機械学習を用いたソフトウェア安定性分析に関する研究

M2020SE002 可知敬朗
指導教員：野呂昌満

1　はじめに

OSS（Open Source Software）は，通常複数のユーザによって頻繁に変更が加えられ，その結果，ソフトウェアの構造が複雑化し，変更による影響の把握が困難になる．これを解決する一方法として，利用関係の観点からソフトウェア全体の構造を明らかにし，変更による影響を分析する試みがある [3, 6].

Horwitz ら [7] によれば，ソフトウェアの構造を議論する場合に，依存関係に着目したグラフ抽象化をすることで，ソフトウェアの特性が理解できるとされている．他方，ソフトウェアの安定性は，プロジェクトやパッケージ単位など様々なレベルで分析されてきた [3, 6]．パッケージに関するメトリクスから，その時間的変化に着目して安定性を分析した研究 [1] があるが，計測される安定性はパッケージの変更に伴う局所的なものである．この点に関し Constantinou ら [3] は，外部安定性と内部安定性の 2 つのメトリクスにより，局所的，大域的両方の観点から安定性を分析している．我々は Constantinou らにならい，コンポーネント間の利用関係（Use）により安定性の議論をすることができると考える．

本稿では，モジュールとその依存関係の観点からソフトウェアをグラフとして抽象化し，大域的な特性を分析する新たな方法を提案する．この方法では，ソフトウェアの特徴を定量化するために，計測対象ソフトウェアを抽象化したグラフに対して表現学習の研究成果 [8] を利用し，特徴ベクトルを抽出する．ベクトルをクラスタリングし，計測対象ソフトウェアの代表ベクトルを特定する．代表ベクトルとクラスタの時間的変化を観測することで，ソフトウェアの局所的な安定性のみならず，大域的な安定性の議論ができる．

以上をまとめると，本研究の目的は，ソフトウェアのグラフ表現に基づきその特性を定量的に取り扱うことである．具体的な研究課題を次の 2 点とする．

RQ1: モジュールの利用関係のための依存関係グラフモデルはどのように定義できるか

RQ2: 依存関係グラフの特徴ベクトル表現によりソフトウェアの安定性はどのように分析できるか

2　ソフトウェアの安定性とその分析

2.1　外部安定性と内部安定性

Constantinou ら [3] は，変更に基づくソフトウェアの大域的な安定性を分析している．モジュールとその利用関係を，モジュールをノード V，モジュール間の利用関係をエッジ E のグラフ $S =< V, E >$ として抽象化した．進化再利用の観点からモジュール利用関係の変化に基づき，アーキテクチャに関連する安定性について次の 2 つの尺度を定義した．

- 外部安定性 (ES: External Stability)
  $ES(i, i+1)$ はバージョン i から $i+1$ にかけての外部安定性指標であり，モジュールの変更に関する計測値から $[0, 1]$ で表現される．0 の場合は既存モジュールが全て削除されたこと，1 の場合は全て再利用されたことを示す．
- 内部安定性 (IS: Internal Stability)
  $IS(i, i+1)$ はバージョン i から $i+1$ にかけての内部安定性指標であり，モジュール利用関係の変更に関する計測値から $[0, 1]$ で表現される．0 の場合は既存の利用関係が全て削除されたこと，1 の場合は全て再利用されたことを示す．

外部安定性と内部安定性により，ソフトウェアの安定性を定量的に表すことが可能である．Constantinou らは，これら 2 つの指標からソフトウェアの大域的な安定性が定義できるものとしている．

2.2　グラフ／ネットワーク表現学習

グラフ／ネットワーク表現学習（Graph/Network Representation Learning）[2] とは，グラフ構造の特性を分散表現（特徴ベクトル）として抽出する機械学習の方法の一つである．代表的な例として graph2vec[8] がある．

graph2vec は，ニューラルネットワークを用いたグラフサンプリングを実現するためのシステムである．サブグラフにラベルを割り当てグラフコーパスを作成し，学習を行うことで特徴ベクトルを抽出する．意味が近いサブグラフ同士は，ベクトルとして抽出された後にもベクトル空間で近い位置に配置されるという特長を持つ．この特長から Narayanan ら [8] は，ベクトルをクラスタリングし有効な分類タスクが実行できると述べている．

