Research on Test Case Generation Method of Airborne Software Based on NLP

Cong Chao, Qinghua Yang, Xiaowei Tu

Shanghai University, Shanghai, 200072, China

Abstract

Software testing is a key stage in the life cycle of airborne software development. At this stage, airborne software test cases are developed manually, so the preparation of test cases requires a lot of time and labor costs and is prone to human errors. To solve this problem, on the basis of Long Short-Term Memory, this paper proposes an airborne software test case automatic generation algorithm based on Bi-LSTM-CRF named entity recognition model and Part-Of-Speech tagging. First, preprocess the airborne software requirement document, replace the testable variable name and filter out the untestable requirement statements. Then, the airborne software domain corpus is trained through Bi-LSTM-CRF model to obtain named entity recognition model. Finally, the tag sequence is generated from the requirement statement through the named entity identification model, and the test case is generated through the triplet generation algorithm and the coverage criteria processing algorithm. The experiment uses the engine indicator software requirements document to verify the effect. The results show that compared with the traditional Bi-LSTM-CRF model, the training method with Part-Of-Speech tagging is more accurate, and the accuracy of the final test case generation can reach more than 80%.

Keywords

Airborne Software, Named Entity Recognition, Bi-LSTM-CRF, Test Case Generation

1. Introduction

Software testing is to evaluate the software according to the requirements collected from the system specifications[1]. Due to the high safety and reliability requirements of airborne software, it is very important to ensure the quality and correctness of software. In addition to strictly controlling the software development process, the software testing process is also of great significance[2]. It is estimated that software testing takes 50% of the total development cost, while testing activities consume about 40% of the overall development time[3].

The requirements-based testing process mainly solves two problems:

(1) Verify that the requirements are correct, complete, clear and logically consistent.

(2) Design necessary and sufficient test cases according to the requirements.

Requirements documents for airborne software are written in natural language, so these requirements written in natural language need to be translated into computer readable patterns to facilitate automated test case generation. NLP can transform sentences expressed in natural language into sentences that can be understood in syntax and semantics and generate corresponding test cases. At present, airborne software test cases are developed manually, but there are some serious problems in the manual development of test cases[4]. In order to improve the efficiency and effectiveness of testing, testers need to create high-quality test cases. However, writing test cases is a long and tedious task, and is prone to human errors. Therefore, we need to find a method to automatically generate high-quality test cases.

ICBASE2022@3rd International Conference on Big Data & Artificial Intelligence & Software Engineering, October 21-23, 2022, Guangzhou, China
EMAIL: chaocong1@163.com (Cong Chao); yangqinghua@shu.edu.cn (Qinghua Yang); tuxiaowei@shu.edu.cn (Xiaowei Tu)
ORCID: 0000-0003-1314-8605 (Cong Chao); 0000-0001-7084-4784 (Qinghua Yang); 0000-0003-3219-3719 (Xiaowei Tu)

This paper describes the process of automatically generating test cases from natural language requirements. The proposed method uses requirements documents as input and test case files as output.

2. Related Work

Requirements-based testing involves multiple manual processes. Among them, software testers must define test standards, design test cases according to requirements, build test cases, execute test cases and verify whether software requirements are met. If the requirements-based test process cannot run correctly or consistently, the test case may not provide the expected effect, and the testing time may increase significantly[1].

Automatic generation of test cases is not a new concept. Anurag and Shubhashis[5] developed the Litmus tool for generating test cases from requirements documents. The tool processes each requirement statement and generates one or more test cases through a five-step process. Charles et al.[6] developed an automated test case generator (ATCG), which also takes requirement statements as input and test cases as output. However, in the above methods, the test coverage generated by the requirement statement is not complete and is not applicable to the field of airborne software.

