 3.** View of a Requirement after KM indexer process.

Requirements Sets (RS) represent a group of phrases that define user needs in a particular domain (i.e., engineering: aircraft cockpit electric requirements). They have been generated by experts and constitute our methodology's main input to find Requirements Pattern Sets (see Section 3.1.3).

#### 3.1.3. Requirements Patterns Set (RPS) for a Domain

A Requirements Patterns Set (RPS) is a group of patterns that can match most of the requirements of a specific domain. It defines the valid structure, vocabulary, syntactic and semantic restrictions of the requirements of the domain.

A domain is a collection of terms and rules used in describing a need or definition of solutions for a concrete business or system, or application. A domain describes the concepts and problems in that domain. The term" domain" can be used as an area of knowledge with common terminology [28]

Reuse SE Suite uses RPS in their core algorithms to check the quality of requirements and support the authoring process.

An RPS for a specific domain is built using a significant sample of requirements human written, selected, and qualified as good requirements. This process requires high knowledge and expertise, both in the business domain of the set (i.e., aerospace engineering) and in modeling and NLP disciplines. It is time-consuming and must be refreshed periodically as new requirements are incorporated into the domain with new valid vocabulary, etc.

Once a requirements patterns set is built, KM tools can use it to check the quality of new requirements or support the writing of new requirements in the phase of the requirements elicitation and definition of the system engineering process.

The quality of an RPS is valued by its simplicity, measured by the number of requirements that the set has, and completeness measured by the total number of requirements matched by the RPS. A good Requirements Patterns Set should have a fewer number of patterns and be able to recognize the maximum number of requirements.

So, the genetic algorithm's goal will be to find an RPS with the minimum number of patterns possible and the maximum number of matched requirements for a specific set of requirements.

#### 3.1.4. Pattern Matching

Requirements patterns are Text Patterns used to manage requirements. They are used to represent valid requirements models in a specific domain. The validation process of a requirement against a pattern consists of verifying if all components of the requirement (tokens) fit the tokens and slots defined in the pattern. In this case, we say that the requirement "match" the pattern and therefore is a good requirement for that pattern.

They can also be used to help in the authoring process. The SE authoring tool analyses the piece of requirement already written on the fly, searches the patterns that match the piece of phrase, and offers valid tokens to end the authoring process in the same way other predictive writing tools do [29].

The validation process of a requirement against a pattern consists of verifying if all components of the requirement (language tokens) fit against the slots defined in the pattern. In this case, we say that the requirement "match" the pattern and, therefore, is a good requirement.

Figure 4 shows an example of the matching process of several requirements against a specific pattern. To consider that a pattern matches a requirement, all requirements must match the pattern's tokens in the same order to consider that the requirement matches the pattern. Green shadowed slots are matched; Orange shadowed slots are slots not matched.

| | | | | | | | | Match Requirement |
|---|---|---|---|---|---|---|---|---|
| Pattern | 1130 | 1247 | 1142 | 1112 | 1130 | 1350 | | |
| Requirement 1 | 1130 | 1247 | 1142 | 1112 | 1130 | | | Yes |
| Requirement 2 | 1139 | 1144 | 1142 | 1350 | | | | No |

**Figure 4.** Matching process of a Pattern against several Requirements.

#### 3.1.5. Optional Slots and Wildcard Token

Slots may be optional, meaning that requirements having or not having the correspondent slot in the specific position are matched by the Pattern. Optional slots are represented in square brackets []. Wildcard slots are special Pattern slots that allow any kind and number of tokens and slots to be matched, in the matching process of a requirement any type and number of slots, until a slot in the requirement is identical to the following pattern slot found. Figure 5 shows an example of the matching process with optional and wildcard slots.

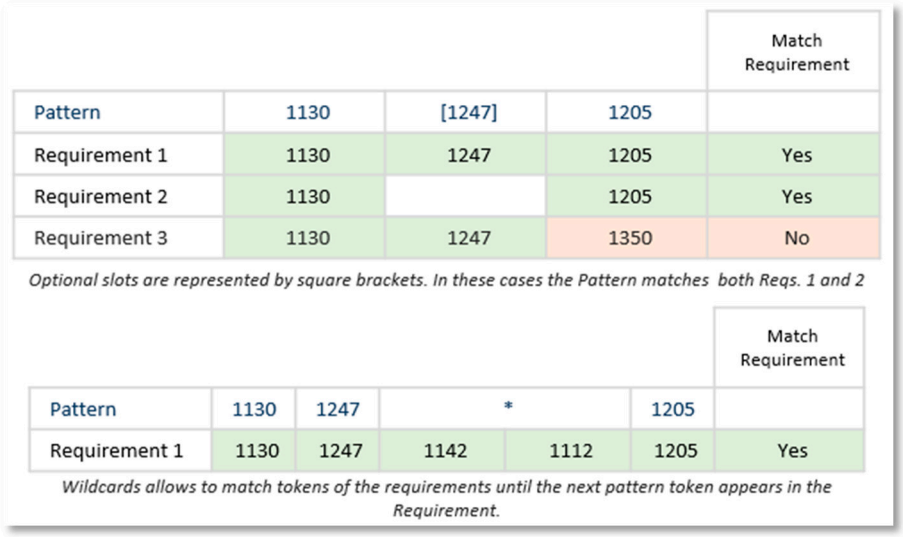


**Figure 5.** Special patterns slots: Optional slots represented with brackets and wildcard tokens, represented with an asterisk.

These are the two main mechanisms used to generate Requirements Patterns that recognize multiple requirements, patterns that match more than one requirement. In addition, for authoring purposes, these characteristics provide flexibility by allowing writing or not particular slots or parts in the process of writing requirements.

#### 3.1.6. Affinity Parameter

Wildcards would allow, eventually, the generation of a unique Pattern with one wildcard token as shown in Figure 6. This "universal" pattern would match any requirement in any domain.

