特別研究報告

題目

遺伝的アルゴリズムの時間的情報拡張による時間的効率の調査

指導教員

楠本 真二 教授

報告者

皆森 祐希

令和 5 年 2 月 14 日

大阪大学 基礎工学部 情報科学科

令和 4 年度 特別研究報告

遺伝的アルゴリズムの時間的情報拡張による時間的効率の調査

皆森 祐希

内容梗概

ソフトウェア開発における自動プログラム修正は開発工数の半分を占める作業といわれているデバッグの工数を削減することが期待されており，研究が盛んにおこなわれている．自動プログラム修正は，バグを含むプログラムに変更を加えることで用意したテストを通過するプログラムを出力する手法である．自動プログラム修正の手法として，遺伝的アルゴリズムに基づいて修正を行うものがある．ここで，自動プログラム修正における遺伝的アルゴリズムは，目的のプログラムが得られるまでプログラム文の挿入，削除，置換及び交叉を行う手法である．現状の自動プログラム修正における課題の 1 つとして，1 つのプロジェクトに対する実行時間が個体を生成するのにかかった時間をそこで，本研究では遺伝的アルゴリズムを採用した APR ツールのひとつである kGenProg に各個体の生成にかかった時間を計測する処理を追加し，得られた個体情報をビルドの成否および解であるかどうかに分類し，それらを定量的に調べることで全体の生成時間のうち解となるプログラムの生成経路にかかった時間およびビルドに失敗した個体に費やした時間を計算することで，自動プログラム修正の時間的効率性を調査した．

主な用語

自動プログラム修正， 時間計測，可視化， JSON

目次

図目次

表目次

1　はじめに

ソフトウェア開発におけるデバッグ作業は費用および時間的な点で多くのコストを必要とする．ある研究によると，ソフトウェア開発にかかるコストのうち，50% 以上をデバッグが占めるという結果が出ている [1, 2]．自動プログラム修正 (APR) は，人の手を介さずにソースコード中に含まれるバグを全自動で取り除く技術であり，盛んに研究が進められている [3, 4]．APR の実用化に向けて，ここ 10 年で数多くの研究がなされており，時間的なコストを削減する取り組みが数多く行われている [9]．TODO : 時間的コスト削減についての追加の論文調査しかし，先行研究においては個体の生成時間に関する研究は多くない．そこで，本稿では，既存の APR ツールを拡張して個体の生成時間を計測し，独自の指標を実際のバグに対して計算することで時間的コストを調査する．

以降，2 章では APR の課題本研究を行う契機となった研究の説明および研究のための予備知識について説明する．

3 章では今回提案した評価指標とその具体例について論述する．

4 章では実験の概要と結果を提示する．

??章では実験から得られた結果による考察について述べる．

??章では本研究の妥当性への脅威について論じる．

7 章では今後の課題について説明し，

最後に**??**章で総括を述べる．

2　準備

本章では，遺伝的アルゴリズム (GA) に基づく自動プログラム修正 (APR) の生成過程について述べる．

2.1　自動プログラム修正 (APR)

APR は，テストケースと空プログラムを入力として，計算機が自動的にプログラムのバグを修正する技術である．APR は大きく探索ベース[*1]と意味論ベース[*2]の 2 つの手法に分類することができるが，本研究では，探索ベース APR 分野のブレイクスルーとなった GenProg [5] の採用する生成と検証 [**?**] に基づく手法に重点を置く．
この手法では，図**??**TODO：図を用いて詳述するのように，対象となるプログラムにおけるバグの位置を特定する欠陥限局を行い，限局した箇所に変更を加えた後ビルドとテストを実行することで修正できたかどうかの評価を行う．

2.2　遺伝的アルゴリズム

遺伝的アルゴリズム (GA) は各世代で個体を選択し，それらに変異，交叉などの操作を加えることでより強い個体を生成する生物の進化に基づくアルゴリズムである．
TODO：図を用いて詳述する具体的に説明すると，ある世代において，図**??**のような個体があったとする．これらの個体から以下の遺伝子操作を行う．

選択 …個体のうちから何らかの関数による適応度 (APR ではテストスイートの通過率) に応じて選択
変異 …個体の遺伝子を変異させる (APR ではコードの一部を変更)
交叉 …複数の遺伝子の一部分を交配させて新しい遺伝子を生成

APR では，変異において以下の操作を行う．

挿入 …選択したコードの近辺に別のコードを追加する
置換 …選択したコードを別のコードに書き換える
交叉 …選択したコードを削除する

[*1] Heuristics-based
[*2] Semantics-based

2.3 既存の APR ツールの課題

既存の APR ツールは多くの課題が解決されておらず [6]，まだまだ実用には程遠い段階である．具体的には，修正後のプログラムの可読性が低い [7],
その中で今回主に取り上げる課題として，1 回の修正実行にビルドやテスト実行といった多くの時間的コストがかかる点 [8] があげられる．また，先行研究においては，ビルドとテストにかかる時間を削減する研究が行われているものの [9] ，個体の生成そのものにかかった時間を計測する機能に関する研究はまだ発展途上である．