2.3　分析の問題点と技術課題

Constantinou らの定義では外部安定性と内部安定性は $[0, 1]$ の範囲で示されるが，ここで，複数のメトリクスから計算される安定性を一次元の $[0, 1]$ の範囲で説明可能かどうか疑問が生じる．

本研究では複数の観点から分析されうる安定性を，グラフ抽象化とグラフ表現学習によるベクトル表現として定量化する．グラフ表現学習を用いてグラフの時間的変化をベクトルで表現 [2] し，その分析によりソフトウェアの安定性を示すことが可能かどうかを課題と考える．

3　グラフモデルの表現学習を用いた安定性分析

本研究では，表現学習による特徴ベクトルの抽出，クラスタリングによる分析を経ることにより，ソフトウェア安定性の定量分析を行う．この方法により，Constantinou らの定義した外部安定性と内部安定性を，$[0, 1]$ の範囲よりも詳細に定量化することを目指す．本研究におけるソ

フトウェア安定性分析の概要を図 1 に示す．

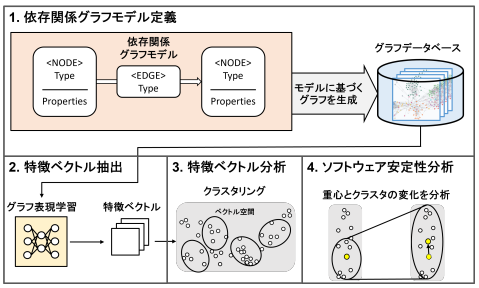


図 1: 提案する安定性分析の概要

3.1 分析プロセス

本研究では，ソフトウェアの構造を定量的に扱う基盤ができれば，安定性の議論を行う事が出来ると考えたので，実験科学のアプローチに基づく次のプロセスを分析のプロセスとする．

1. 仮説，分析内容設定
   設定した仮説に基づき，分析項目を設定する．
2. グラフの生成
   本プロセスは，図 1 に示した「依存関係グラフモデル定義」に対応する．Git からのデータ収集と，グラフの生成，データベースへの格納を行う．
3. フィーチャ分析
   本プロセスは，図 1 の「特徴ベクトル抽出」,「特徴ベクトル分析」に対応する．graph2vec によるベクトル抽出，k-means 法によるクラスタリングを行う．
4. 安定性分析
   本プロセスは，図 1 に示した「ソフトウェア安定性分析」に対応する．クラスタと重心の時間的変化を分析し，大域的な安定性の推移を観測する．
5. 仮説検証
   分析結果から，設定した仮説を検証する．必要に応じ，得られた結果に基づいて，仮説の追加や変更を行い，一連のプロセスを繰り返す．

これらの 5 プロセスに従うことにより，グラフやクラスタリング結果といった本研究における成果物が明確になり，安定性分析に関する議論が可能になる．

3.2 グラフの生成

ソフトウェアの時間的変化をグラフの変化として保存するために，グラフデータベース Neo4j[9] にてグラフを生成する．グラフに関する定義を表 1 に示す．

分析に関して，次に示す理由から，分析対象を Python で記述された OSS としても一般性を失わないと考える．

- 利用関係を明確にできればいいので，使用言語を問う必要がない
- 分析対象リポジトリが豊富に存在する

同ソフトウェア内の他のモジュールを import することを本研究では Use エッジと表現し，利用関係に該当するものとする．Composition エッジは構成関係とディレクトリの階層の表現のために導入している．

表 1: ノードのプロパティとエッジの型の定義

属性/型	記述項目
name	名称
def	ディレクトリ or ファイル
path	モジュールのパス
Use	利用関係，他のモジュールを import
Composition	構成関係，ディレクトリの階層を表現

