J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

소프트웨어 버그 정정에 SeqGAN 알고리즘을 적용☆

Applying SeqGAN Algorithm to Software Bug Repair

양 근 석[1] 이 병 정[1*]
Geunseok Yang Byungjeong Lee

요 약

최근 소프트웨어가 다양한 분야에 적용되면서 소프트웨어 규모와 프로그램 코드의 복잡성이 증가하였다. 이에 따라 소프트웨어 버그의 존재가 불가피하게 발생하고, 소프트웨어 유지보수의 비용이 증가하고 있다. 오픈 소스 프로젝트에서는 개발자가 할당 받은 버그 리포트를 해결할 때 많은 디버깅 시간을 소요한다. 이러한 문제를 해결하기 위해 본 논문은 SeqGAN 알고리즘을 소프트웨어 버그 정정에 적용한다. 자세히는 SeqGAN 알고리즘을 활용하여 프로그램 소스코드를 학습한다. 학습과정에서 공개된 유사 소스코드도 같이 활용한다. 생성된 후보 패치에 대한 적합성을 평가 하기 위해 적합도 함수를 적용하고, 주어진 모든 테스트 케이스를 통과하면 소프트웨어 버그 정정이 되었다고 본다. 제안한 모델의 효율성을 평가하기 위해 베이스라인과 비교하였으며, 제안한 모델이 더 잘 정정하는 것을 보였다.

☞ 주제어 : SeqGAN 알고리즘, 버그 리포트, 소프트웨어 버그 정정, 소프트웨어 유지보수

ABSTRACT

Recently, software size and program code complexity have increased due to application to various fields of software. Accordingly, the existence of program bugs inevitably occurs, and the cost of software maintenance is increasing. In open source projects, developers spend a lot of debugging time when solving a bug report assigned. To solve this problem, in this paper, we apply SeqGAN algorithm to software bug repair. In detail, the SeqGAN model is trained based on the source code. Open similar source codes during the learning process are also used. To evaluate the suitability for the generated candidate patch, a fitness function is applied, and if all test cases are passed, software bug correction is considered successful. To evaluate the efficiency of the proposed model, it was compared with the baseline, and the proposed model showed better repair.

☞ keyword : SeqGAN Algorithm, Bug Report, Software Bug Repair, Software Maintenance

1. 서 론

소프트웨어가 다양한 산업에 적용되면서 소프트웨어 규모와 프로그램 소스코드의 복잡성이 증가하였다. 이에 따라 프로그램 버그가 불가피하게 발생하게 되었고 소프트웨어 유지보수의 비용 또한 증가하고 있다[1].

오픈 소스 프로젝트에서는 개발자가 할당 받은 버그 리포트에 대해 프로그램 버그 정정 과정을 다음과 같이 진행한다. 먼저 버그 리포트의 설명과 요약을 보면서, 소스코드의 어느 부분이 버기 라인인지 식별한다. 그리고 적절하게 프로그램 버그 정정을 진행한다. 만약 자동적으로 식별된 버기 라인을 정정한다면, 개발자의 개발적 생산성이 증가하고 품질 좋은 소프트웨어를 제공할 수 있다. 프로그램 버그 정정과 관련된 유사 연구는 다음과 같다.

DeepFix[2]는 C언어 기반 구문 오류 프로그램 버그 정정 과정을 진행한다. 입력된 소스코드 라인에 대해서 새로운 소스코드를 생성하고 구문 오류를 정정한다. kGenProg[3]은 Java 언어 기반 유전 프로그래밍 기법을 적용하여 프로그램 버그를 정정한다. 유전 연산자에서는 선택, 교배, 변이 과정을 포함한다. 주어진 테스트 케이스를 이용하여 생성된 프로그램 패치가 적합한지 판단한다. 하지만 프로그램 버그 정정 성능이 더욱 향상 되어야 한다.

이러한 문제를 해결하기 위해 본 논문에서는 딥러닝 알고리즘을 활용하여 프로그램 버그 정정을 진행한다. 자세히는 SeqGAN 알고리즘[1]을 활용하여 프로그램 소스코드를 학습한다. 학습과정에서 공개된 유사 소스코드도

1 Dept. of Computer Science, University of Seoul, Seoul, 02504, Korea
* Corresponding author (bjlee@uos.ac.kr)
[Received 5 July 2020, Reviewed 13 July 2020(R2 3 September 2020), Accepted 19 Septebmer 2020]
☆ 이 논문은 2020년도 서울시립대학교 교내학술연구비에 의하여 지원되었음.

