カバレッジに基づくファジングツールの比較評価

都築 夏樹　吉田 則裕　戸田 航史　山本 椋太　高田 広章

ソフトウェアのテスト手法の 1 つとしてファジングがある．この手法では，テストケースを自動で生成，実行し，テストを自動化できる．現在，カバレッジに基づくファジングツールの開発が盛んに行われており，後発ツールの方が優位であることを示すために，ツール同士の比較評価が行われている．しかし，ベンチマークが評価ごとに異なることが多く，評価に用いられたベンチマークとは別のベンチマークを適用した場合に同様の性能を示すか不明である．そこで，カバレッジに基づくファジングツールの評価に利用された実績がある LAVA-M dataset，Binutils，tcpdump に対して AFL，AFLFast，AFLGo，FairFuzz を適用した．Binutils については，3 種類のバージョンに対して適用した．その結果，LAVA-M に適用した場合に AFL 1.94b が優れた結果を示すといった例外があるものの，統計的には後発ツールが優れていることがわかった．

Much research has been done on coverage-based fuzzing tools that can generate test cases and adapt them to a testing target automatically. However, it is difficult to compare the performance of the tools because different testing targets are often used among experiments of the tools. In this paper, we performed an empirical study of four coverage-based fuzzing tools AFL, AFLFast, AFLGo, and FairFuzz. As benchmarks, we use three collections of testing targets LAVA-M dataset, Binutils and tcpdump with their experimental setting from earlier studies. As a result, we confirmed that newer tools are statistically more effective except in a few cases, such as the application of AFL 1.94b to LAVA-M.

1　はじめに

現在，ファジングと呼ばれるソフトウェアのテスト手法が注目されている．この手法では，テストケースの生成および実行を繰り返すことによって自動的にテストを行う．**AFL** [9] はファジングを実装した代表的なツールであり，**AFL** から派生したツールが開発されている [1] [2]．このツールは，ツール開発時に未発見だった多数の不具合を検出した [9]．これにより，ファジングというテスト手法が盛んに研究されるようになり，**AFLFast** [2] や **AFLGo** [1] などの **AFL** を拡張したツールが開発されるようになった．

新たなツールを開発した場合，先出のツールよりも優れていることを示すため，多くの場合に既存ツールとの比較評価を行っている．例えば，**AFLFast** の評価 [2] では，**AFL** と **AFLFast** が検出したクラッシュ数について比較評価を行っている．しかし，評価に利用されるベンチマークは比較評価ごとに共通でない場合が多い．Klees らの調査 [4] によると，実アプリケーションソフトウェアの中で，最も多く評価に利用されている *Binutils*†1 であっても 32 本のファジングに関係する論文中，4 本の論文でしか利用されておらず，加えてこれらの論文間では異なる *Binutils* のバージョンが使用されている．また，Klees らは利用されるベンチマークが異なる場合，評価結果を論文間で比較することは難しいとも述べている．

そのため，本研究では異なるバージョンや異なるベンチマークの利用が評価に与える影響を調査した．本研究では，評価対象のファジングツールとして

Comparing the Performance of Coverage-based Fuzzing Tools

Natsuki Tsuzuki, Norihiro Yoshida, Ryota Yamamoto, Takada Hiroaki, 名古屋大学, Nagoya University.

Koji Toda, 福岡工業大学, Fukuoka Institute of Technology.

†1　https://www.gnu.org/software/binutils/

AFL および **AFL** を拡張した **AFLFast**，**AFLGo**，**FairFuzz** を利用した．ファジングツールのベンチマークとして，実アプリケーションソフトウェアの中で頻繁に評価対象として利用されている *Binutils* と *tcpdump*，ファジングツールの評価のために開発されたベンチマーク *LAVA-M* を用いた．

本研究では，**AFLFast** や **FairFuzz** の評価実験 [2] [5] の再現実験や *Binutils* の複数バージョンを対象とした拡張実験，本研究で扱うツールの評価実験 [1] [2] [5] に利用されていないベンチマークである *LAVA-M* を対象とした評価実験を行った．本論文中では，**AFL** のバージョン違いである **AFL 1.94b** を **A1**，**AFL 2.40b** を **A2**，**AFL 2.49b** を **A3**，**AFL 2.51b** を **A4**，**AFL 2.52b** を **A5** とする．また，**AFLFast** を **AF**，**AFLGo** を **AG**，**FairFuzz** を **FF** とする．

