유사 패치 기반 자동 프로그램 수정 기법

장세창[O1], 최준혁[1], 김성빈[1], 김진대[2], 남재창[1]
[1]한동대학교 전산전자공학부
[2]서울과학기술대학교 컴퓨터공학과
{newwin0198,21900764,22100113}@handong.ac.kr, jindae.kim@seoultech.ac.kr, jcnam@handong.edu

SPI: Similar Patch Identifier for Automated Program Repair

Sechang Jang[O1], Junhyeok Choi[1], Seongbin Kim[1], Jindae Kim[2], Jaechang Nam[1]
[1] School of Computer Science and Electrical Engineering, Handong Global University
[2]Department of Computer Science and Engineering, Seoul National University of Science and Technology
{newwin0198,21900764,22100113}@handong.ac.kr, jindae.kim@seoultech.ac.kr, jcnam@handong.edu

요 약

자동 프로그램 수정(Automated Program Repair, APR) 기술의 주요 관심사는 탐색 공간의 크기 문제이다. 본 연구에서는 버그를 생성한 수정(Bug Introducing Change)의 유사성을 활용하여 탐색 공간을 줄이고, 적절한 수정 연산자를 제안하는 새로운 접근 방식인 Similar Patch Identifier(SPI)를 제안한다. 이 접근법을 평가하기 위해, 기존의 문맥 기반 APR 도구인 ConFix와 자바 결함 벤치 마크인 Defects4J를 활용했다. 실험을 통해 SPI가 탐색 공간을 각 결함에 적합한 10개의 버그 수정 커밋 후보로 줄였음에도, 기존 APR 도구인 ConFix가 고치지 못했던 4개의 버그에 대한 수정을 만들어내는 유의미한 결과를 확인 할 수 있었다.

1. 서 론[1]

자동 프로그램 수정(Automated Program Repair, APR) 기술은 발견된 소프트웨어 버그를 자동으로 수정하는 기술로써 소프트웨어의 발전으로 인한 소프트웨어 유지, 보수 비용 증가로 인해 그 필요성이 증가하고 있다[1]. APR 기술은 접근법에 따라 휴리스틱 기반[2,3], 패턴 기반[4], 학습 기반[5,6] 기술로 분류되며, 대부분의 APR 도구들이 많은 후보 패치를 생성하고 테스트 케이스를 이용하여 정답 패치를 찾는다. 이때 후보 패치 생성을 위해 고려해되는 모든 부분의 범위를 탐색 공간[7]이라 한다. APR 도구의 탐색 공간은 버그 위치 후보의 수, 코드의 길이, 코드의 복잡성에 따라 증가한다.

하지만, 기존 APR 기술들의 패치 탐색 공간은 여전히 방대하여, 효율적인 정답 패치 생성에 한계가 있다. 탐색 공간은 APR 도구의 성능과 직접적인 연관이 있다. 탐색 공간의 확대는 자원의 낭비를 야기한다. 탐색 공간 확대 시 더 많은 패치 후보가 생성되지만 패치 후보의 증가는 패치 평가 시간을 늘리며 오히려 올바른 패치의 밀도를 감소시킨다[7].

본 연구에서는 APR 기술의 탐색 공간 문제를 버그 생성 커밋의 수정 사항을 이용하여 해결하고자 한다. 버그를 생성한 수정 (Bug Introducing Change)을 유사하게 가지는 두 버그는 수정 또한 유사하다는 가설[8]을 기반으로 진행되었다.

본 연구의 접근법을 검증하기 위해 문맥 기반의 APR 도구인 ConFix[9]와 자바 결함 벤치마크인 Defects4J[10]를 사용하여 평가를 진행했다. 실험 과정에서 본 논문이 제안하는 기법인 SPI(Similar Patch Identifier)를 통해 26,660개의 파일 수정 데이터를 활용하여, 특정 버그별로 최적의 버그 수정 커밋 후보를 추천했고, ConFix의 6,485개의 버그 수정 커밋 후보에서 10개의 커밋 후보로 탐색 공간을 줄였다. 실험을 통해 SPI가 실제 버그 수정 상황에서 유효하게 작동함을 확인했고 본 연구의 접근이 APR 도구의 효율성과 정확성을 증가시키는 것에 기여할 수 있음을 확인했다.