3. Implementation of Automatic Test Case Generation

In this paper, we first preprocess the requirements documents, and then train the requirements statements in the airborne software domain through natural language processing technology and Part-of-Speech tagging(POS). Then the requirement statements that need to generate test cases are generated into corresponding named entities through the Bi-LSTM-CRF named entity recognition model, and test cases are generated from standard templates through the triplet generation algorithm and coverage criteria. The test case generation process is shown in Figure 1.

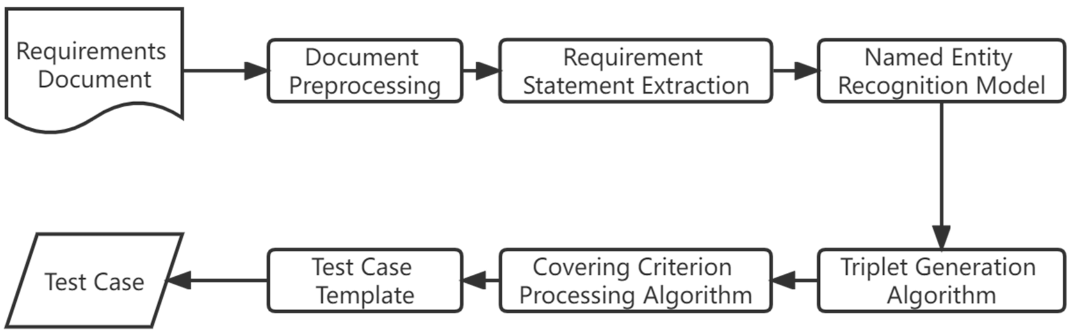
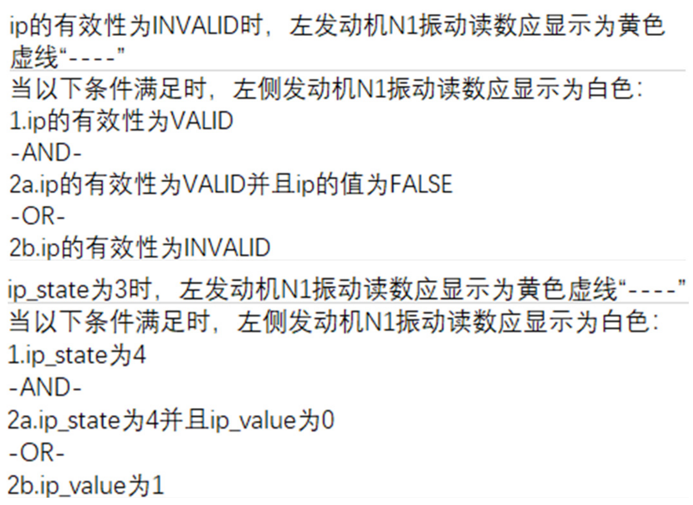

Requirements Document
Document Preprocessing
Requirement Statement Extraction
Named Entity Recognition Model
Triplet Generation Algorithm
Covering Criterion Processing Algorithm
Test Case Template
Test Case

Figure 1: Test case generation process

3.1. Document Preprocessing

Not all the requirement statements in the requirements document can be transformed into test cases for testing, and there are some explanations and definitions of terms. For example, "the engine indication software includes normal and compression modes". Therefore, untestable requirement statements need to be filtered out. A requirement is a contract that specifies what the user \ agent does to the system and how the system responds. So, a testable sentence can be defined as one that has subject, action, and optional object.