3.3 フィーチャ分析

グラフのベクトル表現を得るために graph2vec[8] を利用する．グラフからサブグラフを抽出し，モジュールまたは利用関係の削除，再利用をラベルとしてグラフに関するフィーチャ抽出を行う．各サブグラフは 1,024 次元のベクトルとして変換される．抽出されたベクトルはモジュールまたは利用関係の削除，再利用を情報として含む定量的な表現であり，graph2vec の特長から意味の近いベクトル同士はベクトル空間の近い位置に存在する．ベクトルに対し k-means 法を行うことで，抽出されたベクトルの重心，すなわち計測対象ソフトウェアの代表点を求めることができる．

3.4 安定性分析

3.3 で得られたクラスタと重心について，その時間的変化から外部安定性と内部安定性を分析する．図 2 に，本研究の分析と Constantinou ら [3] の定義した ES 値／IS 値の比較について示す．クラスタの分散について，削除に関するクラスタの分散値が増加した場合，元のソフトウェアではモジュール／利用関係が多く削除されたことを意味するので，ES 値／IS 値は 0 に近づく．再利用に関するクラスタの分散値が増加した場合，モジュール／利用関係が多く再利用されたことを意味するので，ES 値／IS 値は 1 に近づく．重心の移動距離について，連続するバージョン間で重心の移動距離が大きい場合，クラスタの構成が変化したと考えられ，ES 値／IS 値も大きく変動すると考えられる．モジュール／利用関係への変更をノード／エッジに対する変更に置き換え，表現学習によって特性を抽出することにより，安定性の議論を行うことができる．

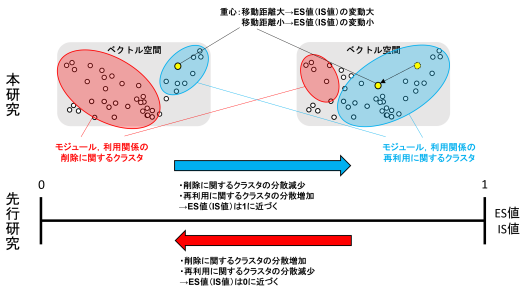


図 2: クラスタと重心の変化に基づく安定性の変化

4　プロトタイプシステムの実装と OSS に対する分析方法の適用

本章では，前章で説明した分析方法を支援するプロトタイプシステムを実装し，OSS に対する分析を試みる．

4.1　実装の目的

プロトタイプシステム実装の目的は次の 3 点である．

1. ソフトウェアをモジュール利用に基づくグラフとして抽象化し，グラフデータベースにグラフとして格納．
2. graph2vec により，グラフの特徴ベクトル表現抽出．
3. ベクトルに対して k-means 法を適用することで，分析対象のソフトウェアの安定性を分析．

4.2　プロトタイプシステムの構成

図 3 にプロトタイプシステムの構成を示す．グラフの保存にはグラフデータベース Neo4j[9] を利用し，グラフの生成には Neo4j で提供されているクエリ言語 Cypher Query を利用した．特徴ベクトル抽出の表現学習 graph2vec[8] は，作者の GitHub 公開ページ [1]より引用した．特徴ベクトル分析において，k-means 法を用いたクラスタリングをするために，scikit-learn を利用した．

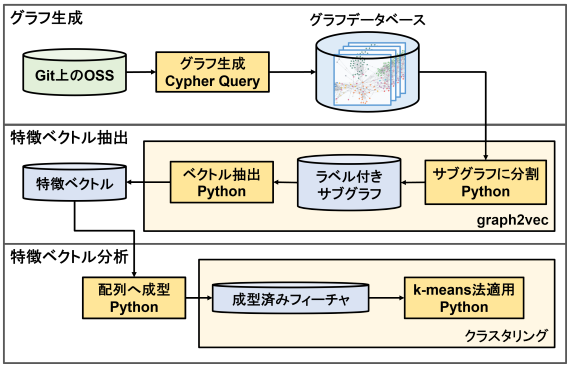


図 3: 分析プロトタイプシステムの構成

4.3　分析対象の OSS

実現したプロトタイプシステムを用い，GitHub で公開されている 2 つのグラフ表現学習に関連する OSS に対して，提案方法を適用した．

1. DGL (Deep Graph Library)[4]
2. PyTorch Geometric[10]

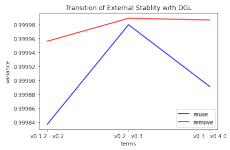
これらの OSS を分析対象とした理由は，グラフ表現学習の実装という同じ目的を持ち，設計思想の異なる両 OSS に関して，安定性の差異を分析するためである．