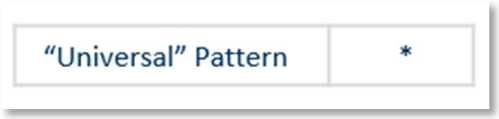


**Figure 6.** Universal Pattern. The Asterisk represents a wildcard token as described in Section 3.1.5.

The Genetic Algorithm will find this "universal" pattern as the best possible one since it matches all possible requirements. To avoid this problem, we have defined a parameter that limits the number of slots that a wildcard token can match. This is the *Affinity Parameter*. A value of 3 in this parameter means that only 3 slots may be matched by the wildcard, so if more requirements slots following the wildcard do not match the pattern next token, then the requirement is not matched by the pattern.

Figure 7 shows how different Affinity parameter values affect the matching process of the same Pattern with the same Requirement.

Affinity parameter is a powerful tool to manage genetic algorithm behavior. In search and extractions environments it can broaden or narrow the distance of specific desired words focusing only on the important words.

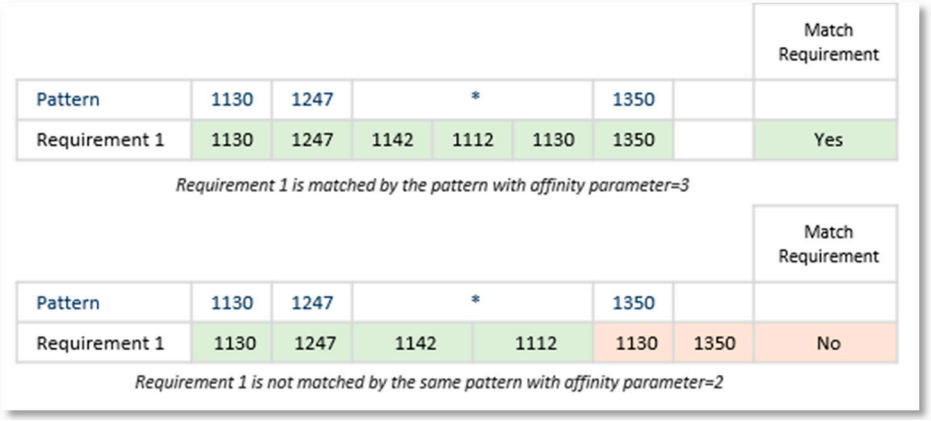


**Figure 7.** Impact of Affinity parameter in the matching process. The Asterisk represents a wildcard token as described in Section 3.1.5.

### *3.2. Algorithms Implementation*

#### 3.2.1. Solution Approach

We will combine the use of Genetic Algorithms (GA) with a divide-and-conquer strategy to obtain an optimum Requirements Patterns Set RPS. Figure 8 shows the global algorithm, including both GA and conquest iterations. We first use the GA to produce a population of patterns matching several requirements; the best candidate will be the pattern that matches the major number of requirements. This pattern is stored in the RPS, and the requirements matched by it are marked as conquest and excluded from the remaining RS. This constitutes a global iteration.

The algorithm continues with a new iteration with a smaller Requirements Set, including only those not yet matched with any of the patterns found. The genetic algorithm and the separate-and-conquest algorithm have both stopping conditions: The first stop condition is the conquest of all requirements; other stopping conditions are managed by parameters as described later in this section.

#### 3.2.2. Genetic Algorithm

Genetic Algorithm Definitions

**Genes:** In this experiment, genes are just slots. The slot is the minimum information that can be inherited from parents to children. It is filled with tokens that act as alleles of the gen. Each gen (slot) may be filled with different tokens (alleles) as shown in Figure 9.

**Individuals**: In the model are a group of genes. An individual has one and only one chromosome (set of ordered genes). There are two types of individuals: requirements and patterns. They have the same structure of chromosomes and genes. Patterns have also, for every gene the attribute that determines if that token may be optional. If true, the slot may appear or not in the requirements to match. There is also a special token present only in pattern individuals, it is the wildcard token, used also in the matching of the requirement process. Figure 9 shows individuals and genes for patterns and requirements.

**Populations**: A population is a group of individuals. In our model we have 2 types of population: Patterns Populations and Requirements Populations also named Requirements Sets. Individuals of the same population may have a different number of genes. This is a significant difference with respect to the nature paradigm, where all individuals of the same species share the same genotype, that is, the number of chromosomes and the number of genes. Figure 10 shows a sample or Requirements Population.

