

On the Efficiency of Test Suite based Program Repair

A Systematic Assessment of 16 Automated Repair Systems for Java Programs

Kui Liu, Shangwen Wang, Anil Koyuncu, Tegawendé F. Bissyandé, Dongsun Kim, Peng Wu, Jacques Klein, Xiaoguang Mao, Yves Le Traon

テストスイートベースのプログラム修正について

Javaプログラムの16の自動修正システムの体系的な評価

背景

- ・ テストベースの自動プログラム修正は過去10年間で多くの研究が行われてきた.
- ・ 生成されたパッチの正確性は多くの研究で疑問視されてきたが, パッチ生成の効率性に対処する研究はほとんどなかった.
- ・ この論文ではテストスイートベースのプログラム修正の効率に関する大規模な実証研究の結果について報告する.

自動プログラム修正

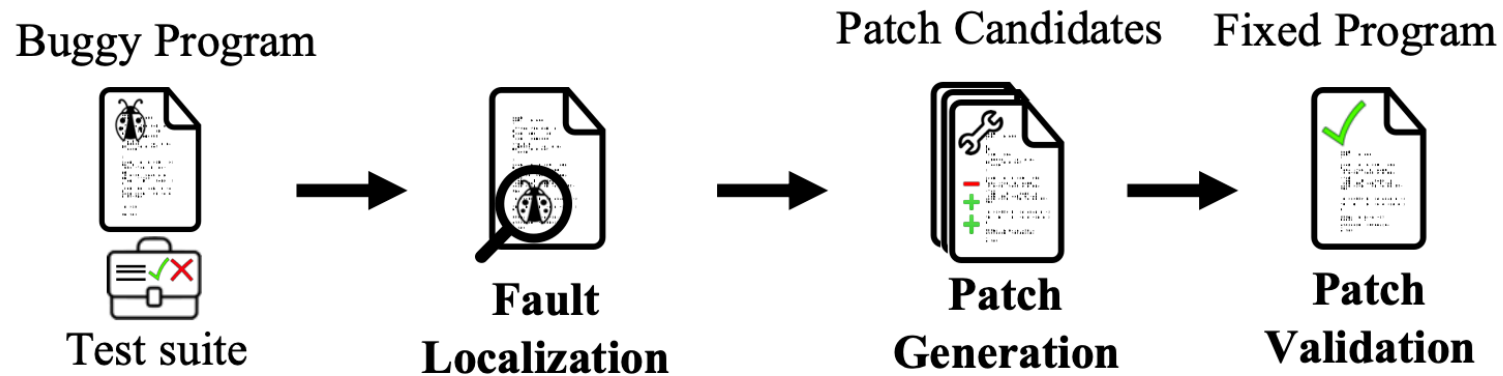


Figure 2: Standard steps in a pipeline of Automated Program Repair.

Research Questions

RQ1：テストスイートベースの修正アプローチで修正できるバグの数はどのくらいか.

RQ2：バグが修正されるまでにどのくらいの数のパッチ候補が生成, チェックされるか.

RQ3：障害位置特定ノイズの影響はどのくらいか.

対象ツール

- Javaプログラムを対象としたAPR(Automated Program Repair)システムに焦点を当てる.
- Webサイトのprogram-repair.orgに記載されているJava APRツールと他の論文でレビューされているJava APRツールの31のAPRツールを検討する.
- オープンソースであり, 利用する条件などが合う16のAPRツールを研究の対象とする.

Table 2: Included and excluded APR tools for our study.

Selected	Reason	APR Tools for Java Programs
No	Not public	PAR [16], xPAR [22], JFix/S3 [21], ELIXIR [50], Hercules [51], SOFix [33], CapGen [57], PraPR ^{\$} [11].
No	Faulty method required	HDRepair [22], JAID [3], SketchFix [13].
No	Other	LSRepair [*] [32], ssFix [★] [61], DeepRepair [†] [59], NPEFix [‡] [5].
Yes	Open-source & working	jGenProg [38], jKali [38], jMutRepair [38], Cardumen [39], DynaMoth [7], Nopol [64], ACS [63], SimFix [14], kPAR [29], FixMiner [18], AVATAR [30], TBar [31], ARJA [67], GenProg-A [67], Kali-A [67], RSRepair-A [67].

実験設定

- ・ バグのベンチマークには研究で一般的に使用されるDefects4Jを使用する. Defects4Jには6つのオープンソースプロジェクトにわたる395のバグが含まれている.
- ・ 障害位置の特定がAPRツールの修正パフォーマンスに影響を与えるので, 全てのツールで各バグに対して同じ障害位置情報を使用するように設定した.
- ・ 修正試行のしきい値を10,000に設定する.(もっとも試行回数が多かったPraPRの平均が9,696.5であったため)

実験結果

RQ1：テストスイートベースの修正アプローチで修正できるバグの数はいくら。

Table 4: Numbers of Defects4J bugs that are correctly (plausibly) fixed by the different APR tools.