Since the constants and variables of the requirements document and the airborne software model have one-to-one correspondence relationship tables, and the test case of airborne software is to verify the assignment of software model variables, the requirements document can be filtered by extracting the constants and variables in the relationship table and replacing the constants and variable names in the requirements document. Figure 2 shows an example of the comparison before and after the requirement document preprocessing.


ip的有效性为INVALID时，左发动机N1振动读数应显示为黄色虚线“----”

当以下条件满足时，左侧发动机N1振动读数应显示为白色：
1.ip的有效性为VALID
-AND-
2a.ip的有效性为VALID并且ip的值为FALSE
-OR-
2b.ip的有效性为INVALID

ip_state为3时，左发动机N1振动读数应显示为黄色虚线“----”

当以下条件满足时，左侧发动机N1振动读数应显示为白色：
1.ip_state为4
-AND-
2a.ip_state为4并且ip_value为0
-OR-
2b.ip_value为1

Figure 2: Comparison of requirements documents before and after preprocessing

3.2. Requirement Statement Extraction

In order to process the requirements item by item, the statements in the requirements document need to be extracted separately. For testable requirements documents, they can be divided into single-line requirements and multi-line requirements. Single-line requirements are requirements with only one sentence. A multi-line requirement contains multiple statements, and the statements are connected by logical symbols such as "AND" or "OR". The last example in Figure 2 is a multi-line requirement.

For the extraction of single-line requirements, it does not require too many operations. It can be obtained directly from the requirements document. For the extraction of multi-line statements, it is necessary to divide each line and logical symbols into multiple statements for processing the statements one by one. For example, the requirements in Figure 2 can be divided into“当以下条件满足时，左侧发动机 N1 振动读数应显示为白色”, “ip_state 为 4”, “-AND-”, “ip_state 为 4 并且 ip_value 为 0”, “-OR-”, “ip_value 为 1”, and then process one by one.

3.3. Named Entity Recognition Model

The purpose of named entity recognition is to identify all entities in the requirement statement. The input of the model is the requirement statement, and the output is the named entity tagging sequence. The requirement statement is transmitted through the Bi LSTM-CRF neural network model. First, bidirectional LSTM is used for forward and backward training to obtain the output score of the tag. Then, run the CRF layer to calculate the gradient of network output and state transition edge. Finally, we update the network parameters, including the parameters of the state transition matrix and the original bidirectional LSTM. In order to improve the accuracy of named entity tagging, the following two problems should be solved:

(1) For the requirement statement, what is the label of the training model.

(2) How to improve the performance of the model.

For the first question, use the label method of SPO (subject predicate object) to tagging the target element, operation instruction and interaction information of the test case as the three types of labels of the training model. For the second problem, the Part-Of-Speech tagging is used as the feature of the training model while tagging the triplets.

The named entity recognition model proposed in this paper is based on Bi-LSTM-CRF, and the POS features are added on this basis.

3.3.1. LSTM Network Structure

Recurrent Neural Network (RNN) is a kind of neural network used to process sequence data. It is very effective for data with sequence characteristics, which enables the trained model to predict results through long distance characteristics[7].

Theoretically, RNN can learn long-term dependence, but there are defects in dealing with long-term memory, such as gradient disappearance and gradient explosion [8]. It tends to consider the recent state. Long Short-Term Memory (LSTM) adds a storage unit to RNN to filter past states, so that it can choose which states have more influence on the current situation, and better discover and utilize the dependencies in the data , instead of simply selecting a nearby state. The module at moment t of LSTM is shown in Figure 3.