같이 활용한다. 그리고 버기 소스코드 라인을 학습된 모델에 입력으로 넣어 프로그램 후보 패치를 생성한다. 생성된 후보 패치의 유효성을 평가하기 위해 적합도 함수[3]를 적용하여 프로그램 패치가 적절하게 되었는지 판단한다. 제안한 모델의 성능을 평가 하기 위해 관련 연구와 비교를 진행하였고 제안한 방법이 더 정정하는 것을 보였다.

본 논문이 기여하는 부분은 다음과 같다.

- SeqGAN 알고리즘을 소프트웨어 버그 정정에 적용하여 프로그램 버그 정정을 진행하였다.
- 제안한 모델의 효율성을 평가하기 위해 유사 연구와 비교 하였으며, 제안한 방법이 더 효율적임을 보였다.
- 개발자들의 개발적 생산성을 향상하고, 소프트웨어 유지보수의 비용을 줄인다.

본 논문의 구성은 다음과 같다. 2장에서 제안한 방법과 관련된 배경지식을 설명한다. 3장에서 관련 연구를 소개한다. 4장에서는 프로그램 버그 정정 방법을 제안하고, 5장에서 실험을 진행한다. 6장에서 실험 결과에 대한 토의를 진행하고, 7장에서 제안한 방법의 결론과 향후 연구를 소개한다.

2. 배경 지식

2.1 프로그램 버그 정정

프로그램 버그 정정에 대한 예시는 다음 그림 1과 같다. 이 프로그램은 사용자로부터 정수를 입력받아, 입력받은 정수에 1을 더하는 프로그램이다. 하지만 라인 11에서 yang_ipt 변수에 1을 더하는 것이 아니라 yang_res 변수에 1을 더하여 프로그램 버그가 발생하였다.

만약 테스트 케이스 1을 적용하면, 프로그램은 1의 결과가 출력이 된다. 따라서 주어진 테스트 케이스를 모두 통과할 수 없다. 의심스러운 버기 라인 7, 라인 11, 라인 13에 대해서 프로그램 후보 패치를 다음과 같이 생성한다.

프로그램 패치 1을 적용하면, 라인 11번에 대해 컴파일 오류가 발생하여 적절한 패치라고 보기 어렵다. 프로그램 패치 2를 적용하면 주어진 테스트 케이스를 모두 통과할 수 있어 프로그램 패치가 적절하게 되었다고 본다. 따라서 프로그램 패치 2번을 적용하여 프로그램 버그 정정 과정을 진행한다.

```
1   import java.util.Scanner;
2
3   public class JICS_Yang{
4       public static void main(String[] args){
5           int yang_res = 0;
6           Scanner yang_sc = null;
7           int yang_ipt = 0;
8
9           yang_sc = new Scanner(System.in);
10          yang_ipt = yang_sc.nextInt();
11          yang_res = yang_res + 1;
12
13          System.out.println(yang_res);
14          yang_sc.close();
15      }
16  }
```

(그림 1) 프로그램 버그 정정 예시
(Figure 1) An Example of Program Bug Repair

```
#테스트 케이스 1
입력: 1
출력: 2

#테스트 케이스 2
입력: 10
출력: 11
```

```
#프로그램 패치 1
라인 7 → int yang_ipt = 0; (제거)

#프로그램 패치 2
라인 11 → yang_res = yang_ipt + 1; (변형)

#프로그램 패치 3
라인13 → System.out.println(yang_ipt); (변형)
```

2.2 딥러닝 알고리즘

Recurrent Neural Networks (RNN)[4]에서 Sequence to Sequence (Seq2Seq)[5]는 자연어 번역에서 자주 사용되지만, 보통의 경우 학습하는데 많은 시간이 걸리며, 속도가 많이 느리다. 이후 입력에 대한 새로운 생성을 하기 위해, Generative Adversarial Network (GAN)[6] 이라는 알고리즘이 제안되었으며 다음과 같은 2가지의 모델을 구성한다. Generator는 입력에 대한 새로운 데이터 생성을 진행하고, Discriminator는 Generator가 생성한 것을 판별한다. 이 과정에서 과거 논문에서는 Generator를 지폐위조범으로 표현하고, Discriminator는 경찰에 비유한다.

3. 관련 연구

3.1 검색 기반 버그 정정

kGenProg[3]은 유전 프로그래밍 기반 Java 프로그램 버그 정정 알고리즘이다. 유전 프로그래밍에는 선택, 교차 변이 알고리즘을 포함한다. 유전 프로그래밍 기반으로 프로그램 후보 패치를 생성하고 주어진 테스트 케이스를 적용하여 적합한지 아닌지 판단한다.