2 関連研究

2.1 カバレッジに基づくファジング

ファジングは，ソフトウェアのテスト手法の 1 種である．この手法では，テストケースの生成と実行を繰り返すことによって，ソフトウェアのテストを自動化することができる．

ファジングを大別すると，ブラックボックスファジング [8]，ホワイトボックスファジング [3]，グレイボックスファジング [2] に分類される [2]．グレイボックスファジングは，実行時の情報を利用するため，入出力のみを利用するブラックボックスファジングとプログラムの構造などを解析してテストケースを生成するホワイトボックスファジングの中間に位置する手法である [2]．

AFL が行うカバレッジに基づくファジングは，グレイボックスファジングに分類される [2]．カバレッジに基づくファジングでは，実行された基本ブロックの列とクラッシュの有無を取集し，新たなテストケースの生成に利用する．利用者が与えたテストケースから，変異により新たなテストケースを生成する．本論文中では，利用者がツールの実行時に与えたテストケースを初期テストケースと呼ぶ．

表 1　ツールが利用したベンチマーク

ツール	ベンチマーク
AF	Binutils 2.26 (c++filt, nm, objdump, readelf, size, strings), Coreutils
AG	Binutils[†2], Diffutils
FF	Binutils 2.28 (c++filt, nm, objdump, readelf), tcpdump, xmllint, mutool draw, djpeg, readpng

2.2 評価対象ツール

Manes らが調査 [7] したファジングツールの中で，カバレッジに基づいてクラッシュを検出している **AFL** (2013)，**AF** (2016)，**AG** (2017)，**FF** (2018) の 4 種類を評価対象とした．

AFL は，カバレッジに基づくファジングツールのベースであり，**AFL** を拡張した多くのツールが開発されている．

Böhme らは，マルコフ連鎖モデルを用いて **AFL** の低頻度の実行経路（以降，パスと呼ぶ）が存在する問題を説明した．そして，そのようなパスを優先的に実行するように **AFL** を改善したツール **AF** を開発した．

AG は，プログラムの変更やパッチを適用した場合などの回帰テストに利用することを目的として **AFL** を拡張したツールである．プログラム中の利用者が指定した位置に効率的に到達するように拡張している．

FF は，Lemieux と Sen が行った実験 [5] において **AFL** が実現できなかったパスを実現するように，**AFL** を拡張したツールである．変異位置のマスキングを行った後にテストケースを変異させる手法によって実現している．各ツールの評価に利用されたベンチマークを表 1 に示す．

3 実験

ベンチマークの変更が与える影響を調査するために比較評価を行う．

†2　バージョンに関する記述なし

本論文中では，*Binutils* について，*Binutils 2.26* を *B1*，*Binutils 2.28* を *B2*，*Binutils 2.32* を *B3* とする．

評価対象のツールは，再現実験を除いて，**AFL** の最新バージョンである **A5**，**AF**，**AG**，**FF** に加えて，拡張元となった **A3**，**A4**，ツールの比較評価 [2] [5] で利用された **A1**，**A2** を用いた．再現実験では，各ツールの評価で利用された **AFL** と提案されたツールを用いた．

本実験では，特に検出したパス数について評価する．**AFL** の出力結果から得られる評価指標として，検出したパス数以外に検出したクラッシュ数が存在する．これを利用しなかったのは，ツールによって指摘された旧バージョン不具合が，新バージョンで修正されているためである．この場合，旧バージョンで評価を行ったツールの新バージョンでの評価に悪影響を及ぼす可能性がある．

実験に使用した PC の環境は，OS が Ubuntu 16.04.4，CPU が i7-6700@3.40GHz，メモリが 8GB である．実験では，各ツールに 1 コアを割り当てた．