APR Tool	C	CI	L	M	Mc	T	Total	CR(%)
jGenProg	0 (5)	2 (2)	0 (2)	3 (11)	0 (0)	0 (0)	5 (20)	25
GenProg-A	0 (5)	2 (15)	0 (1)	0 (9)	0 (0)	0 (0)	2 (30)	6.7
jMutRepair	1 (4)	2 (5)	0 (2)	2 (11)	0 (0)	0 (0)	5 (22)	22.7
kPAR	3 (13)	2 (10)	1 (18)	4 (22)	0 (0)	0 (1)	10 (63)	15.9
RSRepair-A	0 (4)	4 (22)	0 (3)	0 (12)	0 (0)	0 (0)	4 (41)	10
jKali	0 (4)	3 (8)	2 (4)	1 (9)	0 (0)	0 (0)	6 (25)	24
Kali-A	0 (6)	2 (48)	0 (0)	1 (10)	0 (1)	0 (0)	3 (65)	4.6
DynaMoth	0 (6)	N/A	0 (2)	1 (13)	0 (0)	0 (1)	1 (22)	4.5
Nopol	0 (6)	N/A	1 (6)	0 (18)	0 (0)	0 (1)	1 (31)	3.2
ACS	2 (2)	0 (0)	3 (3)	11 (16)	0 (0)	1 (1)	17 (22)	77.3
Cardumen	2 (4)	0 (2)	0 (0)	1 (6)	0 (0)	0 (0)	3 (12)	25
ARJA	1 (10)	2 (29)	0 (3)	3 (15)	0 (1)	0 (0)	6 (58)	10.3
SimFix	3 (8)	7 (19)	5 (16)	10 (25)	0 (0)	0 (0)	25 (68)	36.8
FixMiner	5 (14)	0 (2)	0 (2)	7 (15)	0 (0)	0 (0)	12 (33)	36.4
AVATAR	5 (12)	7 (15)	4 (13)	3 (17)	0 (0)	0 (0)	19 (57)	33.3
TBar	7 (16)	3 (12)	6 (21)	8 (23)	0 (0)	0 (0)	24 (72)	30.8

括弧外はDefects4Jで提供されているパッチと全く同じもので、括弧内のものは意味的に同じパッチである。

- 全体として最新のAPRツールは以前のものよりも多くのバグを正しく修正する傾向がある。(SimFix, ACS, AVATAR, TBar)

実験結果

RQ2：バグが修正されるまでにどのくらいの数のパッチ候補が生成，チェックされるか。

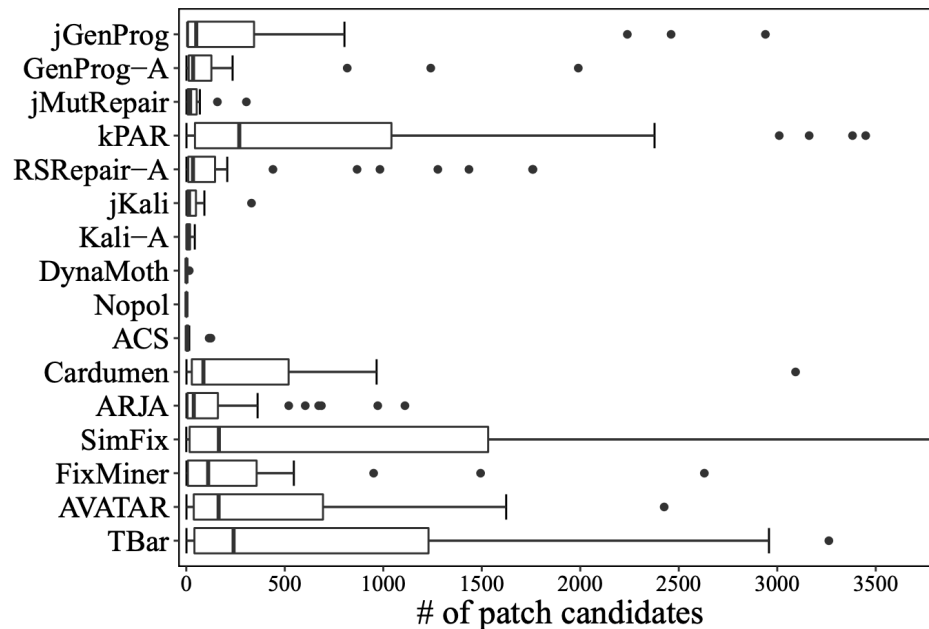


Figure 6: The distribution of NPC scores for 16 APR tools.

- 全てのツールで NPC(number of patch candidate)スコアの中央値は250未満であった。
- SimFix, TBar, kPARは NPCスコアがかなり高いが、RQ1の結果を見ると修正できるバグの数が多いツールほどNPCスコアも高い、つまり効率が悪かった。

実験結果

RQ3：障害位置特定ノイズの影響はどのくらいか.

Table 8: Impact[†] on repairability* when ground-truth fix locations are directly given to the APR system.