Debroy et al.[7]은 표준 돌연변이(Mutation)를 사용하여 프로그램 정정을 시도했다. 정정 모델은 같은 클래스에서 연산자, 관계형, 논리적 등을 대체한다.

3.2 패턴 기반 버그 정정

PAR[8]은 프로그램 정정 템플릿 기반으로 프로그램 버그 정정을 진행한다. 이 템플릿에는 프로그램 버그 정정과정에서 발생하는 일반적인 방법이 포함되어 있다 (예: 예외 처리 등).

Demarco et al.[9]은 기존 if 조건문을 수정하거나 전제 조건을 코드 블록에 삽입하여 조건부 버그를 복구하는 기술을 제안했다. 변형 및 삽입은 만족도 모듈로 이론에 기초하여 합성된다.

3.3 학습 기반 버그 정정

Prophet[10]는 과거의 커밋 히스토리를 사용하여 패치를 생성한다. 커밋 내역은 VCS (Version Control Systems)에서 확보한다. 그리고 올바른 패치를 찾기 위해 확률 분포를 학습한다.

Yang et al.[1]은 SeqGAN 알고리즘을 활용하여 C언어 기반 구문오류 프로그램 버그 정정을 진행하였다. 그리고 연구[11, 12]에서 Java언어 기반 런타임 프로그램 버그 정정 기법을 제안하였다.

3.4 관련 연구와의 정성적 비교

관련 연구와의 정성적 비교는 다음 표 1과 같다.

Long et al.[10]은 학습 모델에 의한 특징 추출 및 C언어 기반 프로그램 패치 생성을 한다. 그러나 본 논문에서는 SeqGAN 알고리즘과 강화 학습 기반 정책을 사용하여 Java 기반 프로그램 버그를 정정한다.

Yang et al.[1, 11, 12] 연구에서도 SeqGAN 알고리즘과 강화 학습 기반 정책을 사용하지만 다음과 같은 차이점이 있다.

(표 1) 관련 연구와의 정성적 비교

(Table 1) A Qualitative Comparison with Related Works

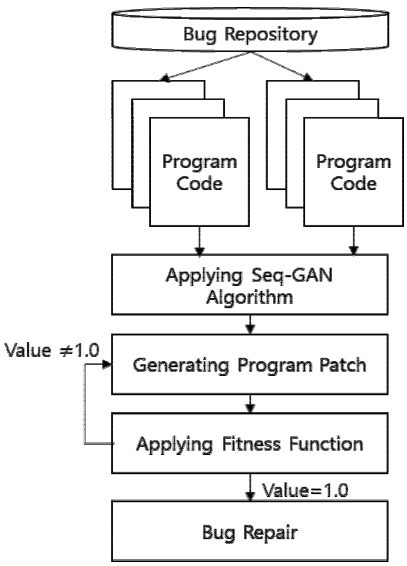
Author	Technique	Machine Learning
kGenProg [3]	Genetic Programming	X
PAR [8]	Pattern-based	X
Demarco [9]	SMT Solver	X
Long [10]	Feature Extraction	Patch Generation with Learning
Debroy [7]	Mutation	X
Yang [1, 11, 12]	Feature Extraction (Source Code)	SeqGAN, Reinforcement-Learning
Our Study	Feature Combination (Github Related Source Code, Source Code)	SeqGAN, Reinforcement-Learning

연구[1]은 C언어 기반 구문오류 프로그램 정정 기법을 제안하였지만, 본 논문은 Java언어 기반 런타임 프로그램 정정 기법을 제안하였다. 해당 연구는 구문오류가 있는 프로그램 소스코드를 학습하여 구문오류를 적절하게 정정하고, C언어 컴파일러의 에러메시지를 줄이는 것에 중점을 두었다. 하지만 본 연구는 생성된 패치에 대해 주어진 테스트 케이스를 실행하여 적합성을 평가하고 프로그램 정정이 적절하게 되었는지 판단한다. 또한 해당 연구는 컴파일 에러메시지 제거를 통해 프로그램 구문오류를 정정해서 비즈니스 로직과 원하는 결과에 대한 확인은 추가 검증이 필요하다. 하지만 본 논문은 사용자로부터 버그 프로그램과 테스트 케이스를 입력 받으면 크고/작은 버그를 자동 정정하여 개발자의 개발적 생산성을 향상하고, 소프트웨어 유지보수의 비용을 줄일 수 있는 특징을 갖고 있다.

연구[11]는 Java 언어 기반 런타임 프로그램 정정 기법을 진행하기 위한 사전 방법론적 접근을 하였고 연구[12]에서 사례 연구를 통해 검증하였지만, 본 논문은 모델 학습과정에서 공개된 유사 소스코드 활용이 프로그램 정정에 도움을 주는 것을 확인하였다. 추가적으로 통계적 검증을 통하여 제안한 방법과 베이스라인간 프로그램 버그 정정에서 유의미한 차이를 보였다.