本研究では，プログラムの位置を指定して実行することができる **AG** を，公平な比較評価のために他のツールと同様に位置を指定せずに利用する．

3.1 再現実験

ベンチマークの差異が与える影響を調査するにあたり，下記の実験を行った，

AF の再現実験 *B1* に対して **A1** と **AF** を適用

FF の再現実験 *B2* と *tcpdump* に対して **A2** と **FF** を適用

AG の評価実験 [1] については，ツールの実行結果のみからすぐに実験を再現できないため実施しない．

AF の再現実験では，**AF** の評価実験 [2] で用いられた *B1* の 6 種類のプログラム中から，クラッシュを検出した *c++filt*，*nm*，*objdump* の 3 種類のプログラムを利用した †3．**AF** の評価実験 [2] と同様に実行時のコマンドは `c++filt`，`nm -C`，`objdump -d` を使用し，評価指標として検出したクラッシュ数を用いた．**AF** の評価実験 [2] では，初期テストケースとして 0 バイトのファイルを利用したと述べられているが，第 1 著者が利用を試みたところ異常終了 †4 したため同様に無効な形式の初期テストケースとして改行コードを与えた．各プログラムに対して，**A1** と **AF** をそれぞれ 6 時間ずつ適用した際の検出したクラッシュ数を図 1 に示す．検出したクラッシュ数について，すべてのプログラムにおいて **AF** が上回っており，特に *objdump* では **AF** のみがクラッシュを検出した．この結果は，**AF** の評価実験 [2] と同様の傾向を示した．

FF の再現実験では，**FF** の評価実験 [5] に利用された 9 種類のプログラムの中から，*B2* に含まれるプログラムである *c++filt*，*nm*，*objdump*，*readelf* および *tcpdump* の 5 種類を利用した．**FF** の評価実験 [5] と同様に，初期テストケースは *c++filt* を除いて **AFL** の testcases ディレクトリからプログラムに対応する形式のファイルを与え，実行時のコマンドは `c++filt`，`nm`，`objdump -d`，`readelf -a`，`tcpdump`