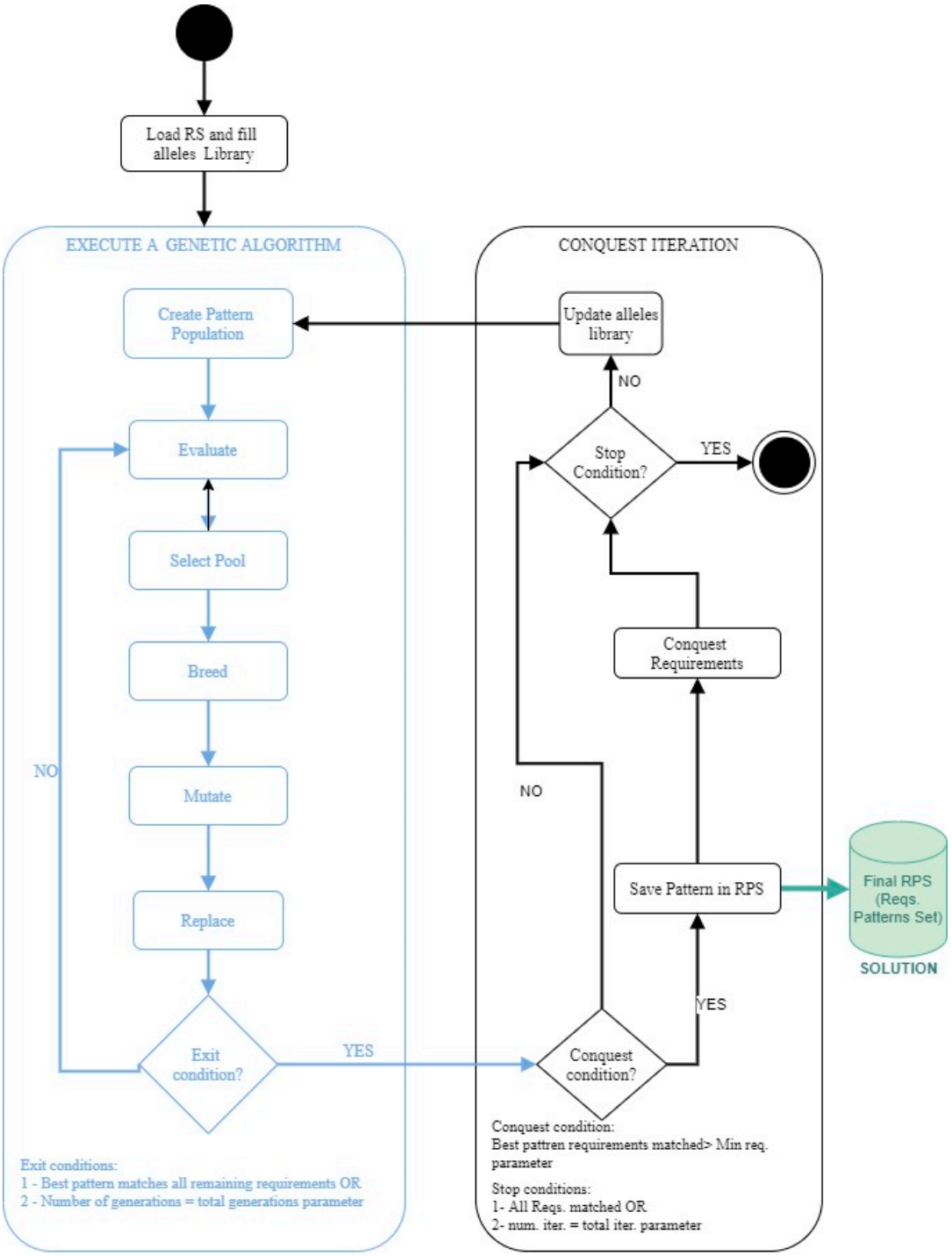


**Figure 8.** Diagram flow with the global algorithm.

| Pattern Individual | 1130 | [1247] | 1142 | * | 1130 | 1350 |
|---|---|---|---|---|---|---|
| Requirement Individual | 1130 | 1247 | 2228 | 1112 | | |
| Genes | Gen 1 | Gen 2 | Gen 3 | ... | | |

**Figure 9.** Genes and individuals (Requirements and Patterns). Asterisk represents a wildcard token as described in Section 3.1.5.

| | | | | | | |
|---|---|---|---|---|---|---|
| Requirement 1 | 1130 | 1247 | 1114 | 1350 | | |
| Requirement 2 | 1130 | 1228 | 1342 | 1350 | 1144 | 1247 |
| Requirement 3 | 1224 | 1144 | 1237 | 1094 | 1130 | |
| Requirement 4 | 1165 | 1144 | 1110 | 1248 | | |
| Requirement 5 | 1165 | 1144 | 1110 | 1248 | 1144 | 1248 |
| Requirement 6 | 1224 | 1144 | 1237 | 1094 | 1130 | 1094 |
| | | | | | | |

**Figure 10.** Sample Population of Requirements.

**Tokens frequency:** In a population of requirements (Requirements Set), the frequency of syntactical tokens is calculated when the genetic algorithm treats a new RS according to Equation (1). Table 2 shows an example of tokens and relative frequency for a particular RS.

$$\text{Token Frequency} = \frac{\text{Num of times token appears in the RS}}{\text{Total number of tokens in RS}} \quad (1)$$

**Table 2.** Sample of the table of tokens with relative frequency in a specific RS.

| Reqs Set Id | Token ID | Token Frequency | Token Count in Reqs Set | Syntactical Text |
|---|---|---|---|---|
| 4 | 1144 | 0.23657 | 3431 | unclassified noun |
| 4 | 1119 | 0.15362 | 2228 | noun |
| 4 | 1224 | 0.09019 | 1308 | definite article |
| 4 | 1108 | 0.04344 | 630 | verb |
| 4 | 1110 | 0.04116 | 597 | symbol |
| 4 | 1230 | 0.03530 | 512 | verb to be |
| 4 | 1229 | 0.03282 | 476 | preposition to |
| 4 | 1012 | 0.02565 | 372 | adjective |
| 4 | 1213 | 0.02462 | 357 | preposition |

Genetic Algorithm Processes

**Create Pattern population:** The first step in GA is the creation of a random population of patterns. This will be the first generation of the algorithm. The number of individuals of the population is a parameter of the GA algorithm; greater values allow more individual genes diversity but impact performance.

The tokens to fill the slots of the individuals of the pattern population are selected randomly but according to their frequency in the RS of the experiment (see Table 2). So, if a particular token has a frequency of 0.03 in the RS, it will have a probability to be chosen of 0.03 when filling each slot of a pattern population. This is the first improvement since we will have a similar frequency of tokens and diversity in both the RS and the pattern population that will match the RS.

**Evaluate, Fitness Function:** we defined two metrics as shown in Equations (2) and (3) to evaluate the performance of a pattern in a particular Requirements Set. RM counts then the number of requirements that a pattern match from the TS. RM value is used to order patterns. The best patterns are those with higher RM values. If several patterns have the same RM, then the second metric, SM is used to rank them. SM counts the total number of tokens matched by the pattern, considering the entire RS. Figure 11 shows an example of fitness calculation for a pattern against an RS of 2 requirements.

$$\text{RM} = \text{Count of Requirements matched by the pattern} \quad (2)$$

$$SM = \sum_{req=1}^{req=numreq} \sum_{slot=1}^{slot=reqslots} \alpha$$
$$\alpha = 1 \; if \; token \; <> \; wildcard - \alpha = 0 \; if \; token = wildcard \tag{3}$$

Note that wildcards do not contribute to SM.

| | | | | | | | RM | SM |
|---|---|---|---|---|---|---|---|---|
| Pattern | 1130 | 1247 | * | | 1350 | | | |
| Requirement 1 | 1130 | 1247 | 1142 | 1112 | 1350 | | 1 | 3 |
| Requirement 2 | 1130 | 1247 | 1142 | 1112 | 1130 | 1350 | 0 | 2 |
| Total Pattern Fitness | | | | | | | 1 | 5 |

**Figure 11.** Calculation of Pattern Fitness for a particular RS of 2 requirements (Affinity = 2). Asterisk represents a wildcard token as described in Section 3.1.5.