4. 프로그램 버그 정정 기법

프로그램 버그 정정에 대한 전반적인 도식도는 다음 그림 2과 같다. 먼저 버그 저장소로부터 프로그램 소스코드를 추출한다. 추출한 소스코드를 딥러닝 알고리즘에 적용하기 위해 프로그램 변환과정을 포함한다. 변환된 프로그램 소스코드를 SeqGAN 모델에 학습을 진행한다. 의심스러운 버기 라인을 학습된 모델에 입력을 하면, 새로운 프로그램 후보 패치가 생성이 된다. 후보 패치의 적합성을 평가하기 위해 적합도 함수를 적용하고, 주어진 테스트 케이스를 모두 통과하면 해당 후보 패치를 적용하여 프로그램 버그 정정이 성공적으로 되었다고 본다.



(그림 2) 프로그램 버그 정정 도식도
(Figure 2) An Overview of Program Bug Repair

4.1 구성

프로그램 소스코드는 다양한 텍스트로 구성이 되어 있으며, 본 논문에서는 해당 텍스트를 토큰으로 간주한다. 보통의 소스코드 구성은 Type, Keyword, Semicolon, Library Function을 포함하며, 그 이외의 토큰은 사용자 토큰으로써 식별된 순서대로 저장한다.

그림 3에서, 사용자 토큰은 <yang_res, 라인 5>, <yang_sc, 라인 6>, <yang_ipt, 라인 7>, <yang_sc, 라인 9>, <yang_ipt, 라인 10>, <yang_sc, 라인 10>, <yang_res, 라인 11>, <yang_res, 라인 11>, <1, 라인 11>, <yang_res, 라인 13>, <yang_sc, 라인 14>으로 식별한다. 이 토큰들은 프로그램 후보 패치가 생성 되었을 때, 변수명 복원에서도 사용된다.

```
1   import java.util.Scanner;
2
3   public class JICS_Yang{
4      public static void main(String[] args){
5           int 〈yang_res〉 = 0;
6           Scanner 〈yang_sc〉 = null;
7           int 〈yang_ipt〉 = 0;
8
9           〈yang_sc〉 = new Scanner(System.in);
10          〈yang_ipt〉 = 〈yang_sc〉.nextInt();
11          〈yang_res〉 = 〈yang_res〉 + 〈1〉;
12
13          System.out.println(〈yang_res〉);
14          〈yang_sc〉.close();
15     }
16  }
```

(그림 3) 사용자 토큰 식별 예
(Figure 3) An Example of User Dictionary Identification

보통 프로그램 소스코드에는 여러 라인으로 구성되어 있으며, 각 라인에 여러 토큰들로 구성이 된다. 다음과 같이 표현한다.

$$\mathrm{Pro}gram_{Buggy} = \begin{Bmatrix} (l_1,\ t_1),\ (l_1,\ t_2),\ \cdots, \\ (l_2,\ t_1),\ \cdots,\ (l_n,\ t_n) \end{Bmatrix}$$

$(l_1,\ t_1)$ 은 프로그램 소스코드의 첫 번째 라인에서의 첫번째 토큰을 의미한다.

만약 그림 1의 프로그램 버그 정정의 예제에서 프로그램 후보 패치를 적용하면 다음과 같이 표현이 될 수 있다.

$$\mathrm{Pro}gram_{Patch} = \begin{Bmatrix} (l_1,\ t_1),\ (l_1,\ t_2),\ \cdots, \\ (l_{11},\ t'_2),\ \cdots,\ (l_n,\ t_n) \end{Bmatrix}$$

$(l_{11},\ t'_2)$ 은 프로그램 소스코드의 라인 11에서 2번째 토큰을 사용자 토큰 목록을 이용하여 변형한다. 따라서 의심스러운 버기 라인을 이용하여 새로운 시퀀스 토큰을 생성한다.

4.2 SeqGAN 알고리즘 적용

SeqGAN 모델은 2가지 모델로 구성되어 있다. 첫 번째 모델은 생성 모델이다. 생성 모델은 RNN 모델에 프로그램 시퀀스를 입력 받고, Hidden Unit Activation 함수를 업

데이트 한다. 다음과 같이 표현한다.

$$HiddenUA_t = Funct(HiddenUA_{t-1}, x_t)$$

- Funct은 입력 프로그램 시퀀스 토큰을 재귀적으로 호출하여 Hidden States에 넣는다.
- t는 시간을 의미한다.