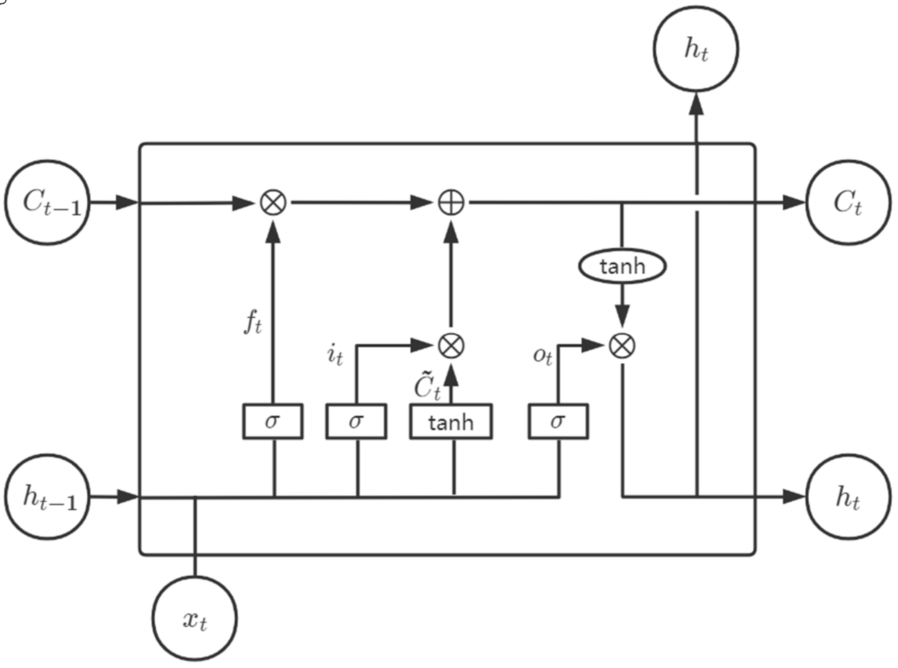


Figure 3: LSTM module at moment t

LSTM adjusts the values of input and hidden layers through the gate structure, which is composed of forgetting gate, memory gate and output gate. Among them, σ Represents a sigmoid function whose output is between 0 and 1. Tanh is a hyperbolic tangent function with values between - 1 and 1.

The forgetting gate determines the forgetting ratio in the last moment C_{t-1}, and the formula is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \tag{1}$$

The memory gate obtains the weight of the new memory through the σ layer, and then adds the weighted new memory $i_t * \tilde{C}_t$ to the existing state, and then realizes the update from C_{t-1} to C_t. The formula is as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \tag{2}$$

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C), \tag{3}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{4}$$

Finally, the short-term memory h_t is obtained by updating the output gate, and the formula is as follows:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \tag{5}$$

$$h_t = o_t * tanh\ (C_t), \tag{6}$$

Both RNN and LSTM can only predict the output of the next moment based on the previous time series information, but in the process of named entity recognition, the output is not only related to the previous state, but also related to the future state. Therefore, this paper uses bidirectional LSTM to predict named entities based on the context, takes words as the minimum unit, takes the word vector encoding sequence x_i generated through the embedding layer as the input of each moment of the LSTM, and then splices the hidden state output sequences of each position of the forward LSTM and the backward LSTM, that is, $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$. The new sequence contains both historical information and future information, which can further improve the accuracy of recognition.

3.3.2. CRF Layer

Conditional Random Field (CRF) is a distribution model that takes the input sequence as a condition and then obtains another set of conditional probabilities of the output sequence. It is widely used in word segmentation, part-of-speech tagging, and named entity recognition. Input the label distribution probability obtained through bidirectional LSTM to the CRF layer, and then output the corresponding label sequence of each word. If the CRF layer is not used as the constraint, the tag with the highest probability of each word is taken as the output when the label is output, which is easy to generate a tag sequence that does not conform to common sense. For example, the subject tag follows the predicate tag. By calculating the transition probability between tags, CRF can obviously filter out these error outputs. In this paper, CRF is used to establish the output of the whole sentence, and the CRF model is used to score the labels of words in the sentence. The tag sequence with the highest score is output. The scoring formula can be expressed as:

$$s(X,y) = \sum_{i=1}^{n} A_{y_i,y_i+1} + \sum_{i=1}^{n} P_{i,y_i}, \tag{7}$$

In Formula (7), A is the transfer matrix of (k+2)*(k+2), A_{y_i,y_i+1} is the transfer probability from tag y_i to tag y_{i+1}, where k is the number of tag categories. P is the emission matrix of n * (k+2), P_{i,y_i} is the emission probability of the tag y_i obtained from the word x_i, where n is the length of the sequence.