**Select Pool:** Selecting parents for the mating pool is done with a tournament methodology. Some parametrization is done for this process, allowing to modulate if only the better parents are selected, or several individuals with less performance (less Fitness) may also be selected. This will permit to tune the variability in the descendants and to manage convergence velocity. If only better individuals are chosen, the algorithm will converge quickly, losing other evolutive possibilities. The process begins with a random selection of a candidate parent. This candidate will be compared with several other randomly chosen candidates. Only the one with the best Fitness will be selected.

**Breed:** Breeding is made by interchanging random genes (slots) between parents. In this experiment, each pair of parents generate a couple of descendants as shown in Figure 12.

| | | | | | |
|---|---|---|---|---|---|
| Parent – Pattern 1 | 1130 | 1247 | 1142 | 1112 | 2228 |
| Parent – Pattern 2 | [2240] | 1119 | * | 1144 | [1350] |
| Descendant – Pattern 1 | [2240] | 1247 | 1142 | 1144 | 2228 |
| Descendant – Pattern 2 | 1130 | 1119 | * | 1112 | [1350] |

**Figure 12.** Sample breeding process. Asterisk represents a wildcard token as described in Section 3.1.5.

**Mutate:** In this experiment, we have defined four kinds of mutations:

- Change of token: Change of token (allele) in one slot of the pattern
- Change of optionality characteristic of a gene: A gene slot, changes its optionality becoming optional or not (opposite of the current situation)
- Increase of size: Increase the number of slots of the individual, adding a random number of new slots.
- Decrease of size: Decrease the number of slots of the individual, extracting slots at the end of the individual.

These mutations are carried out on the descendants after the breed process. The number of mutations in each generation is modulated using the mutation level parameter.

**Replace:** The new generation members stored in the mating pool replace the members with less Pattern Fitness, following the steady-state technique. This technique deletes the members with the worst fitness value. This guarantees that the best individuals

are maintained and can be selected again in the next generations. A new generation is then created.

**Exit:** The GA stops if one of these three circumstances occurs:

- All Requirements have been matched,
- The maximum number of generations as specified by a specific parameter is reached,
- After a specified number of consecutive generations without a significant increase in population fitness. The number of consecutive generations and the minimum increase of fitness are GA parameters.

#### 3.2.3. Separate-and-Conquer Algorithm

The strategy of Separate-and-Conquer has been implemented in our experiment using the algorithm shown in Figure 8.

**Save Pattern in RPS:** In a particular iteration, only the best pattern from those found by the genetic algorithm, the one that matches the major number of requirements, is used to conquest, and stored in the RPS. The rest of the patterns are rejected. There is a parameter to define the minimum number of requirements that a pattern must match to be considered as a real solution. Thus, if the genetic algorithm does not generate any pattern matching at least a number equal to this parameter in a conquest iteration, no pattern will be stored as a solution in that iteration.

**Conquest Requirements:** The requirements matched by the best pattern are marked as conquest and will not be used in future genetic loops; only the remaining requirements will be used in the rest of the iterations.

**Update alleles library:** Using only the remaining requirements, a new calculation of alleles relative frequency is done. This facilitates the searching of new patterns since the relative frequency is used to create new patterns populations: the filling of tokens when creating a new population is done by a probabilistic function that considers the relative frequency of each allele.

**Stop condition:** Stop conditions are

- All Requirements have been matched,
- The maximum number of iterations as specified by a specific parameter is reached,
- After a specified number of consecutive iterations without finding any pattern. This number is also a parameter.

#### 3.2.4. Algorithms Main Parameters

Table 3 shows the main parameters defined for the genetic and conquest algorithms. These parameters may be changed depending on the RS or the scenario of pattern search to fine-tune algorithm behavior.

##### Scenarios Depending on the Objective

Affinity defines how specific a given pattern is. If we want more generalist patterns we will increase this value, allowing authors to include more concepts or terms in the requirements authoring process. This may be useful in the first steps of requirements definitions (user requirements, global requirements). For more specific requirements (e.g., software, architecture, or design requirements) good patterns must be more specifics. To get that, the Affinity parameter must be lower.

Minimum Reqs to Match parameter will determine the accuracy of the patterns found. Low parameter values (e.g., 2) will allow greater and very accurate RPS. This may be interesting in the final steps of building a god RPS.

**Table 3.** Main GA parameters.

| Parameter | Description |
|---|---|
| Affinity | Number of slots of the requirements that are allowed to match by a wildcard token of a token. |
| Generations | Maximum number of generations for the GA algorithm. |
| Maximum Pattern size | Maximum number of slots of the Patterns generated by GA. |
| Maximum Reqs size | Maximum number of words of the requirements to be analyzed. The number of words of the requirement. This is the main quality metric identified in previous works [5,30]. |
| Minimum Reqs to Match | Number of requirements that a pattern found with the GA shall match to be included in the final RPS.<br>If this minimum is not reached by any pattern generated by the genetic algorithm, then the conquest iteration does not get any pattern. |
| Mutation level | Probability of mutation for the entire population.<br>A Mutation level = 0.2 will cause mutations in 20% of the members of the population in each GA generation. |
| Number of conquest iterations | Maximum number iterations. It is an exit condition. |
| Pool Size | Number of members of the mating pool. It must be a number less than Pattern Population members. |
| Population size | Number of patterns (individuals) of the pattern population. |

Scenarios Depending on the Performance

The rest of the parameters affect the performance of the algorithms, i.e., the time taken to find possible solutions. High values for Generation, Population, Pool, and Maximum Pattern size parameters will decrease performance. However, if we significantly lower them, it is likely that the algorithm will not find valid solutions.