입력된 시퀀스에 대한 출력 output (HiddenUA*t*)은 다음과 같다.

$$output(HiddenUA_t) = softmaxFunct(bvector + Wvector\, hiddenUA_t)$$

- *b*는 신경망에서의 바이어스 벡터를 의미하며, *Wvector*는 가중치 벡터 매트릭스를 의미한다.

RNN에서 자주 발생하는 Vanishing Gradient Problem의 문제를 해결하기 위해 본 논문에서도 Long Short-Term Memory (LSTM)[13]을 사용한다. 식별 모델에서는 Convolutional Neural Network (CNN)[14]을 이용한다. 시퀀스의 입력을 다음과 같이 진행한다.

$$x\_input_1 \oplus x\_input_2 \oplus \ldots\ldots \oplus x\_input_i$$

- x_input_i은 토큰 임베딩을 의미하고, ⊕는 토큰 임베딩을 연결 시킨다.

그 다음, CNN에서 Filter Size를 여러 개로 서로 다른 특징을 추출하면서, Max Over Time Pooling[1]을 시킨다. 성능을 향상하기 위해, Highway network[1]를 적용하고, Fully-Connected Layer[1]와 Sigmoid Activation 함수[1]를 이용한다. 그리고 Cross Entropy[1]를 사용하여 최적화를 진행하며, 입력된 프로그램 시퀀스가 정확한지 아닌지 판단한다.

4.3 적합도 함수 계산

생성된 프로그램 패치 후보에 대한 적합성을 평가 하기 위해 본 논문에서는 적합도 함수[3]를 이용하여 계산한다. 사용된 수식은 다음과 같다.

$$FitnessFunct(Program_{Patch}) = \frac{|WhiTest_{Passed}| + |BlkTest_{Passed}|}{|WhiTest| + |BlkTest|}$$

- $Program_{Patch}$는 생성된 후보 패치를 의미한다.
- $WhiTest$는 데이터셋에서 주어진 화이트 박스 테스트 케이스 개수를 의미하고, $BlkTest$는 블랙 박스 테스트 케이스 개수를 의미한다.
- $WhiTest_{Passed}$는 통과한 화이트 박스 테스트 케이스의 개수이고, $BlkTest_{Passed}$는 통과한 블랙 박스 테스트 케이스의 개수이다.

따라서 테스트 케이스를 많이 통과할수록 적합도 함수의 값이 1.0에 근접한다. 모든 테스트 케이스가 통과하게 되면 1.0으로 계산된다.

5. 실 험

프로그램 버그 정정에서 사용된 데이터 셋은 다음 표 2와 같다. 오픈 소스 프로젝트 Defect4J[15]의 데이터 셋을 사용한다.

(표 2) 사용된 데이터 집합
(Table 2) Our Dataset

Project	Program	# of Bugs	# of Test Cases
Defect4J	JFreeChart	26	2,205
	Commons Math	106	3,602

추가적으로 본 논문에서는 사용된 Java 프로그램에 대한 정정된 소스코드와 관련 설명을 저장소[16]로부터 키워드 검색으로 추출한다. 사용한 데이터 셋과 추출한 데이터셋을 모두 이용하여 모델을 학습한다.

5.1 베이스라인

본 논문에서의 베이스라인은 다음과 같이 설정한다.

- kGenProg[3]: 유전 프로그래밍 기반으로 프로그램 버그 정정을 한다.
- SeqGAN[1]: 생성 모델과 식별 모델로 구성되어 입력 시퀀스에 대해 프로그램 버그 정정을 한다.
- jKali[17]: Return 구문을 추가하고, if 조건문을 변형하여 프로그램 버그 정정을 한다.
- Nopol[17]: if 조건문을 변형하여 프로그램 버그 정정을 한다.

베이스라인에서의 SeqGAN 모델은 과거 C언어 기반 구문오류 프로그램 정정 기법을 Java언어 기반 런타임 프로그램 정정으로 변환한 모델이다. 또한 해당 모델은 모든 프로그램 소스코드 기반으로 학습을 진행하는 특성을 갖고 있고, 본 논문에서는 공개된 유사 소스코드도 같이 활용하는 차이점을 지닌다.

5.2 실험 평가를 위한 환경 설정

공정한 비교를 위해 제안한 방법과 베이스라인에 대해 모델 학습과 프로그램 패치 생성에서 CPU Time을 최대 10시간으로 제한한다. 본 논문에서 사용한 CPU 규격은 Intel® Xeon® processor E5-1650 v3 3.50GHz 12Cores 이다.

5.3 연구 질문

연구질문 1: 제안한 프로그램 버그 정정 기법이 어느 정도의 정확성을 갖는가?