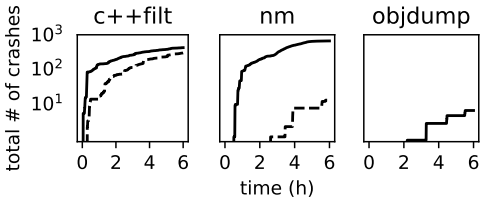

図 1　AF の再現実験の結果：実線 AF，点線 A1

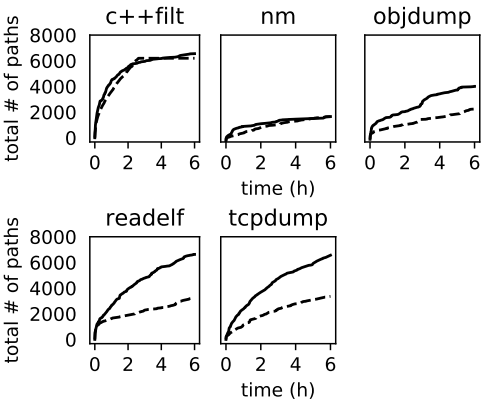

図 2　FF の再現実験の結果：実線 FF，点線 A2

†3　readelf，size，strings は A1，AF ともにクラッシュを検出できなかったため利用しない

†4　No usable test cases というエラーメッセージが表示された

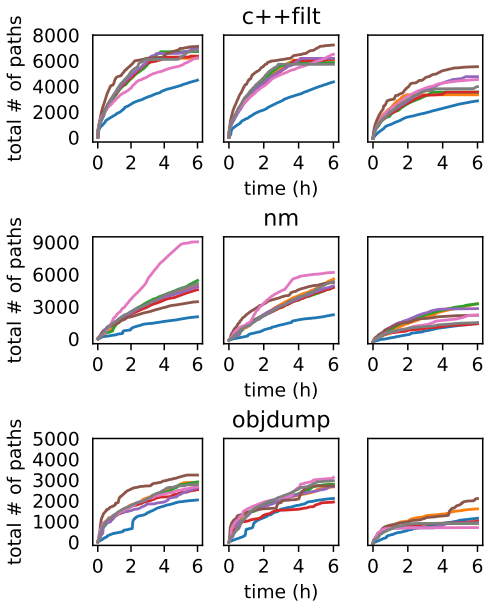

図 3　AF の再現実験を拡張した結果：
左から，*B1*，*B2*，*B3*

-nr を使用した．*c++filt* については，“_Z1fv\n”を与えた．**FF** の評価実験 [5] では，実行された基本ブロック単位での遷移数を評価指標としていたが，ツールの出力結果に含まれないため検出したパス数を評価指標として用いた．各プログラムに対して，**A2** と **FF** をそれぞれ 6 時間ずつ適用した際に検出したパス数を図 2 に示す．**FF** の検出したパス数は，*c++filt* の 2 時間から 4 時間の間を除いて一貫して **A2** を上回った．この結果は，**FF** の評価実験と同様の傾向を示した．

3.2　拡張実験

ベンチマークのバージョンが変更された場合の影響を調査するために以下の実験を行った．

AF の拡張実験　**AF** の再現実験に *B2*，*B3* を追加

FF の拡張実験　**FF** の再現実験に *B1*，*B3* を追加

AF の拡張実験では，**AF** の再現実験で利用された *B1* に含まれるプログラムである *c++filt*，*nm*，*objdump* に加えて，*B1*，*B3* の *c++filt*，*nm*，*objdump* を利用して実験を行う．初期テストケースや各プログラムの実行時のコマンドは，**AF** の再現実験と同様である．ツールを適用した時間は，6 時間である．

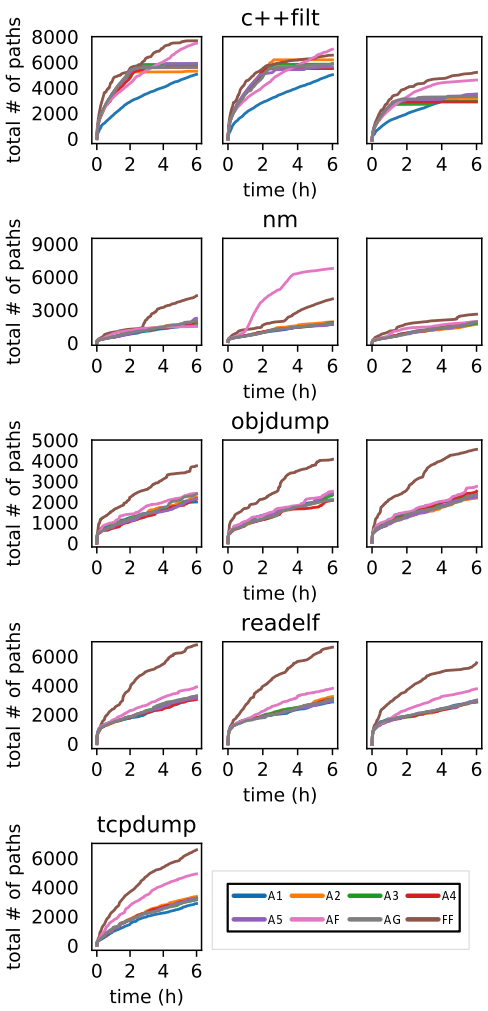

図 4　FF の再現実験を拡張した結果:
左から，*B1*，*B2*，*B3*

FF の拡張実験では，**FF** の再現実験で利用された *tcpdump* および *B2* に含まれるプログラムである *c++filt*，*nm*，*objdump*，*readelf* に加えて，*B1*，*B3* の *c++filt*，*nm*，*objdump*，*readelf* を利用して実験を行う．初期テストケースは，**FF** の再現実験と同様である．ツールを適用した時間や各プログラムの実行時のコマンドは **AF** の拡張実験と同様である．**AF** の拡張実験で利用されていない *readelf*，*tcpdump* は，**FF** の再現実験と同様である．

AF の拡張実験の結果を図 3 に示す．図 3 の結果では，最新バージョンの *B3* において旧バージョンよりも 6 時間経過した時点での検出したパス数が低下す