3.3.3. Word-based Tagging

Triplets have corresponding role components in the requirement statements of airborne software. The subject is usually the variable name in the airborne software, the predicate is generally a verb, and the object is the data related to the subject. According to the above analysis, this paper uses the POS feature and the BIO annotation method to build a neural network. By training the neural network to learn the relationship between triplets and parts of speech, the named entity recognition performance of the model can be effectively improved.

We use the jieba to tag the part of speech of the requirement statement. Since jieba supports adding custom dictionaries, we can supplement the custom dictionaries to cover more comprehensive vocabulary in specific fields and improve the accuracy of tool tagging.

BIO tagging is a kind of union tagging. Specifically, B-X represents that the element is of type X and is located at the beginning of the segment of this type, I-X represents that the element is of type X and is located at the middle or end of the segment of this type, and O represents that it is not an entity type that needs to be tagged.

3.3.4. Model Training

In the process of model training, we use one-hot coding, input the tagged words as samples, and then use the embedding layer to convert the coding into a low dimensional, dense vector to solve the feature sparse problem. To avoid overfitting during training, we add dropout[9] to the LSTM layer with the parameter set to 0.5. The optimizer chooses Adam[10], using stochastic gradient descent algorithm with a learning rate of 0.001 for 100 epochs. The Bi-LSTM-CRF training model is shown in Figure 4.