[1]본 연구는 2023년 과학기술정보통신부 및 정보통신기획 평가원 의 SW중심대학사업 지원(2023-0-00055)과 한국연구재단의 지원 (No.2021R1F1A1063049)을 받아 수행되었음.

2. 접 근

SPI의 핵심 접근은 오픈 소스 프로젝트들의 과거 버그 기록에서 수정 대상 버그과 유사한 버그 사례를 찾고, 실제 해당 버그의 패치에서 사용한 수정 연산자를 찾는 과정에 있다. 찾은 수정 연산을 바탕으로 기존 APR 도구의 패치 생성 단계에 활용한다. 본 연구에서는 ConFix를 활용해 핵심 접근을 평가한다. 그림 1은 유사한 버그를 찾는 과정, 과거 버그 수정 커밋 후보에서 수정 연산자를 찾는 과정, 그리고 APR 도구의 패치 생성 단계, 총 3 단계를 보여준다.

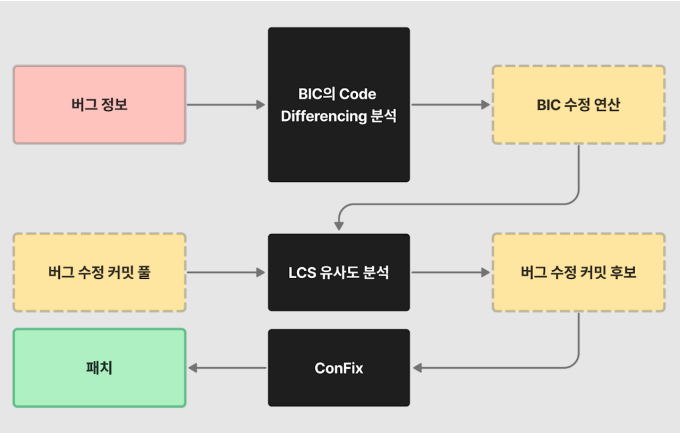

그림 1. SPI의 흐름

유사한 문맥을 가지는 버그는 유사한 수정 연산으로 이루어진 수정이 존재한다는 가정[8]을 따라, 먼저 주어진 버그를 수정하는 연산을 찾기 위해 패치간의 문맥을 파악한다. 본 연구에서는 패치간의 문맥을 BIC의

수정 연산으로 정의하였다. 수정 연산은 주어진 코드 파편에 돌연변이 연산자를 적용한, Insert, Delete, Update, Move 등의 수정 사항으로 표현된 연산의 조합이다[21]. 수정 연산을 활용할 경우, 기존의 수정된 코드 전체를 나타낼 때보다 간결하게 표현된다는 장점이 있다. 수정 연산은 추상 구문 트리기반 코드 Differencing 도구인 GumTree[11]를 사용하여 추출하였다.

그 다음 단계로 유사한 문맥을 가진 과거 버그 수정 커밋을 선별하여 기존 APR 도구의 패치 생성 단계에서의 탐색 공간을 줄인다. 유사도를 판별하기 위해 최장 공통 부분 수열 알고리즘(Longest Common Subsequence, LCS)을 이용하였다. LCS 유사도 분석 단계에서는 패치들 사이 여러 수정 연산들의 LCS를 구하고, 그 길이를 이용해 유사도를 판단한다. 이 때 두 수정 연산이 동일한 지 판단하기 위해 수정 연산과 관계된 추상 구문 트리를 수치형 벡터로 변환하고, 벡터가 동일한 지를 비교하였다. 이후 유사도가 높은 상위 10개의 후보를 채택해 SPI에서 사용하는 APR 도구에게 해당 버그 수정 정보를 제공하여 패치를 생성한다.

오픈 소스 프로젝트들의 실제 버그 수정 커밋 정보를 가져오기 위해 버그가 수정된 시점의 커밋(Bug Fixing Commit, BFC)을 수집하는 DPMiner[12]를 사용하였다. DPMiner는 결함 예측에 필요한 다양한 형태의 데이터를 하나의 프로그램을 통해 수집하는 통합 프레임워크이다. Jira, Github Issue, Github Commit Message 정보를 통해 BFC를 수집한다. 주어진 BFC 정보를 통해 BIC를 수집하기 위해 SZZ[13] 알고리즘에 따라 주어진 BFC에 따른 BIC를 추출했다.