APR Tool	C	CI	L	M	Mc	T	Total	CR (%)
jGenProg	+1 (-3)	-1 (0)	0 (-2)	+1 (+1)	0 (0)	0 (0)	+1 (-4)	+12.5
GenProg-A	0 (-2)	+3 (+1)	+1 (+2)	+2 (-2)	0 (0)	0 (0)	+6 (-1)	+20.9
jMutRepair	0 (-3)	0 (-1)	0 (-2)	0 (-5)	0 (0)	0 (0)	0 (-11)	+22.8
kPAR	+3 (-5)	+11 (+11)	+3 (-5)	+3 (-6)	0 (0)	+3 (+4)	+23 (0)	+36.5
RSRepair-A	0 (-2)	0 (-6)	+1 (+1)	+4 (0)	0 (0)	0 (0)	+5 (-7)	+16.5
jKali	0 (-3)	-1 (-6)	-2 (-4)	-1 (-4)	0 (0)	0 (0)	-4 (-17)	+1
Kali-A	0 (-5)	+2 (-18)	0 (+3)	0 (-2)	0 (-1)	0 (0)	+2 (-23)	+7.3
DynaMoth	0 (-5)	N/A	+2 (+2)	0 (-5)	0 (0)	0 (-1)	+2 (-9)	+18.6
Nopol	0 (-5)	N/A	0 (-3)	+1 (-13)	0 (0)	0 (-1)	+1 (-22)	+19
ACS	0 (0)	0 (0)	-1 (-1)	0 (0)	0 (0)	0 (0)	-1 (-1)	+3.5
Cardumen	-1 (+2)	0 (-2)	0 (+1)	0 (+3)	0 (0)	0 (0)	-1 (+4)	-4.2
ARJA	0 (-8)	+2 (-13)	+1 (+2)	+2 (-2)	0 (-1)	0 (0)	5 (-22)	+20.2
SimFix	+1 (-4)	0 (-2)	0 (-10)	+2 (-4)	0 (0)	+1 (+2)	+4 (-18)	+21.2
FixMiner	+2 (-5)	+6 (+13)	+4 (+7)	+5 (+10)	+2 (+2)	+3 (+3)	+22 (+30)	+14.6
AVATAR	+1 (-4)	+2 (-2)	+1 (-2)	+3 (-4)	+2 (+2)	+2 (+3)	+11 (-7)	+26.7
TBar	+3 (-3)	+12 (+12)	+4 (-3)	+5 (-1)	+3 (+3)	+3 (+5)	+30 (+13)	+32.7

- 最初に，正しいバグの位置を提供したときに，各APRツールが修正できるバグの数がどれだけ向上するかを調査した.
- Cardumenを除いて正解率が向上(最大30%)することが分かった.

実験結果

RQ3：障害位置特定ノイズの影響はどのくらいか.

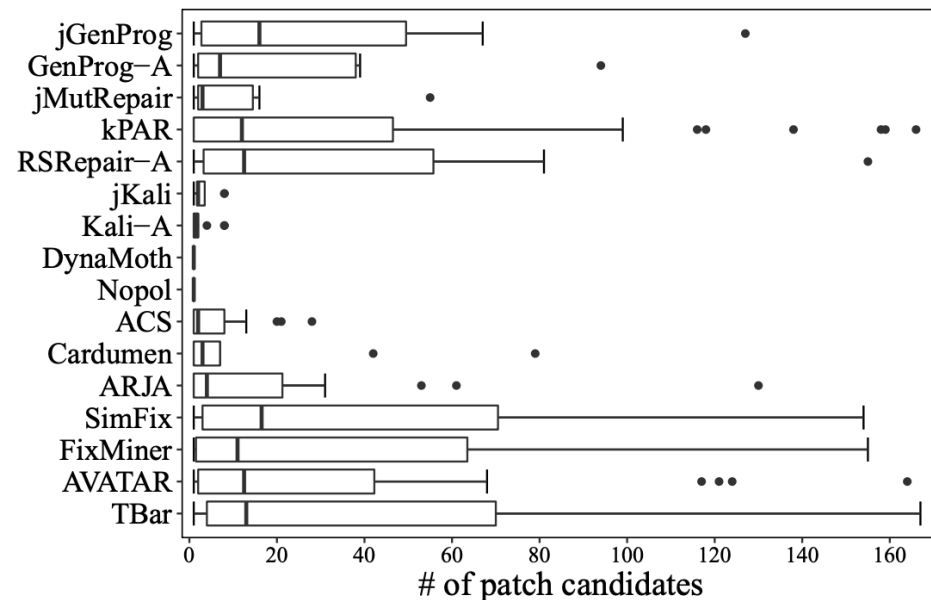


Figure 10: NPC score distribution of each tool given fix positions.

- 図は正しいバグの位置を提供したときのNPCスコアの分布を示している.
- 全体としてスコアの分布範囲が非常に小さくなった.
- SimFixのNPCスコアの中央値は約200から約20にまで減少した.

結論

- ・ この研究ではテストスイートベースのプログラム修正の効率にかんする大規模な調査について報告した.
- ・ バグの位置を正確に提供することでパッチの生成の効率が非常に良くなることが分かった.
- ・ 多くのバグを修正できるツールは, 多くのパッチを生成することが分かった.

所感

- ・ 障害位置の特定にかなりの負担がかかっていることが分かった.
- ・ 最終的なパッチが生成されるまでにかなり多くのパッチ候補が生成されることが分かった.