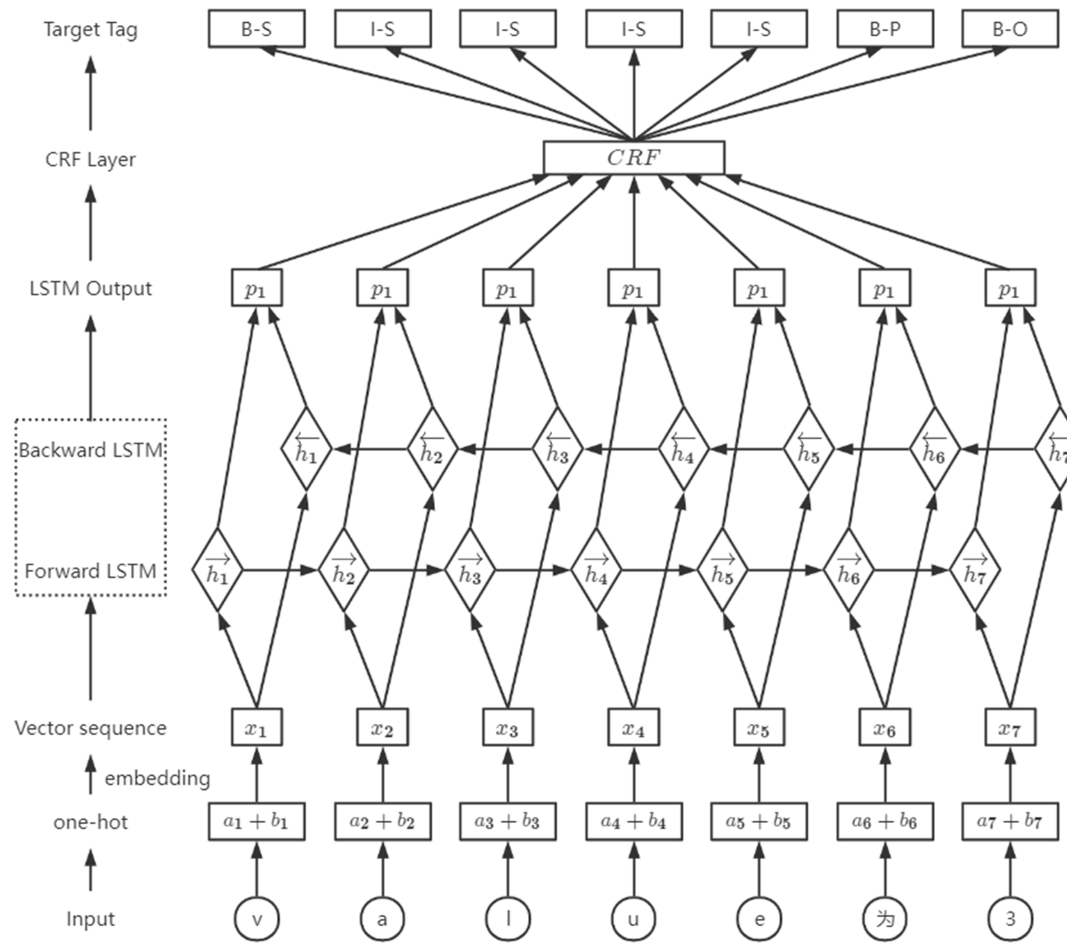

Target Tag
B-S
I-S
I-S
I-S
I-S
B-P
B-O
CRF Layer
CRF
LSTM Output
p_1
Backward LSTM
$\overleftarrow{h_1}$ $\overleftarrow{h_2}$ $\overleftarrow{h_3}$ $\overleftarrow{h_4}$ $\overleftarrow{h_5}$ $\overleftarrow{h_6}$ $\overleftarrow{h_7}$
Forward LSTM
$\overrightarrow{h_1}$ $\overrightarrow{h_2}$ $\overrightarrow{h_3}$ $\overrightarrow{h_4}$ $\overrightarrow{h_5}$ $\overrightarrow{h_6}$ $\overrightarrow{h_7}$
Vector sequence
x_1 x_2 x_3 x_4 x_5 x_6 x_7
embedding
one-hot
$a_1 + b_1$ $a_2 + b_2$ $a_3 + b_3$ $a_4 + b_4$ $a_5 + b_5$ $a_6 + b_6$ $a_7 + b_7$
Input
v
a
l
u
e
为
3

Figure 4: Bi-LSTM-CRF model

3.4. Triplet Generation Algorithm

Since the output of the model during prediction is a sequence of BIO tags, the tags need to be converted into corresponding triplets. The statements in the requirements document are "subject, predicate, object" or "subject, predicate" structures. In order to extract the triplets of requirement statements, a verb centered algorithm is established to extract the relationships between complex statements. The input of the algorithm is a requirement statement, which extracts single/multiple relationships between entities into triplets. For example, " ip_state 为 3 并且 ip_value 为 1。" The extracted triplet is: (ip_state, 为, 3), (ip_value, 为, 1).

3.5. Coverage Criteria Processing Algorithm

In view of the requirements for high safety and reliability of large aircraft, according to DO-178C, airborne software of large aircraft is divided into categories A, B, C, D and E[11], and different categories of airborne software correspond to different coverage criteria.

Since the engine instruction software is class B software and needs to satisfy the Decision Coverage (DC), this paper designs a DC-based processing algorithm.

3.5.1. Decision Coverage

The basic idea of decision coverage is to design enough test cases so that each decision in the program can obtain at least one "true" and one "false", that is, each true or false branch is executed at least once, so it is also called branch coverage.

In order to achieve decision coverage, this paper converts each requirement into two test cases, in which all parameters in one test case are true, and in the other test case, all parameters are false. Since in multiple requirement statements, there may be two statements connected by "OR" with the same parameter setting different values, so this paper adopts the strategy that if the same parameter exists above, the parameter value will remain unchanged.

3.5.2. Keyword Mapping Table

Since the same semantics of the verbs in the demand statement may have multiple representation methods, for example, "is" and "equal to" both indicate setting a value, so this paper replaces the same semantic characters with keywords.

Among them, EQ, NEQ, GR, LE, GRE, and LEE correspond to equal to, not equal to, greater than, less than, greater than or equal to, and less than or equal to, respectively, which are used to replace the same semantics and facilitate the processing of triplets.

