

Semantic Bug Seeding: A Learning-Based Approach for Creating Realistic Bugs

Jibesh Patra

University of Stuttgart Germany jibesh.patra@gmail.com

Michael Pradel

University of Stuttgart Germany michael@binaervarianz.de

In Proceedings of the 29th ACM Joint Meeting on
European Software Engineering Conference and
Symposium on the Foundations of Software Engineering
(ESEC/FSE 2021). 906–918

背景

- バグ埋め込みは様々な場面で利用される
 - 例) バグベンチマーク, ミューテーション, 機械学習への利用等
- 構文変換パターンに基づいた手法が構築されてきたが, 現実的なバグを埋めこめない

目的・アプローチ

- 目的

- 意味を考慮したバグ埋め込み手法を構築

- アプローチ

- 埋め込むためのバグに実際のバグを利用する
- 埋め込み先の文脈を考慮するためにトークン埋め込みを利用する

バグ埋め込み技術

Table 1: Comparison with other bug seeding techniques.

Approach	Kinds of bugs	of Target locations (C1)	Adaptation to target location (C2)	Unbound tokens (C3)
Mutation operators [24, 43]	Few, manually defined	Everywhere	Syntactic	Not supported
Inferred mutat. operators [9]	Many, inferred	Everywhere	Syntactic	Not supported
Neural machine translation [61]	Many, inferred	Implicit by model	Implicit by model	Not supported
Bug synthesis [15, 53]	Memory updates	Hard to trigger paths	N/A	N/A
This work	Many, inferred	Based on semantic fit	Semantic	Supported

手法概要

1. バグパターン作成

2. 意味的にマッチする埋め込み可能箇所の特

3. バグの適用

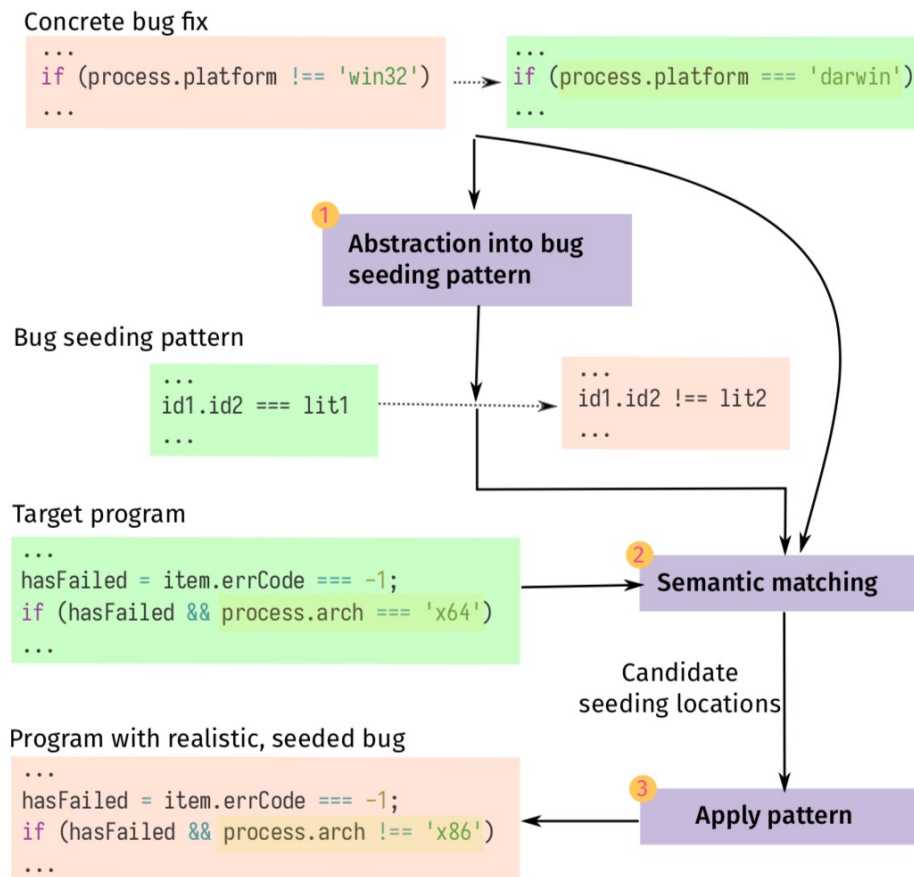


Figure 1: Overview of the approach and running example.