2.4 時間的コスト調査の先行研究

Ghanbari [10] らはソースコードレベルの APR に加えて，さらにバイトコードに APR を施す PraPR を提案し，既存の APR の性能を飛躍的に向上させた．この研究において，提案した PraPR を Defects4J の Chart および Closure バグに対して時間的コストを計算した．結果として，Closure バグでは有効なパッチの数が Chart バグに比べて 10 倍生成されたものの，時間的コストは 20 倍かかった．また，古藤 [9] らは欠陥限局を用いて変更コード片を動的に切り替える手法を提案しビルドに費やす時間を従来手法に比べて 89% から 46% に削減することができ，結果として APR 全体の修正時間の削減につながった．これらの研究をきっかけとして，自動プログラム修正の時間的コストについてより詳らかに調べてみようと思い，本研究をするに至った．

3 評価指標とその実装

3.1 評価指標

3.1.1 STR

APR を実行するにあたり，修正に成功したプロジェクトに対して，その解となる個体に関係する経路の全体の生成時間に占める割合を求める指標として，**STR**(Solution Time Ratio, 解時間比率) を式で定義する．

$$\mathrm{STR} = \frac{\text{解となる個体の経路の総生成時間}}{\text{すべての個体の総生成時間}} \tag{1}$$

ここで，解となる個体の経路の集合は

1. まず解である個体を選択し，集合に入れる
2. 集合内の全個体に対してその親を求め，それらを集合に入れる
3. 集合に変化がなくなるまで 2 を繰り返す

のように求められる．STR を計算する目的として，プログラム修正において成功に必要な個体の生成が時間的にどの程度の割合を占めているのかを定量的に求めることがあげられる．そのため，STR の値は大きい方が好ましい．

3.1.2 FTR

一方で，すべてのプロジェクトに対してビルドに失敗する個体がすべての個体の生成時間に占める割合を求める指標として，**FTR**(Failure Time Ratio, 失敗時間比率) を式で定義する．

$$\mathrm{FTR} = \frac{\text{ビルドに失敗した個体にかかった総生成時間}}{\text{すべての個体の総生成時間}} \tag{2}$$

FTR を計算する目的として，プロジェクト内のプログラムの修正にかかる時間のうちがどの程度の割合を占めるかを定量的に測定し，その傾向を知ることがあげられる．そのため，一般的に FTR の値は小さい方が望ましいとされる．

3.2 STR・FTR の具体的な計算例

先ほど定義した値を求めるための具体的な例として，図 1 の生成木をもつ修正結果について考える．この生成木は，kGenProg の実行結果を記した JSON ファイルをツリー状に表示する Macaw [11] を参考に描画した[*3]．ここで，縦方向は世代を表しており，下に進むにつれてより新しい世代を表す．また，