3.6. Test Case Generation

Test cases need to be generated into corresponding formats before they can be used for testing and generating test scripts. A complete test case should include the start flag, requirement number, requirement content, test case content, end flag, etc. This paper uses the method of filling the test case template to fill the requirement number, requirement content and test case content to the corresponding position.

4. Effect verification

4.1. Model effect verification

The evaluation indexes of model experimental results are P(Precision), R(Recall) and F1(F-measure)[12]. F1 is the result of the weighted calculation of Precision and Recall, which is used as the comprehensive evaluation index of the model. The calculation formulas of P, R, and F1 are:

$$P = \frac{Number\ of\ correctly\ recognized\ tags}{Number\ of\ tags\ recognized} \times 100\%, \tag{8}$$

$$R = \frac{Number\ of\ correctly\ recognized\ tags}{Total\ number\ of\ tags} \times 100\%, \tag{9}$$

$$F1 = \frac{2 \times P \times R}{P + R} \times 100\%, \tag{10}$$

In this paper, we compare the effect of two tagging methods, BIO and BIO-POS, in identifying triplets of requirement statements. It can be seen from Table 1 that the recall rate of BIO-POS tagging method is significantly higher than that of traditional BIO tagging method, which indicates that BIO-POS tagging method has better performance in the task of identifying triplets of airborne software requirements.

Table 1
Comparison of recognition effects under different tagging methods

Tagging	Precision	Recall	F1-measure
BIO	91.18%	92.53%	91.85%
BIO-POS	90.46%	96.18%	93.23%

4.2. Test case generation effect verification

In this paper, representative test case results are selected for effect verification, and the generation effect is shown in Table 2. In order to save the length of the article, positive test cases are intercepted for each requirement.

Table 2
Results of some test cases generated

Requirement	**1. 当 ipFL 有效性为 VALID 时，左发动机模拟表盘的红线标记应显示为白色。**		
Test Case	SET	ipFL_state	3
	VERIFY	左发动机模拟表盘的红线标记应显示为白色	
Requirement	**2. 当 ipEC 有效性为 VALID，并且 ipEC 的值为 TRUE 时，发动机显示软件应为压缩模式。**		
Test Case	SET	ipEC_state	3
	SET	ipEC_value	1
	VERIFY	发动机显示软件应为压缩模式	
Requirement	3. 当以下条件满足时，左发动机 N1 指针应显示为白色： 1. ipFL 的有效性为 VALID -AND- 2a. ipFE 的有效性为 INVALID -OR- 2b. ipFE 的有效性为 VALID，并且 ipFE 的值为 TRUE		
Test Case	SET	ipFL_state	3
	SET	ipFE_state	4
	SET	ipFE_value	1
	VERIFY	左发动机 N1 指针应显示为白色	
Requirement	4. 当涉及显示迟滞的 ipFL 的值增加时，左发动机指针应顺时针旋转。		
Test Case	Test case generation failed		
Requirement	5. 当 N1 值等于或大于 100%时，N1 指示应以 XXX 显示。		
Test Case	SET	N1_value	100
	VERIFY	左发动机指针应顺时针旋转	
Requirement	6. 当 ipFRT 的有效性为 VALID 且 1<=ipFRT 的值<=11，对应推力模式应显示。		
Test Case	SET	ipFRT_state	3
	SET	ipFRT_value	11
	VERIFY	对应推力模式应显示	
Requirement	7. 当 ipHP 的有效性为 VALID，并且 ipHP 的值在[33°，35°]范围内时，襟翼卡位应显示为 4。		
Test Case	SET	ipHP_state	3
	SET	ipHP_value	34
	VERIFY	襟翼卡位应显示为 4	
Requirement	8. ipFC 的有效性变为 INVALID 且保持 1.2s，通信标志 FMS 应不显示。		
Test Case	SET	ipFC_state	4
	WAIT	1.2	
	VERIFY	通信标志 FMS 应不显示	

From the results, we can see that most of the requirement statements have a good conversion effect. In Requirement 5, since there is no reference value for the relevant parameters of a single statement, the corresponding value cannot be set. The next step will be to set the initial value or contact the context to solve such problems.