이 질문은 베이스라인과의 비교하기 전에 제안한 방법이 어느 정도의 버그를 정정하는지 살펴보는 질문이다.

연구질문 2: 제안한 프로그램 버그 정정 기법이 프로그램 버그 정정에 적용될 수 있는가?

이 질문은 베이스라인과의 비교를 통해 제안한 기법이 프로그램 버그 정정에 적용될 수 있는가에 대해 살펴보는 질문이다. 추가적으로 제안한 방법과 베이스라인과의 유의미한 차이를 비교하기 위해 통계 검증[18, 19]도 진행한다.

통계 검증을 진행하기 위해 본 논문에서 다음과 같이 귀무가설을 설정한다.

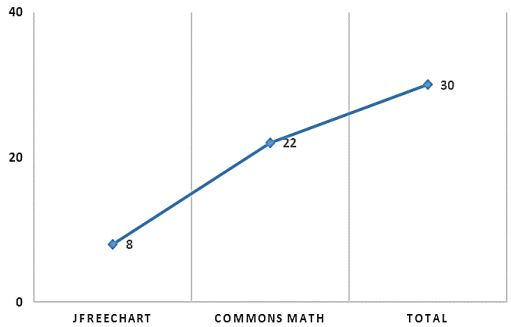
- H1-1_0: 제안한 방법과 kGenProg 방법간 차이가 없다.
- H1-2_0: 제안한 방법과 jKali 방법간 차이가 없다.
- H1-3_0: 제안한 방법과 Nopol 방법간 차이가 없다.
- H1-4_0: 제안한 방법과 SeqGAN 방법간 차이가 없다.

먼저, 제안한 모델과 베이스라인간 정규 분포[20]를 계산한다. 정규 분포가 0.05보다 크면 t검증[18]을 진행하고 0.05보다 작으면 Wilcox검증[19]을 진행한다. 각 검증의 결과 (p-value)가 0.05보다 크면 설정한 대립가설을 승인한다. 따라서 제안한 방법과 베이스라인간 유의미한 차이가 없음을 의미한다.

5.4 연구 결과

연구질문 1: 제안한 프로그램 버그 정정 기법이 어느 정도의 정확성을 갖는가?

제안한 알고리즘의 성능은 그림 4와 같다. X축은 Defect4J 오픈 소스 프로젝트에서의 JfreeChart와 Common Math 프로그램을 의미하고, Y축은 정정한 버그 개수를 의미한다.



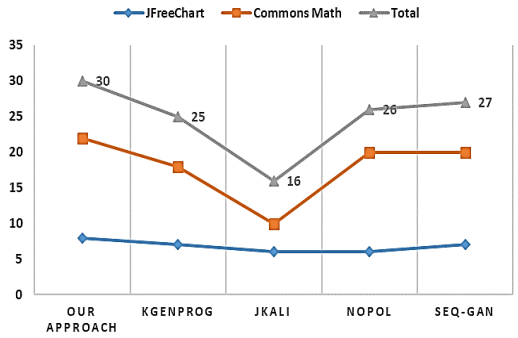
(그림 4) 제안한 모델의 성능

(Figure 4) A Performance of Our Model

연구 답변 #1: 총 132개의 버그에서 30개의 버그를 정정하였다.

연구질문 2: 제안한 프로그램 버그 정정 기법이 프로그램 버그 정정에 적용될 수 있는가?

제안한 모델과 베이스라인간 성능 비교는 그림 5와 같다. X축은 제안한 모델과 베이스라인을 의미하고, Y축은 정정한 버그 개수를 의미한다.



(그림 5) 베이스라인과 성능 비교

(Figure 5) A Comparison Result with baseline

추가적으로 통계 검증을 진행한다. 통계 검증 결과는 표 3와 같다. 모든 귀무가설에 대한 결과 (p-value)가 0.05 보다 작으므로 귀무가설을 기각하고, 이에 대한 대립가설을 승인한다. 예를들어 H1-1_0는 제안한 방법과 kGenProg 방법간 유의미한 차이가 없음을 의미하지만, 검증 결과의 값 (0.038)이 0.05보다 작으므로 귀무가설을 기각하고, 대립가설을 승인한다. 따라서 제안한 방법이 kGenProg 보다 프로그램 버그 정정에서 유의미한 차이를 보인다.

(표 3) 통계 검증 결과
(Table 3) A Result of Statistical Test

귀무가설	결과 (p-value)	대립가설
H1-10	0.038	승인: H1-1_a
H1-20	0.043	승인: H1-2_a
H1-30	0.035	승인: H1-3_a
H1-40	0.039	승인: H1-4_a

연구 답변 #2: 프로그램 버그 정정에서 베이스라인 보다 프로그램을 더 정정하였고, 통계 검증에서도 유의미한 차이를 보였다.