3. 실험 설계

본 연구에서는 제안한 기법을 평가하기 위해 기존의 문맥 기반의 APR 도구인 ConFix[9]를 사용했다. 실험 대상으로는 Defects4J[10] v.2.0.1을 사용하였다. SPI의 성능을 확인하기 위해 ConFix가 버그 수정을 시도했던 Chart, Closure, Lang, Math, Time, Mockito 프로젝트들에 대해 실험을 진행하였다.

우선적으로 SPI의 잠재력을 평가하기 위해 싱글 라인 수정 패치 생성에 집중했다. 실험에 사용된 ConFix가 단일 위치에서의 코드 수정만을 제안하기 때문이다. 또한 현재 SPI는 단일 소스 코드 파일의 변경 수준에서 버그 수정 커밋 후보를 생성한다.

따라서 SPI의 평가는 다음 유형의 버그들을 제외하고 수행됐다: 여러 파일에 걸친 패치가 필요한 버그, 단일 파일에서 여러 위치에 있는 버그, 대규모의 버그, 새로운 논리가 추가된 버그. 실험은 총 136개의 버그로 진행되었다.

적합한 수정 연산을 가진 유사 패치를 찾기 위해서는 정교한 풀(pool)의 생성이 필수적이다. 현재까지 널리 쓰이고 활발히 커밋이 이루어지고 있는 프로젝트들을 선택하기 위해 Apache 프로젝트들 내에서 풀을 선택했다. 버그 추적 소프트웨어인 Jira를 통해 Apache 프로젝트들 중 버그 수정이 활발히 발생한 프로젝트들 5개를 선택했다. 이러한 기준에 따라 Beam[14], Cassandra[15], Hadoop[16], jUDDI[17], Kafka[18], Spark[19]가 선택되었다.

SPI가 APR 도구의 성능에 미치는 영향을 파악하기 위해 ConFix에게 버그의 실제 위치 정보를 제공하였다. ConFix는 4가지 패치 전략과 4가지 구체화 전략을 적용하여 하나의 패치라도 발견되면 해당 패치를 발견한 것으로 간주한다. 또한, 패치 생성이 2시간 이내에 완료되지 않으면, 인간 패치보다 효율성이 떨어지는 것으로 간주해 패치 생성 실패로 판단한다[9].

4. 실험 결과

표 1은 ConFix[9], 정답 패치를 후보로 제공한 ConFix[20], 그리고 SPI의 결과를 비교한다. 표 1의 항목들은 Defects4j[10] 프로젝트들에 대해 각각의 기법이 생성한 정확한 패치와 Plausible Patch의 개수이다. 정확한 패치는 개발자가 생성한 패치와 의미적으로 동일한 Patch를 말하며, Plausible Patch는 해당 버그 수정에 대한 모든 testcase를 통과한 패치를 말한다[1].

SPI는 Defects4J[10] 프로젝트 136개 중 32개의 패치를 생성했으며 그 중 12개가 정확했다. 괄호 내에 있는 숫자는 SPI가 생성한 Plausible Patch 개수를 말한다. Original ConFix는 Mockito를 대상으로 실험을 진행하지 않았고, ConFix with Answers[20]는 Chart를 대상으로 실험을 진행하지 않아 N/A로 표시하였다.

ConFix가 357개 버그에서 생성한 22개의 정확한 패치와 비교하여, SPI의 접근 방식은 더 적은 수의 정확한 패치를 생성하였다. 그러나 SPI는 ConFix가 수정하지 못했던 Closure 10과 86, Lang 59, Math 59 버그에 대해 정확한 패치를 생성했다. 또한 SPI는 탐색 공간 축소를 위해 10개의 과거 커밋 후보만을 수정 연산을 추출하기 위해 사용했다. 이는 총 6,485개의 버그 수정 커밋 후보를 사용한 ConFix의 0.15%의 비율이다.

표 1. 성공적으로 생성한 패치 결과

ConFix With Answer는 ConFix에 정답 수정 커밋이 제공되었을 때를 의미.

Project	SPI	Original ConFix	ConFix with Answers
Chart	**3(4)**	4	N/A
Closure	**3(12)**	6	20
Lang	**2(4)**	5	5
Math	**4(11)**	6	20
Time	**0(1)**	1	1
Mockito	**0(0)**	N/A	4
Sum	**12(32)**	**22**	**50**

표 2는 SPI가 고려하는 파일 수정의 개수가 성능에 미치는 영향을 나타낸다. 표2의 결과는 표1과 마찬가지로 SPI가 생성한 정확한 패치와 Plausible 패치의 개수를 나타낸다. 표 내의 File Change는 SPI가 후보 채택시 고려하는 파일 수정의 개수이다. SPI가 고려하는 파일 수정의 개수가 증가하면 SPI는 더 많은 버그를 수정한다. 이는 SPI가 고려하는 패치가 많아진다면, SPI의 성능을 현재의 실험보다 향상시킬 수 있다는 잠재력을 보인다.