[*3] X 印における数字の意味合いが違うなど細かい違いがある点に注意

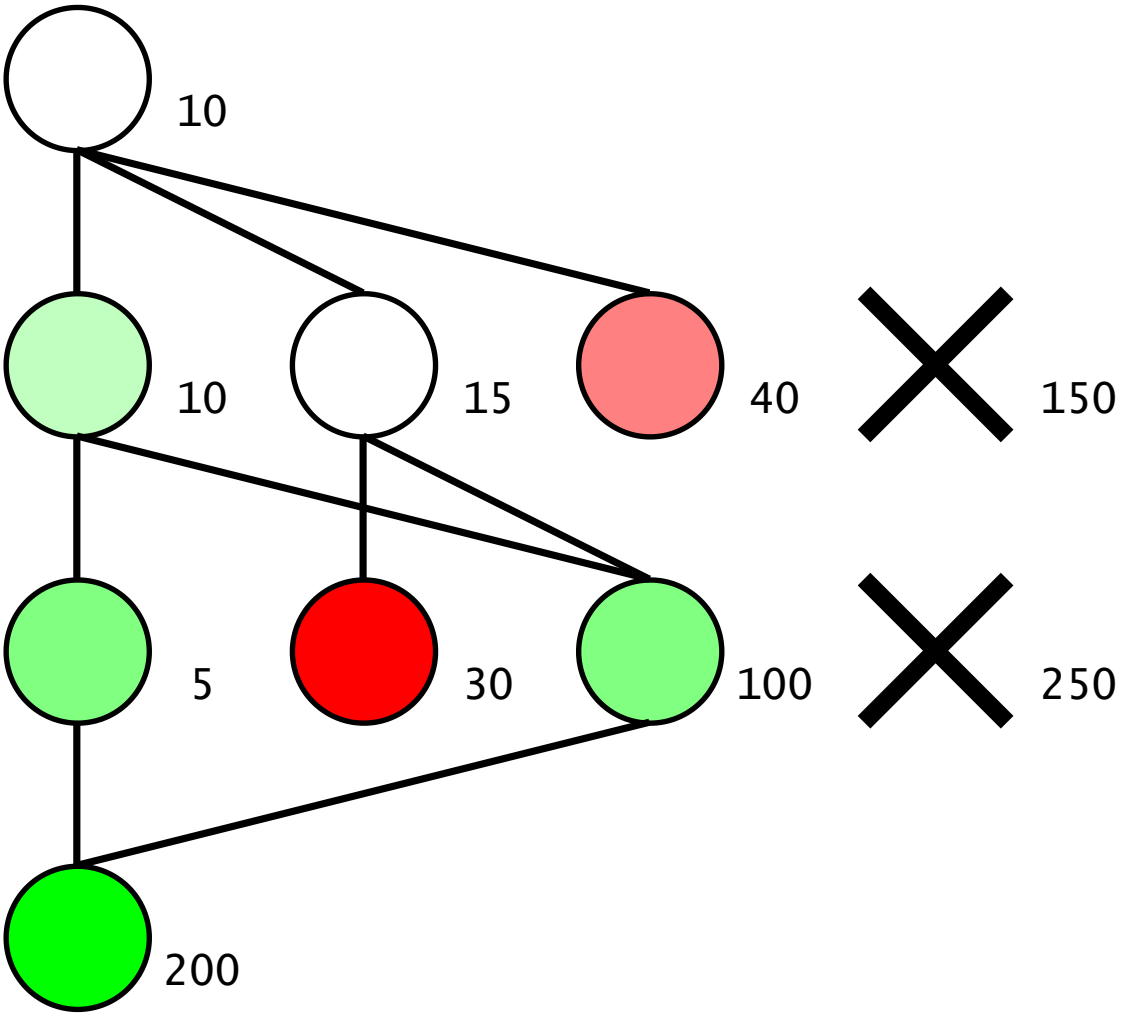

図 1　生成結果の例

横の列は一つの世代におけるすべての個体を表す．円及び X 印はそれぞれビルドに成功した 1 つの個体，その世代でビルドに失敗したすべての個体を表す．円の色は個体の Fitness(全体のテストケースに占める期待通りのテスト結果が得られた割合) を表し，緑に近いほど高い Fitness を右下の数字はその個体あるいは個体の集合の生成時間を表す．なおこの例におけるプログラムの総生成時間は 800 である．この時，STR は図 2 で表される部分の時間 $(10+15+5+100+200) \div 800 = 0.4125$ となり，FTR は X 印で示されたすべての時間を足した値を総生成時間で割ったもの，すなわち $(150+250) \div 800 = 0.5$ と求められる．