4.4　DGL（Deep Graph Library）の安定性分析結果

DGL は Distributed (Deep) Machine Learning Community が開発を行う，グラフ表現学習の実装を提供する OSS である [4]．提供が開始された 2018 年 12 月 7 日の v0.1.2 から 2019 年 10 月 7 日の v0.4.0 までのリリースにおいて，主要なアップデートである v0.1.2，v0.2，v0.3，v0.4.0 を対象とし，分析を行った．

4.4.1　DGL：クラスタの分散

v0.1.2 - v0.2，v0.2 - v0.3，v0.3 - v0.4.0 のバージョン移行について，外部安定性（ES），内部安定性（IS）に関連するクラスタの分散の値の推移を図 4 に示す．



(a) ES に関する分散推移　(b) IS に関する分散推移

図 4: 分散の推移 (DGL)

4.4.2　DGL：重心の移動距離

各クラスタの重心ベクトルについて，連続する分析期間における重心ベクトル同士のユークリッド距離を求め，重心ベクトルがどれだけ移動したかを分析する．

v0.1.2 - v0.2 を期間 1，v0.2 - v0.3 を期間 2，v0.3 - v0.4.0 を期間 3 とし，分析結果を表 2 に示す．

表 2: 重心の移動距離（DGL）

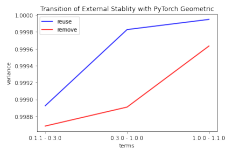
	期間 1 - 期間 2		期間 2 - 期間 3	
	ES	IS	ES	IS
再利用クラスタ	33.66	31.64	35.33	34.78
削除クラスタ	32.37	35.28	32.06	33.44

4.5　PyTorch Geometric の安定性分析結果

PyTorch Geometric は，PyG が開発を行う，グラフ表現学習の実装を提供する OSS である [10]．提供が開始された 2018 年 5 月 25 日の 0.1.1 から 2019 年 4 月 1 日の 1.1.0 までのリリースにおいて，主要なアップデートである 0.1.1，0.3.0，1.0.0，1.1.0 を対象とし，分析を行った．

4.5.1　PyTorch Geometric：クラスタの分散

0.1.1 - 0.3.0，0.3.0 - 1.0.0，1.0.0 - 1.1.1 のバージョン移行について，外部安定性（ES），内部安定性（IS）に関連するクラスタの分散の値の推移を図 5 に示す．



(a) ES に関する分散推移　(b) IS に関する分散推移

図 5: 分散の推移 (PyTorch Geometric)

4.5.2　PyTorch Geometric：重心の移動距離

DGL の分析と同様の方法で重心の移動距離を求める．

[1] https://github.com/MLDroid/graph2vec_tf

0.1.1 - 0.3.0 を期間 1，0.3.0 - 1.0.0 を期間 2，1.0.0 - 1.1.0 を期間 3 とし，分析結果を表 3 に示す．

表 3: 重心の移動距離（PyTorch Geometric）

	期間 1 - 期間 2		期間 2 - 期間 3	
	ES	IS	ES	IS
再利用クラスタ	39.04	33.13	32.12	33.25
削除クラスタ	34.12	35.64	33.90	35.87

5　考察

5.1　研究課題に関する考察

RQ1 について，本稿では OSS を対象にグラフ抽象化を行い，時間的変化をグラフデータベースに保存することで，ソフトウェアの特性の変化をグラフの変遷として定義した．RQ2 について，表現学習を用いた特徴ベクトル抽出によりグラフを定量表現に変換し，クラスタリングと時系列分析から安定性を分析した．

分析から得られたクラスタの分散値，重心の移動距離の変化を利用し，表 4 に示す ES 値または IS 値の変動が把握できることから，安定性の推移を観測できる．表 4 の 0 または 1 方向大は，ES 値／ IS 値が 0 または 1 に大きく近づいたことを示す．0 または 1 方向小は，ES 値／ IS 値が 0 または 1 にやや近づいたことを示す．さらに重心とその移動距離から，ソフトウェアの代表点が一定であるか否か，その遷移を分析できると考える．