Through the analysis of the requirements document, it can be found that the forms in Table 3 can cover more than 80% of the testable requirement statements, that is to say, as long as the relevant generation algorithms are processed well, most of the conversion results can be generated correctly.

5. Conclusions

This paper explores the application of automatic test case generation method based on NLP in the field of airborne software. The strong robustness of Bi LSTM-CRF named entity recognition model and the ability to effectively use past and future features are comprehensively considered. Aiming at the specific corpus in the field of airborne software, the BIO-POS tagging method is used to train the model, and a good effect of named entity recognition is obtained. According to the results of named entity recognition, a verb-centered triplet generation algorithm and a triplet-based coverage criterion processing algorithm are proposed. Experiments show that the correct rate of the test cases generated by the algorithm in this paper is more than 80%. However, when the variables in some sentences do not have corresponding reference values or the sentence patterns are special, the method in this paper will not be able to identify effectively. Therefore, the next step is to study the processing of requirement statements that need to contact the context to obtain initial values and special sentence patterns, so as to further improve the generation effect of test cases.

6. Acknowledgements

First of all, I would like to give my heartfelt thanks to all the people who have ever helped me in this paper. My sincere and hearty thanks and appreciations go firstly to my supervisor, Mr. Yang Qinghua, whose suggestions and encouragement have given me much insight into these studies.

In addition, thanks to COMAC Shanghai Aircraft Design and Research Institute and my corporate mentor Mr. Yi Zichun. Thank you for providing me with research direction and experimental materials.

Finally, I am really grateful to all those who devote much time to reading this thesis and give me much advice, which will benefit me in my later study.

7. References

[1] Veera, Prm , et al. "Req2Test - Graph Driven Test Case Generation for Domain Specific Requirement." (2018).

[2] Gao, Y. , G. An , and C. Zhi . Verification and validation of flight control system airborne software. 2021.

[3] Nagpal, K. , and R. Chawla . "IMPROVEMENT OF SOFTWARE DEVELOPMENT PROCESS: A NEW SDLC MODEL.".

[4] P Kulkarniȧ, and Y Joglekarȧ. "Generating and Analyzing Test cases from Software Requirements using NLP and Hadoop.".

[5] Dwarakanath, A. , and S. Sengupta . "Litmus: Generation of Test Cases from Functional Requirements in Natural Language." Springer Berlin Heidelberg Springer Berlin Heidelberg, 2012.

[6] Morgan, Charles P., et al. "ATCG: An Automated Test Case Generator." Journal of Information Technology Management 27.3 (2016): 112-120.

[7] Long Qiuxian. Automatic Function Testing Algorithm Based on Deep Learning .2018.Tianjin University, MA thesis. DOI:10.27356/d.cnki.gtjdu.2018.002016.

[8] Lai, Siwei, et al. "Recurrent convolutional neural networks for text classification." Twenty-ninth AAAI conference on artificial intelligence. 2015.

[9] Hinton, G. E. , et al. "Improving neural networks by preventing co-adaptation of feature detectors." arXiv e-prints (2012).

[10] Cheng Ming, et al. "Fishery standard named entity recognition with integrated attention mechanism and BiLSTM+CRF." Journal of Dalian Ocean University 35.02(2020):296-301. doi:10.16535/j.cnki.dlhyxb.2019-289.

[11] Wang, L. Z. , Y. Wang , and X. U. Zhang-Hou . "Software Verification and Validation of Tokamak Safety-critical Instrumentation and Control System." Journal of Changchun Normal University (2018).
[12] Chen Yanyu, and Du Ming. "Insurance Named Entity Recognition based on Bi-LSTM-CRF." Intelligent Computer and Applications 8.03(2018):111-114.