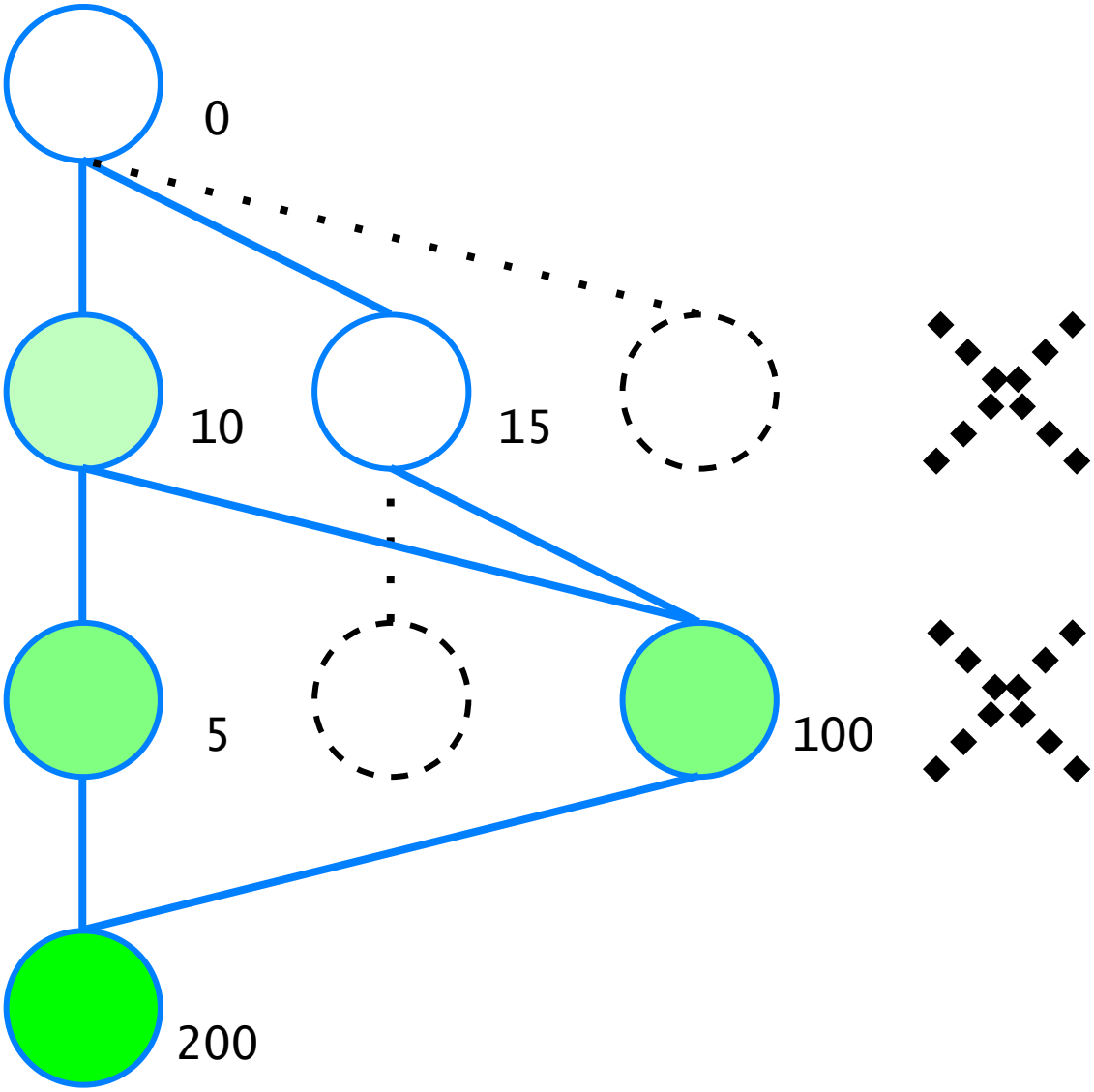

図 2　生成結果の例

3.3　APR ツールへの時間計測機能実装

提案手法では，既存の APR ツールである kGenProg [12] を拡張して実装する．ソースコード内部のふるまいは変えずに，`Mutation` クラスと `RandomCrossover` クラスに時間計測を行うコードを挿入した．具体的には，`org.apache.commons.lang3.time.StopWatch` クラスをインポートし，各個体を生成するループの最初に `StopWatch.createStarted()` メソッドを呼び出すことによって時間計測を開始，処理を終えた後に `getTime()` メソッドを呼び出すことで生成時間を取得し，それを個体情報に格納する．この際，kGenProg に付属している JSON ファイルの出力オプションをオンにすることで，プ

ロジェクトにおける個体の解析を可能にする．

次に，JSON ファイルを Python で記述したプログラムを用いて処理し，各個体の ID(通し番号)・生成時間・Fitness(ただしここでは ID に対応する個体がビルドに失敗した場合-1 を格納する) の情報を取得した後，その情報をもとに STR と FTR を計算する．

具体的には，出力となる JSON ファイルを読み込み，そのプロジェクトで生成された個体の時間を 1 つずつ取得する．この時，Fitness の値で追加の処理を行う．例えば Fitness が-1 であればビルドに失敗した個体であるので失敗時間 (FTR を計算する際の分子) に加算する．また，Fitness が 1 であれば，テストケースを満たす解となる個体であるので 3.1.1 で挙げた手順で解となる個体の親を求める．プログラム中では再帰的なアルゴリズムを用いている．

4　実験

4.1　概要

本章では，GA を採用した APR ツールである kGenProg [12] を用いて，Defects4J [13] の Lang プロジェクトの Lang1〜Lang44 を対象に自動プログラム修正を行った．そのうち，ビルドに成功し，かつ解を得ることができた Lang6, Lang22, Lang25 および Lang39 の 4 つのプロジェクトを対象として先ほど定義した STR と FTR を計算し，その値を確認する．なお，生成時間には不確定性があるため各プロジェクトごとに APR を複数回実行している．表 1 に APR ツール実行時の設定を，表 2 に実行回数や解に至るまでの総個体数など，各プロジェクトに対する実験の条件を示す． ここで，サンプル数がプロジェクトによって異なるのは，1 回のプログラム修正にかかる総時間が異なるためである．

4.2　実験結果

ここもう少し詳細に書く￥

表 1　APR ツールの設定

項目	値
実験題材	Defects4J Lang6, Lang22, Lang25, Lang39
題材数	4
乱数シード	2
実験環境	Corei5-1240P 16GB mem

表 2　各プロジェクトの詳細

プロジェクト名	Lang6	Lang22	Lang25	Lang39
到達世代数	1	7	1	4
個体数	8	624	38	300
ビルド失敗個体数	7	511	36	274
サンプル数	100	15	70	40