표 2. 파일 수정 개수가 커질때 SPI가 생성하는 패치의 개수

Project	File Change: 3,262	File Change: 26,660
Chart	3(4)	**3(4)**
Closure	3(8)	**3(12)**
Lang	2(3)	**2(4)**
Math	3(7)	**4(11)**
Time	0(1)	**0(1)**
Mockito	0(0)	**0(0)**
Sum	**11(23)**	**12(32)**

또한 LCS 유사도를 이용한 패치 추천이 효과적으로 필요한 수정 연산을 제공했다는 것을 표 3의 결과를 통해 확인했다. 추천된 패치들의 평균적인 LCS 유사도가 패치를 생성했을 경우 더 높음을 볼 수 있다. 예를 들어, Closure 프로젝트에서 패치를 생성했을 경우, 패치를 생성하지 못했을 경우보다 높은 점수인 0.83을 보여준다.

LCS 알고리즘을 이용한 유사도 측정이 패치 생성의 효율성을 평가하는 데 유효한 도구로 활용됨을 통계적으로 확인하기 위해 Mann-Whitney U 검정을 실시하였다. 결과적으로, 패치를 성공적으로 생성했을 때와 생성하지 못했을 때의 LCS 유사도 점수는 통계적으로 유의미한 차이를 보였다. p-값은 0.00018로 계산되었으며, 유의수준 5%에서 매우 유의미하다는 것을 의미한다. 이는 패치가 성공적으로 생성된 경우, 그렇지 못한 경우에 비해 상대적으로 높은 LCS 유사도를 나타낸다는 것이 검증되었음을 나타낸다.

이는 문맥을 기반으로 한 버그 수정 커밋 추천이 실제 코드 수정에서

좋은 성공률을 보임을 시사한다. ConFix에 비해 적은 양의 패치를 생성하지만, 추가적인 패치를 생성하며, 작은 개수의 버그 수정 커밋을 이용해 효과적으로 버그를 수정함을 보여주었다. SPI는 효과적으로 탐색 공간의 크기를 줄이고, 효율적으로 패치를 생성하는 것을 도울 수 있다.

표 3. LCS 평균 점수의 차이
*: 통계적으로 유의미한 차이를 보임

Project	Patch-Generated LCS Score	No Patch-Generated LCS Score	Overall LCS Score
Chart	0.88	0.75	0.80
Closure	0.83	0.63	0.68
Lang	0.83	0.51	0.60
Math	0.81	0.70	0.73
Time	0.96	0.50	0.75
Mockito	N/A	0.75	0.65
All	0.82*	0.65*	0.70

5. 타당성 평가

ConFix[9]의 성능은 추상 구문 트리 변경 사항 수집에 영향을 받는다. ConFix는 다수의 개발자가 작성한 버그 수정 커밋으로부터 풀을 구축한다. 이후, 버그 수정으로부터 추상 구문 트리(AST) 변경사항과 해당 AST 문맥을 수집한다. 이렇게 수집된 변경사항들은 패치 생성을 위해 동일한 문맥을 가진 가능한 수정 위치에만 적용된다. ConFix는 결함을 수정하는 수정 정보가 제공되면 사람이 작성한 패치와 동일한 수준의 패치 생성의 가능성을 보였고[20], SPI의 문맥을 고려하는 접근과 통합했을 때 긍정적인 효과를 보였다.

Defects4J[10]는 APR 도구의 성능을 평가하는 벤치마크이다. Defects4J는 각각의 버그를 재연하기 위해 버그가 이미 수정된 프로젝트 내에 해당 버그를 주입하는 방식으로 버그를 재현한다. SPI의 접근에 따르면 BIC는 버그가 이미 수정된 코드에서 버그가 생성된 코드의 수정이 된다. SPI의 성능을 평가하기 위해서 ConFix에서 사용된 Defects4J가 벤치마크로 사용되어야 했고, 실험을 통해 특수한 상황 속에서도 BIC를 이용하는 접근은 패치 생성의 효율성에 영향을 미침을 확인했다.