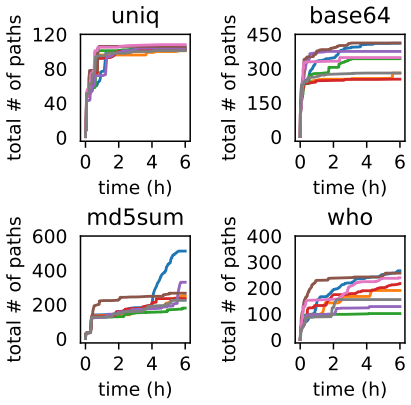

図 5　*LAVA-M* に適用した結果

る傾向があった．特に，**AF** は，他のツールと比較して，最新バージョンの *B3* に含まれる *nm*，*objdump* において検出したパス数が大きく低下した．**FF** の拡張実験の結果を，図 4 に示す．図 4 の結果では，*B2* の *c++filt*，*nm* を除くすべてのプログラムで **FF** が優れていた．また，バージョンによる検出したパス数の減少は，**AF** の拡張実験よりも小さかった．

3.3　LAVA-M ベンチマークでの評価

LAVA-M ベンチマークを利用した場合の影響を調査するために実験を行った．uniq，base64，md5sum，who の 4 種類のプログラムを含む *LAVA-M* をベンチマークとして利用した．このベンチマークは，**AF**，**AG**，**FF** のいずれの評価実験でも利用されていない．実行時のコマンドや初期テストケースは，Li らの実験 [6] を参考にした．ツールを適用した時間は **AF** の拡張実験と同様である．LAVA-M に適用した結果を図 5 に示す．実験の結果，最も古い **AFL** のバージョンである **A1** がいずれのプログラムに対しても最も多くのパスを検出した．

3.4　考察

まず，初期テストケースの影響を考える．図 3 は無効な形式の初期テストケースが与えられており，図 4 は有効な形式の初期テストケースが与えられている．図 3 と図 4 を比較すると，*B1* と *B2* の *nm* において，図 3 より図 4 の検出したパス数が減少した．また，*B3* の *objdump* において，図 3 より図 4 の検出したパス数が増加した．その他については同等であった．この結果から，初期テストケースによって検出したパス数に差があることがわかった．各ツールの評価実験 [2] [5] では，有効な形式のテストケースもしくは無効な形式のテストケースのいずれかを利用している．しかし，初期テストケースの形式によって結果が変わるため，評価では両方の形式を利用すべきだと考えられる．以上から，以下のことがわかった．

> 初期テストケースの形式はパス数に影響するため，有効・無効の両形式を利用すべき

つぎに，ベンチマークのバージョンが異なる場合について考察する．図 3 と図 4 から，図 3 の *B3* の *nm* を除いて **AF** もしくは **FF** が最も検出したパス数が多いことがわかった．図 3 と図 4 の 22 種の結果の中で，**FF** は 16 種，**AF** は 5 種で最も多くのパスを検出した．そのため，いずれのバージョンにおいても **AF** もしくは **FF** が有効であることがわかった．以上から，以下のことがわかった．

> **AF** ないし **FF** が有効である．どちらがより有効かはバージョン間であまり変化しない

最後に，各ツールの相対的な検出したパス数を比較する．3.2 節と 3.3 節の結果について，1 時間ごとの検出したパス数をまとめた結果を図 6 に示す．図 6 では，外れ値は表示していない．集計にあたって，**A5** が 6 時間の時点で検出したパス数が 1 となるように，各結果を除算した．図 6 から，検出したパス数について，すべての時間において，**FF** の検出したパス数が最も多く，次点で **AF** が優れていることがわかった．**AG** は他の **AFL** のバージョンと同程度の結果を示した．ツール間の統計的な有意性を確認するために，Steel-Dwass 検定を行ったところ，以下は有意水準 5%で差が認められた．

- すべての時間において **FF** と **AF** を除くすべてのツール
- すべての時間において **A1** と **AF**
- 3 時間において **A1** と **A3**
- 4 時間，5 時間において **A4** と **AF**
- 6 時間において **A1** と **A5** や **A4** と **A5**，**A5** と