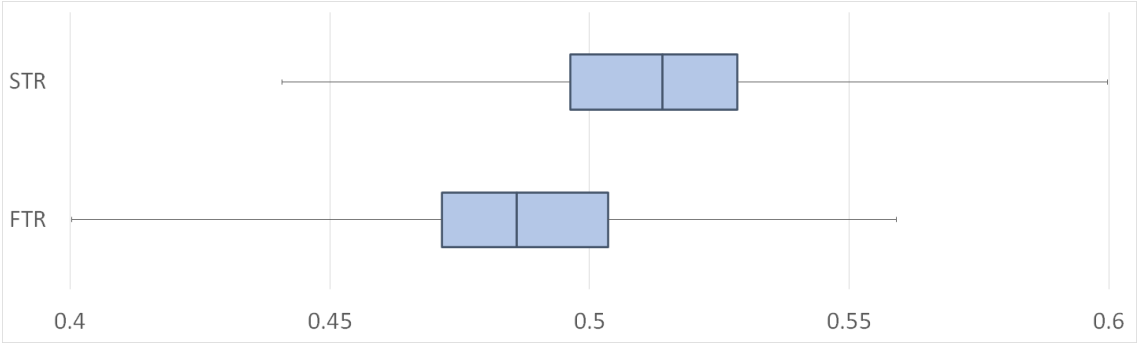

図 3　箱ひげ図：Lang6

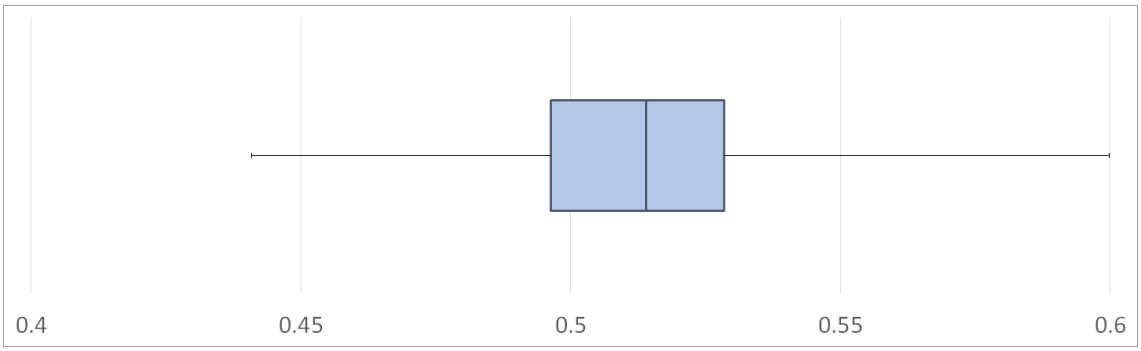

図 4　STR の箱ひげ図：Lang6

4.2.1　Lang6

図 3 に Lang6 における STR と FTR の箱ひげ図を，表 3 に STR と FTR の平均 (以降 AVG)，最小値 (以降 MIN)，第 1 四分位数（以降 1Q)，中央値 (以降 MED)，第 3 四分位数 (以降 3Q)，最大値 (以降 MAX) の各データを示す.

4.2.2　Lang22

図 6 に Lang22 における STR と FTR の箱ひげ図を示す.

表 3　Lang6 プロジェクトの STR, FTR

評価指標	AVG	MIN	1Q	MED	3Q	MAX
STR	0.5117	0.4409	0.4964	0.5140	0.5284	0.5997
FTR	0.4883	0.4003	0.4716	0.4860	0.5036	0.5591

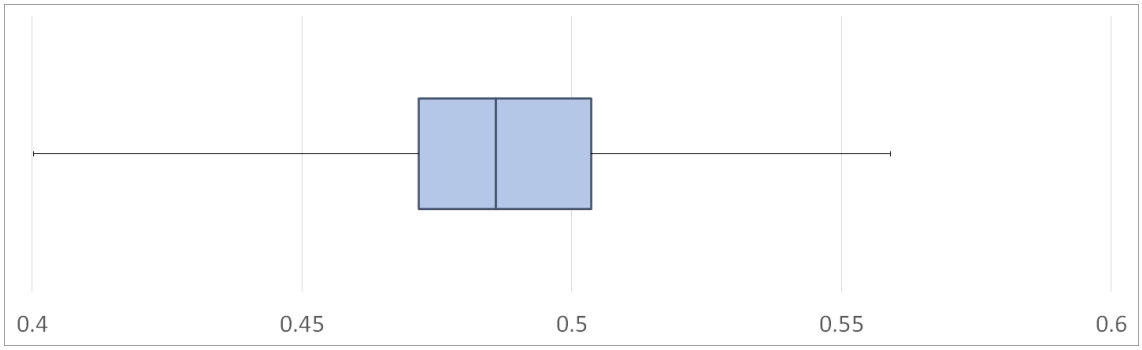

図 5　FTR の箱ひげ図：Lang6

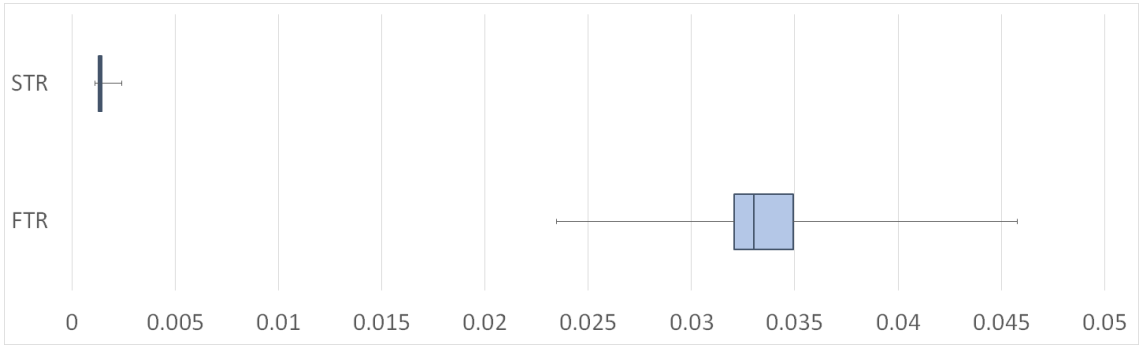

図 6　箱ひげ図：Lang22

4.2.3　Lang25

図 9 に Lang25 における STR と FTR の箱ひげ図を示す.

