6K-09

事後条件を用いたプログラム自動修正のための適応度関数の試作

伊東雄策† 鷲崎弘宜† 深澤良彰†
早稲田大学†

1. はじめに

プログラム自動修正技術の中で特に注目されているものが, 遺伝的アルゴリズムを用いる手法 [1]である. この技術は, 候補プログラムの集団にランダムな操作を加えることを繰り返すことで, バグが解消されたプログラムを生成する. この手法は修正に成功するまでに時間がかかりやすい点が課題であり, 複雑なバグを修正することが困難である.

自動修正の効率化のため, 本研究ではプログラム個体の評価値を表す適応度の改善を目指す. 既存研究では人が用意したテストの通過率を適応度としているが, この方法はプログラムの最終的な結果のみを用いるため, 粗い評価しか与えられない. 実際に, 適応度が限られたいくつかの値のみに集り, 個体の良さを区別できないことが指摘されている [2].

本研究では Java を想定し, テスト結果に加えて, プログラムの内部状態を反映した適応度関数を提案する. 内部状態を調べるため, メソッドごとに事後条件の候補をランダムに生成し, テスト実行時の各条件の判定結果をもとに適応度を計算する.

2. プログラム自動修正

遺伝的アルゴリズムによるプログラム自動修正では, 全てのテストに合格するプログラムが発見されるまで, 以下の操作を繰り返し行う.

自動バグ限局：バグが存在する可能性が高い行を特定し, 行ごとに疑わしさを表す疑惑値を計算する.

変異・交叉：新たな個体を生成する. 変異は疑惑値の高い行をランダムに変更する操作, 交叉は 2 つの個体の間で一部を交換する操作である.

Fitness Function Using Postconditions for Automated Program Repair

Yusaku Ito†, Hironori Washizaki†, Yoshiaki Fukazawa†
Waseda University†

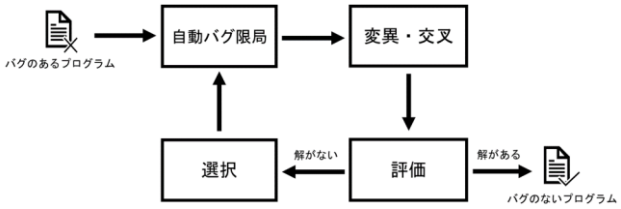

図 1 プログラム自動修正の流れ

評価：生成された個体をテスト実行し, 結果をもとに評価値を計算する. この評価値を適応度と呼ぶ.

選択：現在の個体の集団から適応度の高い少数の個体を選び, 残りの個体を破棄する.

3. 提案手法

適応度関数の改善のため, テスト以外の仕様情報を併用することを考える. 本手法は, 自動的かつ短い計算時間で仕様を抽出することを実現する.

3.1. 事前処理

疑惑値を持つ行のある全メソッドに対して, 利用可能な変数を組み合わせて, x==0 や x>=y など単純なパターンに沿って条件式を作成する. その中から一定個数をランダムに選び, 事後条件候補とする.

3.2. 事後条件の自動評価

ある事後条件候補 c が成り立つ個体と成り立たない個体で, どちらがテストを通過しやすいか比較することで, c が正しい仕様かどうかを推定する. 一定世代が経過するごとに, 以下のように 2 つのパラメータ correctness および incorrectness を計算する.

correctness(c) = (条件 c を常に満たす個体の平均評価)−(それ以外の個体の平均評価)

incorrectness(c) = (条件 c を一度も満たさない個体の平均評価)−(それ以外の個体の平均評価)