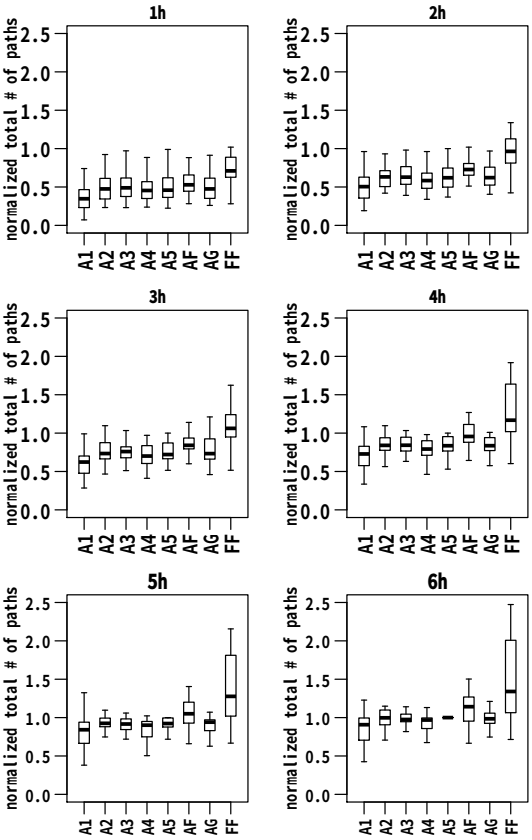

図 6　ツールごとの検出したパス数

AF

この結果から，図 5 のプログラムで A1 が最も検出したパス数が多いといった例外があるものの，統計的には後発ツールが優れていることがわかった．また，**AG** は，**AFL** のいずれのバージョンとも有意差が認められなかった．そのため，本来の目的以外に利用したとしても，**AFL** と同程度の性能を発揮することがわかった．以上から，以下のことがわかった．

> 対象により先発ツールが有効な場合があるが，統計的には後発ツールがより有効である

本実験において，**AF**，**AG**，**FF** の拡張元の **AFL** のバージョンは異なる．この違いが実行結果に影響を及ぼした可能性が考えられるが，**AFL** のバージョン違いによる差よりも，異なるツール間の差のほうが大きいため，影響は小さいと考えられる．

4　まとめ

本研究では，各ツールの評価実験の再現や *Binutils* のバージョン違いを用いた拡張実験，各ツールの評価に利用されていない *LAVA-M* を用いた評価実験を行った，その結果，*LAVA-M* では **A1** の検出したパス数が最も多いといった例外が存在するものの，後発のツールが優れていることがわかった．

謝辞 本研究にあたり，多くの助言を頂いた豊田工業高等専門学校の藤原賢二助教に深謝いたします．

参 考 文 献

[1] Böhme, M., Pham, V.-T., Nguyen, M.-D., and Roychoudhury, A.: Directed greybox fuzzing, *Proc. of CCS*, 2017, pp. 2329–2344.

[2] Böhme, M., Pham, V.-T., and Roychoudhury, A.: Coverage-based greybox fuzzing as markov chain, *IEEE Trans. Softw. Eng.*, (2017), pp. 489–506.

[3] Godefroid, Patrice and Levin, Michael Y and Molnar, David: Automated whitebox fuzz testing., *Proc. of NDSS*, Vol. 8, 2008, pp. 151–166.

[4] Klees, G., Ruef, A., Cooper, B., Wei, S., and Hicks, M.: Evaluating Fuzz Testing, *Proc. of CCS*, 2018, pp. 2123–2138.

[5] Lemieux, C. and Sen, K.: Fairfuzz: A targeted mutation strategy for increasing greybox fuzz testing coverage, *Proc. of ASE*, 2018, pp. 475–485.

[6] Li, Y., Chen, B., Chandramohan, M., Lin, S.-W., Liu, Y., and Tiu, A.: Steelix: program-state based binary fuzzing, *Proc .of FSE*, 2017, pp. 627–637.

[7] Manes, V. J., Han, H., Han, C., Cha, S. K., Egele, M., Schwartz, E. J., and Woo, M.: The Art, Science, and Engineering of Fuzzing:A Survey, *arXiv preprint arXiv:1812.00140*, (2018).

[8] Miller, B. P., Fredriksen, L., and So, B.: An empirical study of the reliability of UNIX utilities, *Commun. ACM*, Vol. 33, No. 12(1990), pp. 32–44.

[9] Zalewski, M.: American fuzzy lop, `http://lcamtuf.coredump.cx/afl`, (2019 年 8 月 5 日閲覧).