表 4　Lang22 プロジェクトの STR, FTR

評価指標	AVG	MIN	1Q	MED	3Q	MAX
STR	0.001476	0.001096	0.001291	0.001354	0.001458	0.002418
FTR	0.03432	0.02346	0.03208	0.03304	0.03492	0.04578

表 5　Lang25 プロジェクトの STR, FTR

評価指標	AVG	MIN	1Q	MED	3Q	MAX
STR	0.09386	0.07261	0.08936	0.09106	0.1070	0.1177
FTR	0.7986	0.7371	0.7725	0.7989	0.8053	0.8512

図 7　STR の箱ひげ図：Lang22

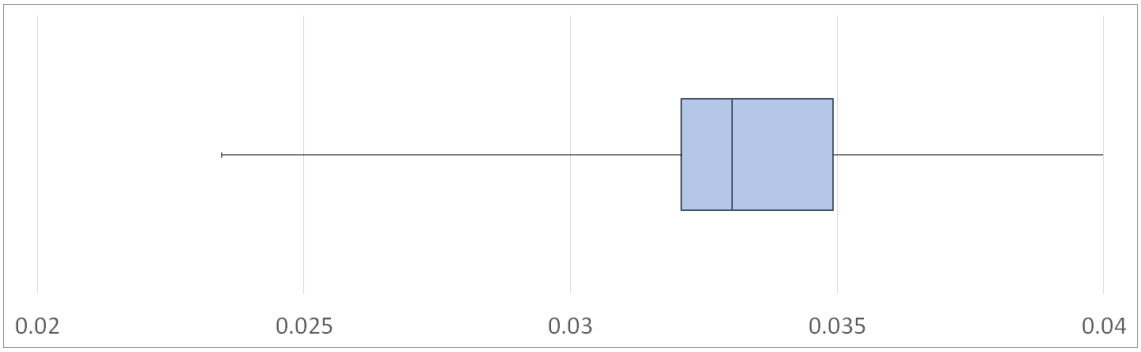

図 8　FTR の箱ひげ図：Lang22

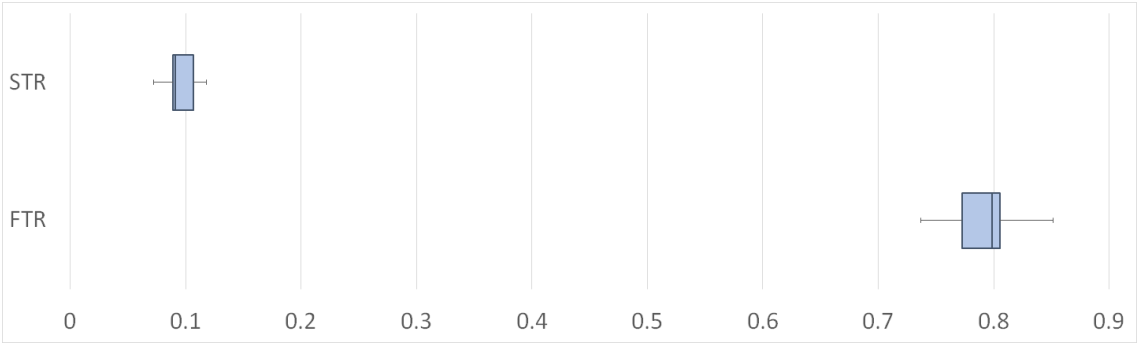

図 9　箱ひげ図：Lang25

4.2.4　Lang39

図 12 に Lang39 における STR と FTR の箱ひげ図を示す.