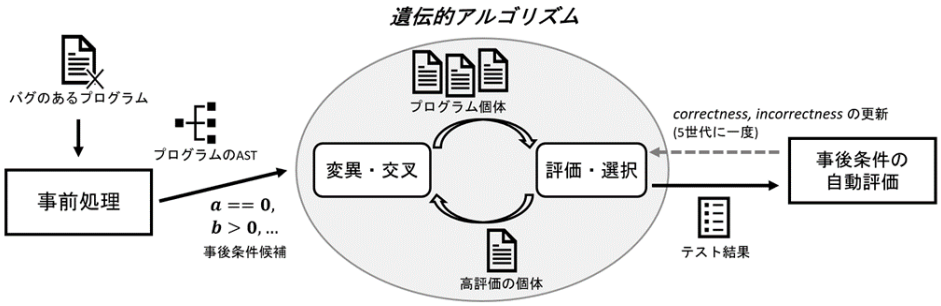

図 2 提案手法の概要

correctness(c)が高い場合は c が正しい仕様である可能性が高く，incorrectness(c)が高い場合は c の否定が正しい仕様である可能性が高いと考えられる．

3.3. 適応度関数

個体をテスト実行し，テスト通過率と事後条件候補の判定結果の両方から適応度を計算する．事後条件候補 c が常に真の場合は correctness の値だけスコアを加算し，c が常に偽の場合は incorrectness の値だけ加算する．結果を 0 から 1 の範囲に正規化し，テスト通過率との積を最終的な適応度とする．

4. 適応度関数の妥当性の評価

自動修正向けのサンプルプログラム[1]を利用して事後条件候補の評価結果が適切かどうかを実験によって確かめた．また，正解までの距離と適応度の間に適切な相関($r \leq -0.15$)があるか調査した．比較対象としては自動修正ツール kGenProg [3]を用いた．

30 世代目の時点で，事後条件候補の評価結果と正解プログラムが満たす仕様を比較した結果，明らかに誤っていたものは全体の 3.33%であり，ほとんどの場合に妥当な評価を与えることができた．

一方，相関係数の値にはほぼ変化が見られず，いずれも $r \leq -0.15$ の条件を満たさなかったため，適応度関数が十分に妥当になったとはいえない．

表 1 距離と適応度の相関(FDC)の計測結果

実験対象	世代	FDC	
		kGenProg	提案手法
GCD01	16-20	-0.081	-0.051
	26-30	-0.128	-0.088
	36-40	0.012	0.065
QuickSort01	16-20	0.022	0.039
	26-30	0.022	-0.001
	36-40	0.046	0.029

[1] https://github.com/kusumotolab/kGenProg/tree/master/example

5. プログラム修正効率への影響の評価

現実的なバグを修正する際に修正効率が向上するか調べるため，Defects4J Math データセット[2]を用いて実験を行い，修正効率を kGenProg と比較した．

制限時間内に修正に成功した割合を比較すると，11 個のバグのうち 1 個のみで成功率が改善し，5 個で成功率が低下した．また，修正成功までの必要世代数の平均値も増大した．全体として，適応度関数の改善によって修正効率は向上しなかった．

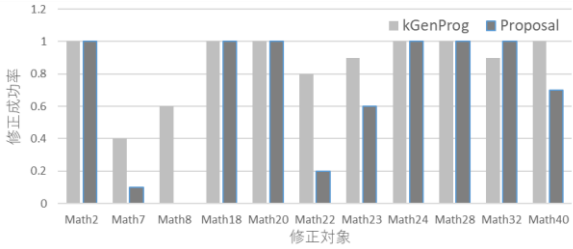

図 3 修正成功率の比較

6. まとめ

本研究では，プログラムの内部変数の状態を反映した適応度の計算方法を提案した．提案手法は事後条件を適切に推定することができた一方，適応度の妥当性や修正効率は改善されなかった．今後は，事後条件の推定結果をバグ限局のために利用する等，別の形で自動修正に活用することを検討している．

参考文献

[1] Weimer, W. et al.: GenProg: A Generic Method for Automatic Software Repair, IEEE Transactions on Software Engineering, vol.38, no.1, pp.54-72 (2012).

[2] Eduardo Faria de Souza et al.: A novel fitness function for automated program repair based on source code checkpoints, GECCO 2018, pp.1443–1450 (2018).

[3] Y. Higo et al. : kGenProg: A High-Performance, High-Extensibility and High-Portability APR System, APSEC 2018, pp.697-698 (2018).

[2] https://github.com/rjust/defects4j