Mutation level and Pool size also affect the convergence speed, that is, the number of generations a particular iteration gets for the bests results and the ability to explore more solutions of the solution spectrum.

### 3.2.5. Metrics for Measure the Quality of the Solution

We will use the quality metrics listed in Table 4.

**Table 4.** Quality Metrics.

| Metric | Description | Formula | Good Quality Represented by |
|---|---|---|---|
| *AvRm* | Average of Reqs matched by the Patterns of RPS | $AvRm = \frac{\sum_{p=1}^{Pat.\ in\ RPS} RqMatched_p}{Count\ of\ Pat.\ in\ RPS}$ | High value |
| *%Rm* | Percentage of Requirements of RS matched by RPS | $\%Rm = \frac{\sum_{p=1}^{Pat.\ in\ RPS} RqMatched_p}{Count\ of\ Reqs.\ in\ RS} \times 100$ | High value |
| *BPRm* | Number of Reqs matched by the Best Pattern of RPS | $BPRm = MAX(RqMatched_p)$ | High value |

*AvRm* is the main ratio used to evaluate the quality of a solution. The worst solution will have an *AvRm* value equal to 1; that means that we have found one pattern for every requirement, so no improvement has been made from the initial situation. High values mean that we have got generalist patterns capable of matching several requirements. This is

a good situation for authoring at least in the initial phases, where flexibility is needed. This metric will be compared with the same metric obtained in the previous experiment.

*%Rm* is the percentage of Requirements conquest by the RPS found. We will try to maximize this metric. Nevertheless, it's difficult to get a value of 100% since, in every RS we find some quantity of "unique" requirements, that is, requirements that do not fill any kind of template or boilerplate. If this value is too low (below 70%), the RS may include a too wide range of vocabulary or there is more than one domain or type of requirements in the RS.

*BPRm* defines how many requirements are matched by only one pattern, the best one found in a particular RPS; this is a good metric of the Genetic Algorithm performance since the fitness function of the GA is based on this value for the evaluation and pool selection process. Therefore, when tunning GA parameters we will try to maximize this value. Total value depends on the RS analyzed and others GA parameters, especially Affinity Parameter; values greater than 4 should be expected in any case.

## 4. Experiments

### *4.1. Problem Statement*

The problem to solve is to obtain a Requirements Pattern Set (RPS) automatically that matches the maximum number of Requirements of a Requirement Set (RS) for a particular domain. If the RS is representative, then the RPS found can be used as the set of patterns for requirements authoring for that domain using the KM RAT tool.

The quality of the solutions will be measured using the metrics defined in Section 3.1.3. *AvRm*, the ratio between the number of Patterns found and the total number of the Requirement set, will be used to compare the performance of our method against the previous experiment.

### *4.2. Case Study and Data Sets*

We have used as a case study, the same corpus of requirements used in previous work (see Chapter 2). These requirements are a subset of a collection of requirements provided by INCOSE (International Council on Systems Engineering). They were previously tagged by experts as good or badly written. They were also classified by domain or type of requirements (hw/sw requirements, functional requirements, maintenance requirements, etc.).

For our experiments, we have only used the subset of requirements tagged as good written. They represent a universe of 545 requirements classified in 10 Requirements sets according to their domain. Table 5 lists the Requirements Sets.

**Table 5.** Requirements Sets are used as inputs to the experiment.

| RS Id | RS Name | Number of Requirements | Content |
|---|---|---|---|
| 3 | Hwsw_1 | 45 | Hardware and base software requirements |
| 41 | Funct_Gen1 | 34 | Functional requirements |
| 42 | Funct_SMG | 77 | Functional requirements |
| 43 | Funct_STS | 47 | Functional requirements |
| 44 | Funct_Gen2 | 101 | Functional requirements |
| 45 | Funct_QAR | 60 | Functional and Quality requirements |
| 46 | Funct_QM | 65 | Functional and Quality requirements |
| 47 | Funct_SMG | 75 | Functional requirements |
| 48 | Hwsw_2 | 23 | Hardware and base software requirements |
| 49 | SEC | 18 | Security requirements |
| | TOTAL | 545 | |

The requirements provided by INCOSE are protected by confidentiality, so in the sample tables or figures shown in this paper, we use alternative textual requirements.

We have employed the same database obtained with the KM indexer process in the previous experiment; that guarantees the same input data, the same tokenization, and syntactical classification of the text words of the requirements. This approach will allow us to compare our results with the ones obtained using the count of the frequency of pairs of consecutive tokens and substitution.

### *4.3. Experiment Results*

#### 4.3.1. Example

Figure 13 shows a real snapshot of the software developed for the experiment showing how a pattern matches several requirements from a requirement set.

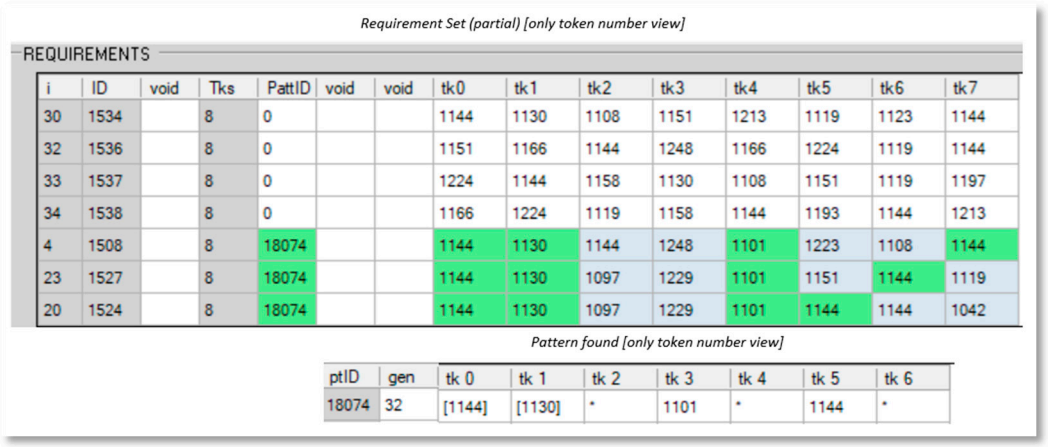


**Figure 13.** Snapshot of a real experiment using the software developed. This is the result of just one conquest iteration. Pattern ID 18074 matches Requirements Id 1508, 1527, and 1524 of the RS. Asterisks represent wildcard tokens as described in Section 3.1.5.