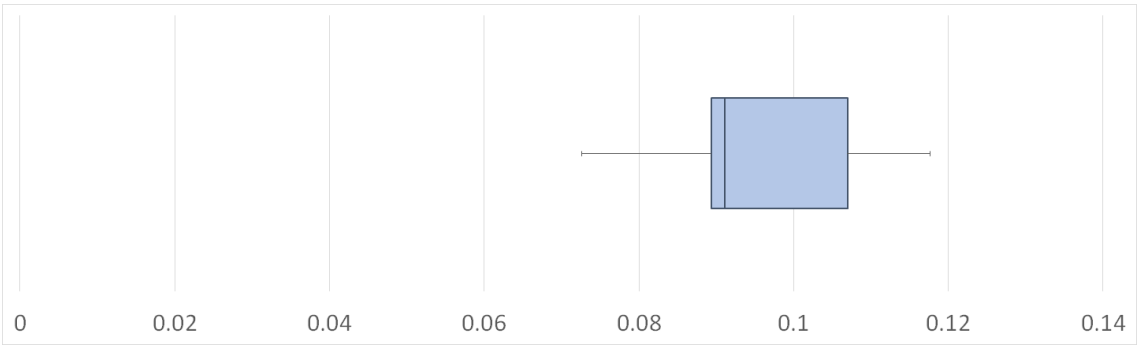

図 10　STR の箱ひげ図：Lang25

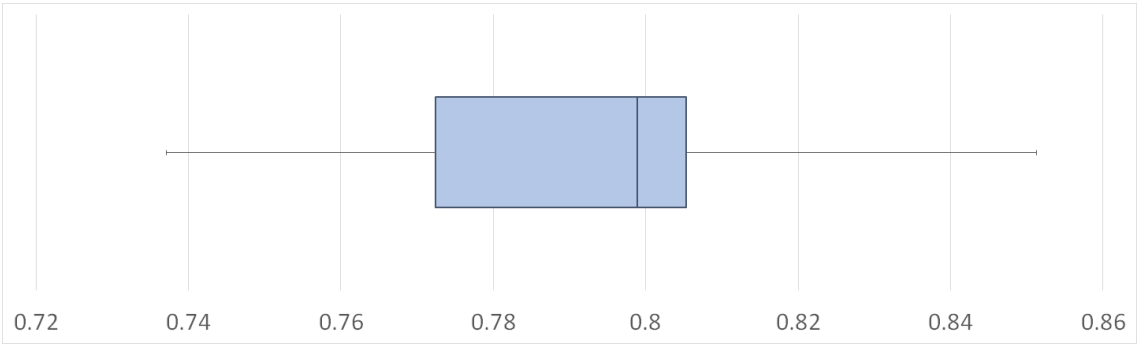

図 11　FTR の箱ひげ図：Lang25

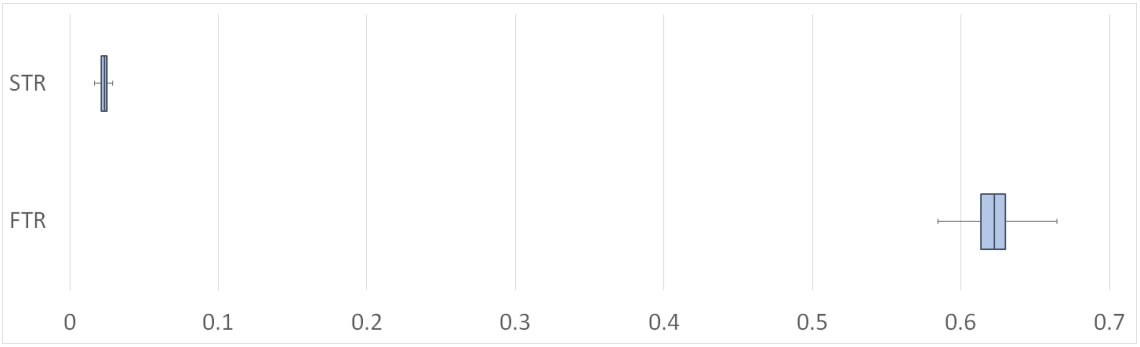

図 12　箱ひげ図：Lang39

表 6　Lang39 プロジェクトの STR, FTR

評価指標	AVG	MIN	1Q	MED	3Q	MAX
STR	0.02406	0.01645	0.02105	0.02326	0.02510	0.02873
FTR	0.6218	0.5847	0.6137	0.6224	0.6299	0.6644

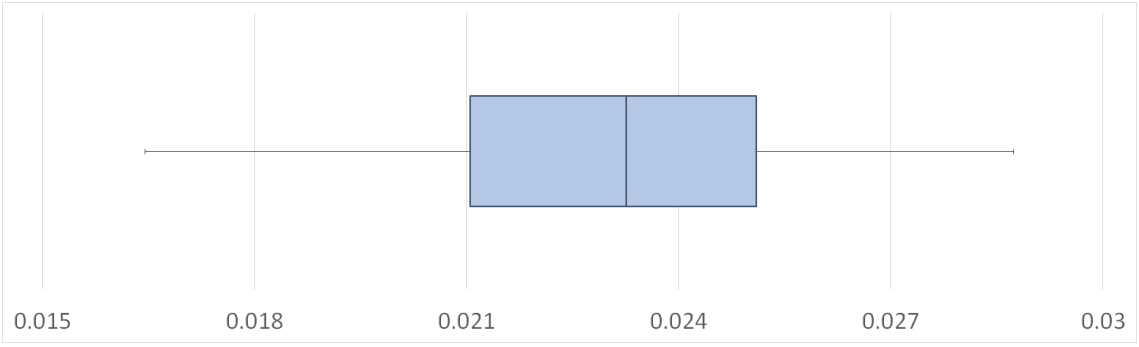

図 13　STR の箱ひげ図：Lang39

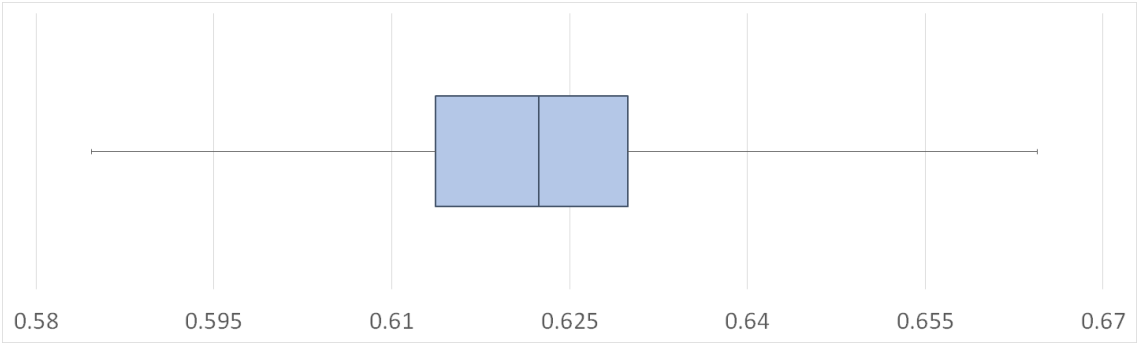

図 14　FTR の箱ひげ図：Lang39

5　考察

次に，4 章で行った実験の結果から，各プロジェクトおよび全体的な STR と FTR の傾向について考察する．まず，Lang6 プロジェクトについて述べる．修正を完了するまでに生成した個体の数が 8 つと比較的少ないこともあってか，次に，Lang22 プロジェクトの修正においては，他のプロジェクトの修正により多くの時間および操作を要した．そのため，他のプロジェクトに比べると STR，FTR のいずれの値も小さくなっている．ついで，Lang25 プロジェクトの特徴として，