6. 결론 및 향후 계획

본 연구에서는 BIC의 유사성을 이용하는 접근을 제시하였다. 버그 수정 연산자간의 유사성이 존재한다는 가정하에 시작되어, 오픈 소스 프로젝트들과 비교하여 BIC 간의 문맥이 유사한 경우 해당 수정 연산자를 패치 생성에 사용하였다. 이를 통해 APR 기술의 근본 문제점인 탐색 공간 문제를 해결하고자 했고, 적절한 재료와 돌연변이 연산자를 포함하는 패치 후보를 추출해 탐색 공간을 줄였다. 기존 APR 도구, ConFix를 활용해 Defects4J 내에서 성능 평가를 시행했고, 탐색 공간 축소 및 성능의 향상을 확인했다.

실험에서 SPI의 잠재력을 발견했고, 성능의 향상을 위해 활발한 커밋 활동 뿐만 아니라 더 추가된 기준에 따라 더욱 정교화된 GitHub 오픈 소스 프로젝트들에서 버그 수정 사항을 포함한 SPI의 수정 연산 벡터 풀을 구축할 계획이다. 추가로, BIC의 수정 연산 벡터간 유사성을 판단할때 사용한 LCS 알고리즘의 단순성과 사용한 APR 도구 자체의 한계를 극복하기 위해 더 적합한 알고리즘과 APR 도구를 사용해 성능의 향상을 기대한다.

참조 문헌

[1] Goues, Claire Le, Michael Pradel, and Abhik Roychoudhury. “Automated program repair.” Communications of the ACM 62.12 (2019), 56-65.
[2] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. “GenProg: A Generic Method for Automatic Software Repair.” IEEE Transactions On Software Engineering 38, 1 (2012), 54–72.
[3] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. “Automatic patch generation learned from human-written patches.” 35th International Conference on Software Engineering, (2013), 802–811.
[4] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F. Bissyandé. “TBar: Revisiting Template-Based Automated Program Repair.” In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis. (2019), 31–42.
[5] Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. “Coconut: combining context-aware neural translation models using ensemble for program repair.” Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis. (2020), S101-114.
[6] Chen, Z., Kommrusch, S., Tufano, M., Pouchet, L. N., Poshyvanyk, D., & Monperrus, M. Sequencer: “Sequence-to-sequence learning for end-to-end program repair.” IEEE Transactions on Software Engineering, 47, 9, (2019), 1943-1959.
[7] Fan Long and Martin Rinard. 2016. “An Analysis of the Search Spaces for Generate and Validate Patch Generation Systems.” In IEEE/ACM 38th International Conference on Software Engineering (2016). 702–713.
[8] Qi Xin, and Steven P. Reiss. “Leveraging syntax-related code for automated program repair.” 32nd IEEE/ACM International Conference on Automated Software Engineering. (2017), 660-670.
[9] Kim, Jindae, and Sunghun Kim. “Automatic patch generation with context-based change application.” Empirical Software Engineering 24.6 (2019): 4071-4106.
[10] Just, R, Jalali, D., Ernst, M.D. “Defects4J: A database of existing faults to enable controlled testing studies for Java programs.” Proceedings of the 2014 International Symposium on Software Testing and Analysis. 2014.
[11] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. “Fine-Grained and Accurate Source Code Differencing”. 29th ACM/IEEE International Conference on Automated Software Engineering (2014), 313–324.
[12] https://github.com/ISEL-HGU/DPMiner
[13] Jacek Śliwersk, Thomas Zimmermann, and Andreas Zeller. 2005. “When Do Changes Induce Fixes?” SIGSOFT Softw. Eng. Notes 30, 4 (May 2005), 1–5.
[14] https://github.com/apache/beam
[15] https://github.com/apache/cassandra
[16] https://github.com/apache/hadoop
[17] https://github.com/apache/juddi
[18] https://github.com/apache/kafka
[19] https://github.com/apache/spark
[20] Sunghyun Choi, Junghyun Heo, Chaewoo Yu, and Jaechang Nam. “Analysis on Results of ConFix Execution Through Correct Patch Change Information” KCC 2022.
[21] Connor, Aidan, Aaron Harris, Nathan Cooper, and Denys Poshyvanyk. “Can we automatically fix bugs by learning edit operations?.” IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), (2012) 782-792