6. 토 의

6.1 실험 결과

본 논문에서는 프로그램 버그 정정에서 SeqGAN 알고리즘을 활용하여 프로그램 버그 정정을 진행했다. 사용된 오픈 소스 프로젝트에서의 132개의 버그 중 30개의 버그를 정정하였으며, 베이스라인 보다 더 버그를 정정하였다. 통계 검증에서도 베이스라인 보다 유의미한 차이가 있음을 보였다. 성능에 대해서는 전체 소스코드만을 이용하여 모델을 학습하는 베이스라인 SeqGAN 보다 공개된 유사 소스코드를 활용하는 것이 프로그램 버그 정정 성능을 향상하였다.

6.2 위협 요소

6.2.1 구조적 위협 요소

제안한 모델의 성능을 평가하기 위해 프로그램 버그 정정 과정에서 일반적으로 사용되는 적합도 함수를 이용하였다. 따라서 제안한 모델이 항상 좋은 결과를 보인다고 일반화 할 수는 없으며, 적합도 함수 뿐만 아니라 다양한 평가 척도를 이용하여 모델을 검증할 예정이다.

6.2.2 내부 위협 요소

프로그램 후보 패치의 적합성을 평가하기 위해 사용된 적합도 함수 기본 개념은 주어진 테스트 케이스를 모두 통과하면 적합하다고 본다. 하지만 모든 테스트 케이스를 통과하였다고 프로그램 버그가 정확하게 정정되었다고 일반화 할 수는 없다.

6.2.3 외부 위협 요소

프로그램 버그 정정 모델을 학습하기 위해 본 논문에서는 오픈 소스 프로젝트를 사용하였다. 비즈니스 프로젝트 적용 가능 여부는 추가 연구가 필요하며, 향후 다양한 오픈 소스 프로젝트를 사용하여 모델을 검증할 예정이다.

7. 결 론

본 논문에서는 SeqGAN 알고리즘을 소프트웨어 버그 정정에 적용하여 프로그램 버그 정정 기법을 제안했다. 자세히는 소프트웨어 저장소로부터 소스코드를 추출하고, 키워드 검색으로 관련 프로그램 소스코드와 설명을 추출하였다. 추출한 소스코드에 대해서 프로그램 변환 과정을 진행한다. SeqGAN 알고리즘을 이용하여 프로그램 소스코드에 대해 모델을 학습한다. 의심스러운 버기 라인을 학습 모델에 입력을 하여 새로운 프로그램 소스코드 시퀀스를 생성한다. 생성된 후보 프로그램의 적합성을 평가하기 위해 주어진 테스트 케이스를 모두 통과하는 후보 패치에 대해 적절하게 패치 되었다고 본다. 따라서 적절한 후보 패치를 버기 프로그램에 적용하여 프로그램 버그 정정을 진행한다. 제안한 방법의 효율성을 평가하기 위해 유사 연구와 비교를 하였으며, 제안한 방법이 더 효율적임을 보였다. 추가적으로 통계 검증을 통하여 제안한 모델과 베이스라인간의 유의미한 차이를 보였다. 향후 다양한 딥러닝 알고리즘을 소프트웨어 버그 정정에 적용할 예정이며, 프로그램 정정 모델을 더욱 구체화 할 예정이다.

참고문헌(References)

[1] G. Yang, K. Min, B. Lee, "Applying Deep Learning Algorithm to Automatic Bug Localization and Repair", In Proc. of ACM Symposium on Applied Computing,

pp. 1634-1641, 2020.
https://doi.org/10.1145/3341105.3374005

[2] R. Gupta, S. Pal, A. Kanade, and S. Shevade, "Deepfix: Fixing Common C Language Errors by Deep Learning", In Proc. of Thirty-First AAAI Conference on Artificial Intelligence, pp. 1345-1351, 2017.
https://arxiv.org/abs/1510.02927

[3] Y. Higo, S. Matsumoto, R. Arima, A. Tanikado, K. Naitou, J. Matsumoto, Y. Tomida, and S. Kusumoto, "kGenProg: A High-Performance, High-Extensibility and High-Portability APR System," In Proc. of Asia-Pacific Software Engineering Conference, pp. 697-698, 2018.
https://doi.org/10.1109/APSEC.2018.00094

[4] R. Messina, and J. Louradour, "Segmentation-free Handwritten Chinese Text Recognition with LSTM-RNN", In Proc. of 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 171-175, 2015.
https://doi.org/10.1109/ICDAR.2015.7333746