6　妥当性の脅威

4 章における実験において考えうる妥当性の脅威について論ずる．この実験では内的要因と外的要因に大別される．まず内的要因として，今回実行した環境とは異なる環境において実行した際に算出される STR と FTR の値が異なる可能性がある点があげられる．また，実行時に他のタスクによりメモリが占領されている場合，普段と異なる値が算出されうる点があげられる．

次に外的要因として，大規模なプロジェクトにおいて何万といった個体が生成されたときに STR と FTR の値が大きく異なる今回対象としたプロジェクトは個体数が多いもので 600 程度と比較的少なかった．

7　今後の課題

7.1　APR ツールによる生成時間の違いの検証

本研究においては，APR ツールとして探索ベースであり，かつ遺伝的アルゴリズムを採用した kGenProg を対象に個体の生成時間計測を行ったが，遺伝的アルゴリズムでないほかの探索ベースの APR ツールや意味論ベースの APR ツールにおいて時間計測を行う．

7.2　時間計測を行う対象プロジェクトの拡張

本研究では，Defects4J の Lang バグのうち，4 つのプロジェクトを対象として STR と FTR の計算を行った．今後は，Defects4J のほかのバグやその他の対象について時間研究を行う．

8　おわりに

本稿において，APR ツールに個体の生成にかかった時間を計測する

謝辞

本研究を進めるにあたり，多くの方々からご支援およびご助言を賜りました．

楠本 真二 教授には，本研究を快く快諾し，暖かく見守ってくださりました．心より感謝申し上げます．

肥後 芳樹 准教授には，議論を重ねに重ね，本研究の完成のご支援及び的確なご助言を賜りました．心より感謝申し上げます．

柗本 真佑 助教には，テーマが決まらず途方に暮れていた際，鋭くも的確なご助言を賜りました．深く感謝いたします．

古藤 寛大先輩には，困難に直面した際いつも迅速にご助言を賜り感謝してもしきれません．

本研究に至るまでに，講義，演習等でお世話になりました大阪大学基礎工学部情報科学科の諸先生方に，御礼申し上げます．

最後に，これまでお世話になりました家族，小中高校の教員方，その他すべての方に感謝申し上げます．

参考文献

[1] B. Hailpern and P. Santhanam. Software debugging, testing, and verification. *IBM Systems Journal*, Vol. 41, No. 1, pp. 4–12, 2002.

[2] Tom Britton, Lisa Jeng, Graham Carver, and Paul Cheak. Quantify the time and cost saved using reversible debuggers. Technical report, Cambridge Judge Business School, 2012.

[3] Luca Gazzola, Daniela Micucci, and Leonardo Mariani. Automatic software repair: A survey. In *Proceedings of the 40th International Conference on Software Engineering*, pp. 1219–1219, 2018.

[4] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. Automated program repair. *Communications of the ACM*, Vol. 62, No. 12, pp. 56–65, 2019.

[5] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. GenProg: A Generic Method for Automatic Software Repair. *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 54–72, 2012.

[6] Claire Le Goues, Stephanie Forrest, and Westley Weimer. Current challenges in automatic software repair. *Software quality journal*, Vol. 21, pp. 421–443, 2013.

[7] Edward K Smith, Earl T Barr, Claire Le Goues, and Yuriy Brun. Is the cure worse than the disease? overfitting in automated program repair. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 532–543, 2015.

[8] Liushan Chen, Yu Pei, and Carlo A Furia. Contract-based program repair without the contracts. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 637–647. IEEE, 2017.

[9] 古藤寛大, 肥後芳樹, 柗本真佑, 楠本真二. 変更コード片の動的切替による自動プログラム修正のビルド時間削減の試み. 電子情報通信学会技術研究報告, Vol. 120, No. 407, pp. 19–24, 2 2021.

[10] Ali Ghanbari, Samuel Benton, and Lingming Zhang. Practical program repair via bytecode mutation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 19–30, 2019.

[11] Yuya Tomida, Yoshiki Higo, Shinsuke Matsumoto, and Shinji Kusumoto. Visualizing code genealogy: How code is evolutionarily fixed in program repair? In *2019 working conference on software visualization (VISSOFT)*, pp. 23–27. IEEE, 2019.

[12] 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. Asia-Pacific Software Engineering Conference*, pp. 697–698, 2018.

[13] René Just, Darioush Jalali, and Michael D Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 international symposium on software testing and analysis*, pp. 437–440, 2014.