バグパターン作成

- バグ修正シーケンスペア

- (C_{bug}, C_{corr})

- $C_{bug} = [t_1^b, \dots, t_n^b], C_{corr} = [t_1^c, \dots, t_n^c], t : \text{トークン}$



識別子 $\rightarrow id_k$
文 $\rightarrow lit_k$

- バグ修正パターン

- (P_{corr}, P_{bug})

`(process.platform !== 'win32', process.platform === 'darwin')`



`(id1.id2 === lit1, id1.id2 !== lit2)`

埋め込み可能箇所の特定制

1. 埋め込み先コードのトークン化, 抽象化

- [item, ., errCode, ==, -1], [process, ., arch, ==, 'x64']
- [id1, ., id2, ==, lit1]

2. 構文的に埋め込み可能箇所の特定制

- バグパターンと構文的に一致する箇所を特定制
- [id1, ., id2, ==, lit1]

1. 意味的に埋め込み可能箇所の特定制

- 自然言語処理を用いて埋め込むバグと意味的に最も類似する箇所の特定制
- [process, ., arch, ==, 'x64']

バグの適用

- unbound token

- バグシーケンスには現れるが、正常シーケンスには現れないトークン
- $(P_{corr}, P_{bug}) = (id1.id2 === lit1, id1.id2 !== lit2)$
- 正確に対処できないと意味的なバグ埋め込みが不可能

バグの適用

- 類推*を用いたトークンバインディング

- トークン埋め込みを用いて青い点を求める

- 青い点を平均することでピンクの点を求める

- ピンクの点から最も近いトークンを求める

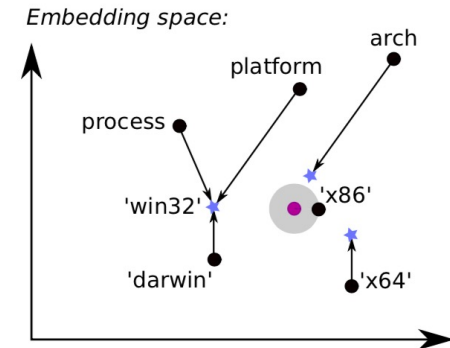


Figure 2: Example of analogy queries to bind unbound tokens.

$$V_{tgt} = \{emb('x64') + emb('win32') - emb('darwin'), \\ emb(arch) + emb('win32') - emb(platform), \\ emb(process) + emb('win32') - emb(process)\}$$

*類推：「日本」と「東京」の関係から「フランス」で連想できる単語は？－「パリ」

評価実験

- RQ1：実際のバグの再現するのにSemSeedはどれほど効果的か
- RQ2：既存手法の違いとどのように比較されるか
- RQ3：パラメータの影響は何か
- RQ4：埋め込まれたバグは学習にどれほど役立つか
- RQ5：既存研究で埋め込まれたバグとどう違うか
- RQ6：SemSeedはバグ埋め込みにおいてどれほど効率的か

データセットとパラメータ

- データセット
 - GitHubからJavaScriptのプロジェクト100件
 - 3,600件のバグ修正
 - ガイドバグ：2,880件
 - 保留バグ：720件
- パラメータ
 - 意味的な類似度 m
 - 0.2
 - unbound tokenを選択するためのトークンセット T
 - 1,000
 - 場所ごとに埋め込むバグの数 k
 - 10

RQ1：バグ再現

- 埋め込まれたバグと保留バグを比較し，現実性の高いバグの割合を計算
 - 比較するバグを2つの条件で絞る
 - バグパターンはガイドバグと保留バグの両方で現れる
 - unbound tokenがトークンセットに含まれる
- 53件のバグを比較
- 53件中，47件のバグを再現できた
 - 再現できたバグの内，31件はunbound tokenを含む

RQ2：既存手法との違い

- 以下の機能を既存のものに置き換えた場合と比較
 - 埋め込み可能箇所の特定制
 - 全ての構文的に一致する箇所に
 - バグの適用
 - unbound tokenのバインドをトークンセットからランダムに選択
- 53件中, 16件のバグしか再現できなかった
 - 62%のバグパターンを埋め込み損なった

RQ3：パラメータの影響

- パラメータ m
 - 意味を考慮して埋め込み可能箇所を特定する方が再現の効果が高い
- パラメータ T と k
 - T と k に応じて29から47のバグを再現する

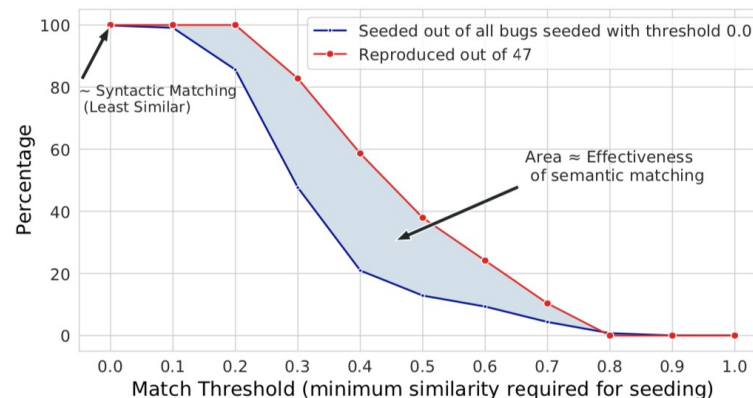


Figure 3: Influence of matching threshold m on seeded bugs.