[5] I. Sutskever, O. Vinyals, and Q.V. Le, "Sequence to Sequence Learning with Neural Networks", In Proc. of Advances in Neural Information Processing Systems, pp. 3104-3112, 2014.
https://arxiv.org/abs/1409.3215

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A.C. Courville, and Y. Bengio, "Generative Adversarial Nets", In NIPS, 2014.
https://papers.nips.cc/paper/5423-generative-adversarial-nets

[7] V. Debroy, and W.E. Wong, "Using Mutation to Automatically Suggest Fixes for Faulty Programs", In Proc. of Third International Conference on Software Testing, Verification and Validation, pp. 65-74, 2010.
https://doi.org/10.1109/ICST.2010.66

[8] D. Kim, J. Nam, J. Song, and S. Kim, "Automatic Patch Generation Learned from Human-written Patches", In Proc. of the 2013 International Conference on Software Engineering, pp. 802-811, 2013.
https://doi.org/10.1109/ICSE.2013.6606626

[9] F. DeMarco, J. Xuan, D. Le Berre, and M. Monperrus, "Automatic Repair of Buggy IF Conditions and Missing Preconditions with SMT", In Proc. of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis, pp. 30-39, 2014.
https://doi.org/10.1145/2593735.2593740

[10] F. Long, and M. Rinard, "Automatic Patch Generation by Learning Correct Code", In ACM SIGPLAN Notices, vol. 51, no. 1, pp. 298-312, 2016.
https://doi.org/10.1145/2837614.2837617

[11] Geunseok Yang, Cheolhun Lee, Hyunho Choi and Byungjeong Lee, "A Novel Technique for Automatic Bug Repair by using Seq-GAN Algorithm", In Proc. of the 22nd Korea Conference on Software Engineering (KCSE 2020), pp. 43-46, 2020.
http://sigsoft.or.kr/KCSE2020/

[12] Geunseok Yang, Hyunho Choi, Cheolhun Lee and Byungjeong Lee, "Utilizing Seq-GAN Algorithm with Feature Extraction from Source Code and Bug Report for Automatic Bug Repair", In Proc. of Korea Computer Congress (KCC 2020), 2020.
http://www.kiise.or.kr/conference/main/index.do?CC=kcc&CS=2020

[13] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", In Neural Computation, 9(8), pp. 1735-1780, 1997.
https://doi.org/10.1162/neco.1997.9.8.1735

[14] Y. Kim, "Convolutional Neural Networks for Sentence Classification", In arXiv preprint arXiv: 1408.5882, 2014.
https://arxiv.org/abs/1408.5882

[15] M. Martinez and M. Monperrus, "Astor: A Program Repair Library for JAVA", In Proc. of International Symposium on Software Testing and Analysis, pp. 441 – 444, 2016.
https://doi.org/10.1145/2931037.2948705

[16] Github, https://github.com/

[17] M. Martinez, T. Durieux, R. Sommerard, J. Xuan, and M. Monperrus, "Automatic repair of real bugs in java: A large-scale experiment on the defects4j dataset", In Empirical Software Engineering, 22(4), 1936-1964, 2017.
https://dx.doi.org/10.1007/s10664-016-9470-4

[18] The T-Test, Research Methods Knowledge Base, http://www.socialresearchmethods.net/kb/stat_t.php.

[19] F. Wilcoxon, "Individual Comparisons by Ranking Methods" In Biometrics Bulletin, Vol. 1, No. 6, pp. 80-83, 1945.

https://dx.doi.org/10.2307/3001968
[20] Shapiro–Wilk test, WIKIPEDIA,
http://en.wikipedia.org/wiki/Shapiro-Wilk_test.

◐ 저 자 소 개 ◑

양 근 석(Geunseok Yang)
2013년 한국기술교육대학교 컴퓨터공학부(공학사)
2015년 서울시립대학교 일반대학원 컴퓨터과학과(공학석사)
2020년 서울시립대학교 일반대학원 컴퓨터과학과(공학박사)
2020년 9월～현재 서울시립대학교 컴퓨터과학부 연구교수
관심분야 : 지능형 소프트웨어공학, AI인공지능
E-mail : ypats87@uos.ac.kr

이 병 정(Byungjeong Lee)
1990년 서울대학교 계산통계학과
1998년 서울대학교 대학원 전산과학과
2002년 서울대학교 대학원 전기컴퓨터공학과
2002년 3월～현재 서울시립대학교 컴퓨터과학부 교수
관심분야 : 소프트웨어 진화, 소프트웨어 공학
E-mail : bjlee@uos.ac.kr