In this example, the pattern with ID 18074 has been found in generation 32 of the genetic algorithm and has conquest 3 requirements. In the snapshot, green slots are slots matched by the pattern with the same token value and blue slots are slots "matched" by wildcard tokens (token 2, 4, and 6 in the pattern). White requirements are requirements not conquest by text pattern found.

This is just an example of conquest iterations. In real experiments, we have used a maximum of 300 generations and 50 conquest iterations (see Table 6).

**Table 6.** Main GA parameters used in experiments.

| Parameter | Value |
|---|---|
| Population size | 50 to 80 |
| Pool size | 20 |
| Mutation level | 25% |
| Minimum Reqs to Match | 2 |
| Affinity | 3 |
| Conquest iterations | 50 |
| Generations | 300 |
| Maximum Reqs size | 12 |

#### 4.3.2. Summary of Results

Table 7 summarizes the experiments and the results. The table shows the best results of several attempts done with every RS. The algorithm uses a repetitions parameter that determines how many times the same experiment is done. The GA is not deterministic; that means that we can get different solutions in different attempts with the same inputs and with the same parameterization. The repetitions will determine the overall time taken by

each experiment. Since the final goal of the experiment is trying to evaluate the goodness of the solution, with no attention to time performance, this has not been taken in care. Future works may be done to optimize this. All these experiments have had the same GA parameters listed in Table 6.

**Table 7.** Summary results.

| Exp. Id | RS | RS Total Reqs. | Reqs. Conquest | *%Rm* | RPS | *AvRm* | *BPRm* |
|---|---|---|---|---|---|---|---|
| 1 | Hwsw_1 | 45 | 41 | 91% | 9 | 4.56 | 11 |
| 2 | Funct_Gen1 | 34 | 25 | 74% | 7 | 3.57 | 6 |
| 3 | Funct_SMG | 77 | 73 | 95% | 20 | 3.65 | 15 |
| 4 | Funct_STS | 47 | 47 | 100% | 14 | 3.36 | 14 |
| 5 | Funct_Gen2 | 101 | 93 | 92% | 22 | 4.23 | 17 |
| 6 | Funct_QAR | 60 | 52 | 87% | 10 | 5.20 | 15 |
| 7 | Funct_QM | 65 | 58 | 89% | 14 | 4.14 | 9 |
| 8 | Funct_SMG | 75 | 54 | 72% | 12 | 4.50 | 16 |
| 9 | Hwsw_2 | 23 | 17 | 74% | 5 | 3.40 | 6 |
| 10 | SEC | 18 | 12 | 67% | 4 | 3.00 | 5 |
| | TOTAL | 545 | 472 | 87% | 117 | 4.03 | |

We have got averages of Requirements conquest (*%Rm* values) ranging from 67% to 100%. As the Minimum Reqs Matched parameter is set to 2, the requirements not matched are, in any way, unique requirements; if we would want to match these requirements, we will have to build patterns specifics for them, manually.

We have also got very good averages of requirements matched per pattern (*AvRm* metric) in each experiment (from 3.00 to 5.20). The last column of the table shows the number of requirements matched by the best pattern of the RPS found in each experiment. Values rank from 5 to 15 requirements matched by the same pattern. The variations are related to GA performance and also with the variability of the requirements in each domain. Domains more accurate (in the number of vocabulary terms and size of requirements) show better values of *BPRm* and higher values of *AvRm*, meaning that it is possible to characterize the entire scenario with fewer patterns.

We can compare our results with the previous experiment. through global values and *AvRm* metric. To compare both results we will do a correction to the global RPS obtained in our experiments. This correction consists in adding a pattern for each requirement not matched, to have the same Requirements achieved inf both cases (See Table 8).

**Table 8.** Correction of RPS found in our experiments. This allows a comparison of our experiment with the previous one.

| | RS | Reqs. Conquest | RPS | *AvRm* |
|---|---|---|---|---|
| TOTAL GA Experiments | 545 | 472 | 117 | 4.03 |
| TTOTAL GA—RPS corrected | 545 | 545 | 190 [1] | 2.87 |
| Previous experiment | 545 | 545 | 442 | 1.23 |

[1] We add a new pattern for each requirement not matched by GA algorithms. This allows to compare results with the previous experiment.

In the previous experiment, they needed 442 patterns to match 545 requirements, with an average of 1.23 requirements matched per pattern. In our experiment, once we correct the results to get the same requirements conquest, we have got a total of 190 patterns to match the same 545 requirements. Our average of requirements matched per pattern *AvRm* is 2.87 which represents an improvement of 233% of the results obtained in the previous experiment.

#### 4.3.3. Algorithm Running Time

Table 9 shows information about the average running time of each experiment and the average time needed to get a new valid pattern. As performance has not been the goal of this experiment future work will be done to improve performance based on GA parameters (Generations, Pool, and Population size).