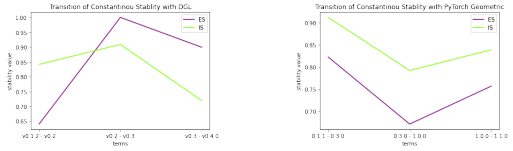
表 4: 本研究のメトリクス

	削除クラスタ		再利用クラスタ	
	分散増	分散減	分散増	分散減
距離大	0 方向大	1 方向大	1 方向大	0 方向大
距離小	0 方向小	1 方向小	1 方向小	0 方向小

本研究の分析では，Use 関係に着目したグラフのベクトル表現から安定性に関する議論を行った．これに加えてグラフの Composition 関係に着目すると，パッケージ内の Use とパッケージ外からの Use を分析できるので，モジュール強度や独立性についても議論できると考える．

5.2　Constantinou らのメトリクスとの比較

Constantinou らの定義に基づいて安定性を計測した結果を図 6 に示す．



(a) DGL　　(b) PyTorch Geometric

図 6: Constantinou らの定義に基づく ES 値／ IS 値

先行研究の方法で計測した ES 値／ IS 値は，本研究による安定性の計測結果と完全には一致しなかった．この点について，本研究では先行研究では計測しきれなかった特性を分析できた可能性があると考える．先行研究ではモジュールと利用関係がどれだけ削除／再利用されたかによって安定性を計測しており，この観点は本研究におけるベクトルのクラスタの分散値の増減と対応する．本研究では，この観点に加え重心とその移動距離を用いることにより，ソフトウェアの代表点の遷移を分析できる．以上から，本研究ではクラスタの分散値，重心の変化を分析の観点とすることで，先行研究では分析できなかったソフトウェアの特性を分析可能であると考える．

6　おわりに

本研究では，ソフトウェアのグラフ表現に基づく特性を定量的に扱い安定性として分析するために，グラフ抽象化とグラフ表現学習を用いた大域的な分析方法を提案した．安定性の議論を行うために RQ1，RQ2 を研究課題として掲げた．

本提案により，先行研究 [3] に基づく安定性を，グラフのベクトル表現を用いて詳細に分析した．成果として，ES 値／ IS 値の変動のみならず，ソフトウェアの代表点の変遷を含めた安定性を観測できる点が挙げられる．

今後の課題は，他の大規模 OSS への本提案の適用，分析期間の拡大，Composition 関係を観点に加えたさらなる分析である．

参考文献

[1] Alenezi, M. and Khellah, F. : Architectural Stability Evolution in Open-Source Systems, *Proc. of ICEMIS 2015*, Article No.: 17, Sep. 2015, pp. 1-5.

[2] 浅谷 公威, 他 : ネットワーク表現学習によるネットワークの成長可視化, 情報処理学会第 186 回知能システム研究会, Vol. 2017-ICS-186, No. 6, Mar. 2017, pp. 1–6.

[3] Constantinou, E. and Stamelos, I. : Architectural Stability and Evolution Measurement for Software Reuse, *Proc. of SAC 2015*, Apr. 2015, pp. 1580–1585.

[4] Distributed (Deep) Machine Learning Community. : Deep Graph Library, `https://www.dgl.ai/`.

[5] Facebook Inc. : PyTorch, `https://pytorch.org/`.

[6] Fayad, M. E. and Altman, A. : An Introduction to Software Stability, *Commun. ACM*, Vol. 44, No. 9, Sep. 2001, pp. 95–98.

[7] Horwitz, S. and Reps, T. : The Use of Program Dependence Graphs in Software Engineering, *Proc. of ICSE 1992*, Jun. 1992, pp. 392–411.

[8] Narayanan, A., et al. : graph2vec: Learning Distributed Representations of Graphs, *Proc. of MLG 2017*, Aug. 2017.

[9] Neo4j, Inc. : Neo4j, `https://neo4j.com`.

[10] PyG. : PyTorch Geometric, `https://pytorch-geometric.readthedocs.io/en/latest/`.