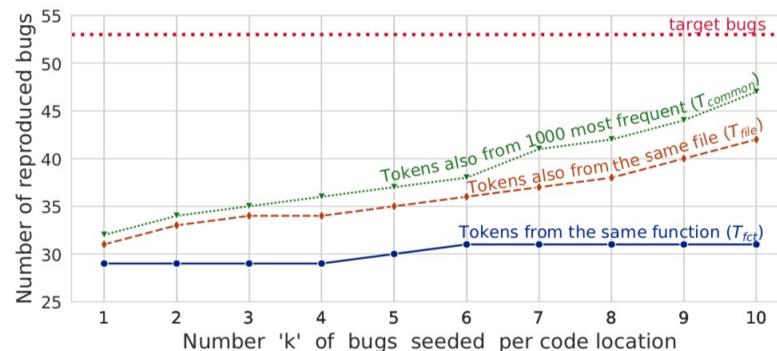
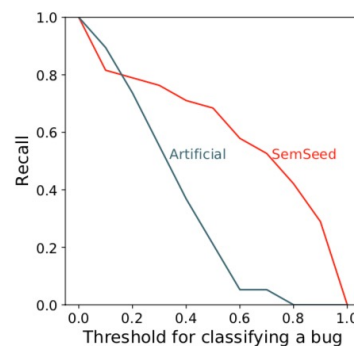


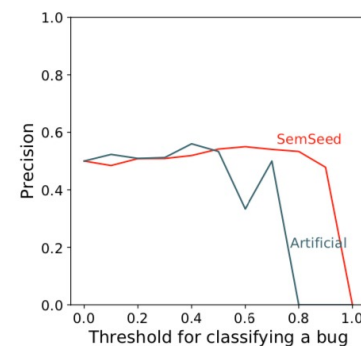
Figure 4: Reproduced real-world bugs depending on token set T and number k if bugs to seed.

RQ4：学習への効果

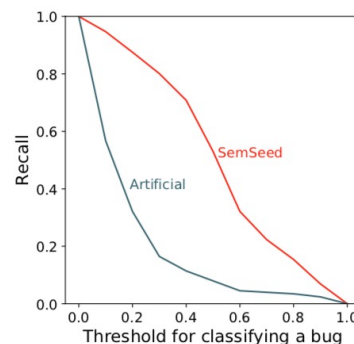
- DeepBugs[49]の学習データとして通常のバグを使う場合とSemSeed再現バグを使う場合で比較



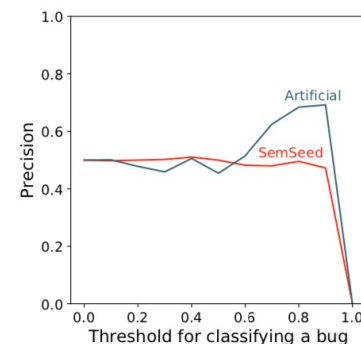
(a) Recall wrong assignments.



(b) Precision wrong assignments.



(c) Recall wrong binary operand.



(d) Precision wrong binary operand.

Figure 5: Precision and recall of DeepBugs with artificially seeded and SemSeed-seeded bugs.

RQ5：埋め込まれるバグの違い

- 既存手法 (Mutandis[43], Major[24]) で埋め込んだバグパターンと既存手法のバグパターンを比較
- 2,880のガイドバグから677,217のバグをランダムに選んだ1,000のjsファイルに埋め込む
- 98.2%のバグは既存手法で定義された23種類のバグパターンと異なる
- 事前定義されたバグパターンセットを超えてバグ埋め込みすることで既存手法を補完

RQ6：効率

- RQ5の処理を以下のマシンで実行し時間を計測
 - CPU：48 Intel Xeon E5-2650 CPU cores
 - Memory：64GB of memory
- 677,217のバグの埋め込みに140分かかることがわかった
- 1つのバグを埋めるのに0.01秒かかる

まとめ

- 自然言語処理を利用することで意味を考慮したバグ埋め込み手法を構築
- 評価実験から
 - 既存手法では扱えなかったバグパターンからバグ埋め込みが可能になり、埋め込めるバグが大幅に増えた
 - 学習用データとして利用できる
 - 短時間で実行できる

所管

- ミューテーション解析に利用する場合、結果がどう変わるか気になった
- 埋め込みバグが他の学習データとしてどれほど使えるか気になった