**Table 9.** Running time of the experiments.

| Exp. Id | RS | RS Total Reqs. | Reqs. Conquest | Duration (mins) | Average Time to Find Each Pattern (mins) |
|---|---|---|---|---|---|
| 1 | Hwsw_1 | 45 | 41 | 20.92 | 2.32 |
| 2 | Funct_Gen1 | 34 | 25 | 16.04 | 2.29 |
| 3 | Funct_SMG | 77 | 73 | 47.23 | 2.36 |
| 4 | Funct_STS | 47 | 47 | 20.97 | 1.50 |
| 5 | Funct_Gen2 | 101 | 93 | 48.56 | 2.21 |
| 6 | Funct_QAR | 60 | 52 | 21.43 | 2.14 |
| 7 | Funct_QM | 65 | 58 | 25.02 | 1.79 |
| 8 | Funct_SMG | 75 | 54 | 24.67 | 2.06 |
| 9 | Hwsw_2 | 23 | 17 | 7.59 | 1.52 |
| 10 | SEC | 18 | 12 | 10.83 | 2.71 |

#### 4.3.4. Impact of Requirements Size and Affinity Parameters

Requirement size, that is, the number of words in a requirement of a requirement has been identified in previous studies [30] as the main metric to determine the quality of requirements. Lower size requirements are in general best-defined requirements.

This parameter also has a major impact in searching text patterns using our methodology. Table 10 shows how the maximum size of the requirements parameter impacts the number of patterns found. In general, beyond 12 words it is difficult to find good patterns.

**Table 10.** Impact of Requirements size and Affinity Parameter in global results using HWsw_1 RS.

| | | Affinity = 3 | | Affinity = 4 | |
|---|---|---|---|---|---|
| Num. of Reqs. [1] | Maximum Req. Size [2] | Reqs Conquest | RPS | Reqs Conquest | RPS |
| 45 | 12 | 40 | 5 | 45 | 5 |
| 45 | 13 | 40 | 5 | 45 | 4 |
| 45 | 14 | 21 | 6 | 41 | 4 |
| 45 | 15 | 19 | 4 | 38 | 5 |
| 45 | 16 | 12 | 3 | 35 | 4 |

[1]—Hwsw_1 Reqs Set. [2]—Number of words.

In the experiments, we have used the maximum reqs size parameter to truncate requirements with a higher number of words.

The affinity parameter constitutes the second most important parameter. It has a great impact on the GA performance, as shown in Table 10. To determine the adequate value for this parameter, the scenario shall be considered.

## 5. Conclusions and Future Works

This paper proposes a complete methodology to get text patterns automatically from sets of requirements. The method, based on combining Genetic Algorithms and the separate-and-conquer strategy, is a novel approach. We have got good results in terms of patterns found using different Requirements Sets belonging to different domains. In all the cases, the algorithm has found textual patterns matching a significant number of requirements. Compared with previous works we have got improvement up to 233%.

Specialists may use the methodology to get textual patterns, instead of the heavy process of building them manually.

We consider the parametrization of the algorithm as an important strength of the solution since it allows to change how the algorithm will behave depending on the objective. The algorithm has 14 main parameters that change how the genetic and the divide-and-conquer will work. This gives high flexibility to the methodology that can be used in different authoring scenarios or even being used for other purposes, such as text searching or classification.

*Future Works: New Scenarios with Different Parameters Values*

Based on the variation of Affinity and Minimum Reqs to Match parameters, future works will be done comparing the two main scenarios described in Table 11 and using the same Requirements Sets.

**Table 11.** Scenarios for using the methodology in future works.

| Scenario | Goal | Parameter's Value |
|---|---|---|
| 1—Introducing authoring tools. | To have a generalist RPS that allow authoring with relative high freedom in a particular domain (fewer specific requirements) | Affinity—High<br>Minimum match—High |
| 2—Using authoring tools in a very restricted environment where precision is required | To have a very specific RPS that conduct authoring very narrowly (e.g., design phase) | Affinity—Low<br>Minimum match—Low |

**Author Contributions:** Conceptualization, J.P. and V.M.; Formal analysis, J.P. and V.M.; Funding acquisition, A.F.; Methodology, J.P., V.M. and A.F.; Software, J.P.; Supervision, J.M.Á.-R.; Writing—original draft, J.P.; Writing—review and editing, A.F. and J.M.Á.-R. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research has received funding from RESTART project: "*Continuous Reverse Engineering for Software Product Lines*" (ref. RTI2018-099915-B-I00). Convocatoria de Proyectos de I + D + i Retos Investigación del Programa Estatal de I+D+i orientada a los Retos de la Sociedad 2018. Grant agreement num. 412122. ECSEL18: Project New Control (Project 6221/31/2018) and National PCI funding nº 449990.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. McDermott, T.; DeLaurentis, D.; Beling, P.; Blackburn, M.; Bone, M. AI4SE and SE4AI: A Research Roadmap. *Insight* **2020**, *23*, 8–14. [CrossRef]
2. Micouin, P. *Model-Based Systems Engineering: Fundamentals and Methods*; iSTE: London, UK; Wiley: Hoboken, NJ, USA, 2014.
3. Alvarez-Rodríguez, J.M.; Mendieta, R.; Moreno, V.; Sánchez-Puebla, M.Á.; Llorens, J. Semantic recovery of traceability links between system artifacts. *Int. J. Softw. Eng. Knowl. Eng. (IJSEKE)* **2020**, *30*, 1–28. [CrossRef]
4. Dick, J.; Hull, E.; Jackson, K. *Requirements Engineering*; Springer International Publishing: Cham, Switzerland, 2017. Available online: http://link.springer.com/10.1007/978-3-319-61073-3 (accessed on 4 November 2021).
5. Génova, G.; Fuentes, J.M.; Morillo, J.L.; Hurtado, O.; Moreno, V. A framework to measure and improve the quality of textual requirements. *Requir. Eng.* **2011**, *18*, 25–41. [CrossRef]
6. Reuse Company, "Systems Engineering Suite". Available online: https://www.reusecompany.com/systems-engineering-suite (accessed on 10 October 2020).
7. Mitchell, M. *An Introduction to Genetic Algorithms*; MIT Press: Cambridge, MA, USA, 1996.
8. Furnkranz, J. Separate-and-Conquer Rule Learning. *Artif. Intell. Rev.* **1999**, *13*, 3–54. [CrossRef]
9. Parra, E. Metodología Orientada a la Optimización Automática de la Calidad de los Requisitos. Ph.D. Thesis, Universidad Carlos III de Madrid, Madrid, Spain, 2016. Available online: https://e-archivo.uc3m.es/handle/10016/23936 (accessed on 1 August 2019).
10. Ebert, C. 50 Years of Software Engineering: Progress and Perils. *IEEE Softw.* **2018**, *35*, 94–101. [CrossRef]

11. Kotonya, G. *Requirements Engineering: Processes and Techniques*; J. Wiley: Chichester, UK; New York, NY, USA, 1998.
12. Hogan, A.; Blomqvist, E.; Cochez, M.; d'Amato, C.; de Melo, G.; Gutierrez, C.; Gayo, J.E.L.; Kirrane, S.; Neumaier, S.; Polleres, A.; et al. Knowledge Graphs. 2020. Available online: https://www.morganclaypool.com/doi/10.2200/S01125ED1V01Y202109DSK022 (accessed on 9 November 2021).
13. INCOSE. Systems Engineering Vision 2020. 2020. Available online: https://www.incose.org/incose-member-resources/chapters-groups/ChapterSites/blues/chapter-news/2009/02/23/se-vision-2020 (accessed on 23 November 2021).
14. Mavin, A.; Wilkinson, P.; Harwood, A.; Novak, M. EARS (Easy Approach to Requirements Syntax). In Proceedings of the IEEE International Conference on Requirements Engineering, Atlanta, GA, USA, 31 August–4 September 2009; pp. 317–322. [CrossRef]
15. Schmaal, R.J.L.; Balsters, H.; Valera, S. Formal Specification of a Meta Hierarchical Logical Data Model Using Object Role Modeling. In *On the Move to Meaningful Internet Systems: OTM 2011 Workshops, Crete, Greece, 17–21 October 2011*; Meersman, R., Dillon, T., Herrero, P., Eds.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 7046, pp. 370–379. Available online: http://link.springer.com/10.1007/978-3-642-25126-9_48 (accessed on 9 November 2021).
16. Alvarez-Rodríguez, J.; de Pablos, P.; Vafopoulos, M.; Labra-Gayo, J. Towards a Stepwise Method for Unifying and Reconciling Corporate Names in Public Contracts Metadata: The CORFU Technique. In Proceedings of the Research Conference on Metadata and Semantic Research, Thessaloniki, Greece, 19–22 November 2013; Garoufallou, E., Greenberg, J., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2013; Volume 390, pp. 315–329. [CrossRef]
17. Bommasani, R.; Hudson, D.A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M.S.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. On the Opportunities and Risks of Foundation Models. *arXiv* **2021**, arXiv:2108.07258.
18. Rothman, D. *Transformers for Natural Language Processing: Build Innovative Deep Neural Network Architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and More*; Packt Publishing: Birmingham, UK, 2021.
19. Halpin, T.; Curland, M. Automated verbalization for ORM 2. In Proceedings of the OTM Confederated International Conferences on the Move to Meaningful Internet Systems, Montpellier, France, 29 October–3 November 2006; pp. 1181–1190.
20. Moreno, V.; Génova, G.; Parra, E.; Fraga, A. Application of machine learning techniques to the flexible assessment and improvement of requirements quality. *Softw. Qual. J.* **2020**, *28*, 1645–1674. [CrossRef]
21. Mallawaarachi, V. Introduction to Genetic Algorithms—Including Example Code. *Toward Data Sci. Medium* **2017**. Available online: https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3 (accessed on 23 July 2018).
22. Khanali, M.; Ahmadzadegan, S.; Omid, M.; Nasab, F.K.; Chau, K.W. Optimizing layout of wind farm turbines using genetic algorithms in Tehran province, Iran. *Int. J. Energy Environ. Eng.* **2018**, *9*, 399–411. [CrossRef]
23. Tonella, P.; Susi, A.; Palma, F. Interactive requirements prioritization using a genetic algorithm. *Inf. Softw. Technol.* **2013**, *55*, 173–187. [CrossRef]
24. Bartoli, A.; de Lorenzo, A. Learning text patterns using separate-and-conquer genetic programming. In Proceedings of the European Conference on Genetic Programming, Copenhagen, Denmark, 8–10 April 2015; pp. 16–27. [CrossRef]
25. Karova, M.N.; Petkova, J.; Penev, S.P. Web Application of Traveling Salesman Problem using Genetic Algorithms. In Proceedings of the Papers, Volume 2, XLII International Scientific Conference on Information, Communication and Energy Systems and Technologies ICEST, Ohrid, Macedonia, 24–27 June 2007; pp. 24–27. Available online: http://rcvt.tu-sofia.bg/ICEST2007_2_99.pdf (accessed on 27 January 2021).
26. Stoltz, E. Evolution of a salesman: A complete genetic algorithm tutorial for Python. *Toward Data Sci. Medium* **2018**. Available online: https://towardsdatascience.com/evolution-of-a-salesman-a-complete-genetic-algorithm-tutorial-for-python-6fe5d2b3ca35 (accessed on 18 February 2021).
27. Jurafsky, D.; Martin, J.H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, 2nd ed.; Pearson Prentice Hall: Upper Saddle River, NJ, USA, 2009.
28. Harsu, M. A Survey on Domain Engineering. Available online: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.197.7897&rep=rep1&type=pdf (accessed on 12 September 2020).
29. Chen, M.X.; Lee, B.N.; Bansal, G.; Cao, Y.; Zhang, S.; Lu, J.; Tsay, J.; Wang, Y.; Dai, A.M.; Chen, Z.; et al. Gmail Smart Compose: Real-Time Assisted Writing. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, Anchorage, AK, USA, 4–8 August 2019. [CrossRef]
30. Parra, E.; Dimou, C.; Llorens, J.; Moreno, V.; Fraga, A. A methodology for the classification of quality of requirements using machine learning techniques. *Inf. Softw. Technol.* **2015**, *67*, 180–195. [CrossRef]