6K-07

コールグラフの機械学習によるソフトウェア健全性分析方法の提案と評価

可知 敬朗† 牧野 槙一郎† 青山 幹雄†
南山大学理工学部ソフトウェア工学科†

1. 研究背景と課題

OSS は複数のユーザによって頻繁に変更が加えられている. そのため, ソフトウェアの構造が複雑化し, 変更による影響が不透明になりやすい. 従って, 呼出し関係にあるプログラム全ての影響を分析する必要がある. プログラム全体の呼出し関係をコールグラフモデルで表現できるが, 時間的推移に着目して大域的な特性を分析する研究は少ない. ネットワーク構造の影響分析や変化予測には, 対象とした構造をグラフモデルとして捉え, その特性を分析することが有用である. グラフの大域的な特性によってソフトウェアの良さが明らかになることが期待できる. 本研究では大域的な特性を健全性とする.

以上の研究背景を踏まえ次の 3 点を研究課題とする.

RQ1: コールグラフモデルの定義
RQ2: コールグラフの機械学習によるソフトウェア健全性分析方法の提案
RQ3: 提案方法の有効性, 妥当性の評価

2. 関連研究

(1)ネットワーク表現学習

ネットワーク表現学習[1]とはグラフ構造からノードやエッジ, サブグラフの特徴量を分散表現として獲得する方法である. 一方法として graph2vec[5]がある.

(2)グラフデータベース(GraphDB)

GraphDB は, グラフ構造を管理できる DB である. 代表的な GraphDB として Neo4j がある.

(3)機械学習による OSS コミュニティのグラフモデル分析

先行研究[4]は機械学習を用いて, OSS コミュニティの開発者の活動に関する特徴量を獲得し, その進化構造を分析する方法を提案している.

3. アプローチ

OSS の健全性を観測するためには, プログラムの呼出し関係を含めた大局的分析が必要である. プログラムの呼出し関係をコールグラフでモデル化し, 機械学習の一方法の表現学習を適用することで, グラフの特徴量を獲得しソフトウェアの健全性を評価する(図 1).

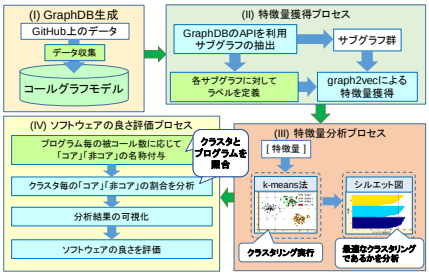

図 1 アプローチ

A Health Analysis Method of Software Systems Using Machine Learning on Call Graph Models and Its Evaluation
†Takaaki Kachi, Shinichiro Makino, Mikio Aoyama, Department of Software Engineering, Nanzan University.

4. 提案方法

提案プロセスは次の 6 ステップから成る.

(1) コールグラフモデル定義
対象とする機械学習 OSS ライブラリを基にコールグラフモデルを定義する. 本研究の対象は GitHub の Python プログラムとする.
(2) 仮説分析内容設定
(3) GraphDB 実装
(4) 特徴量獲得, 特徴量分析
(5) ソフトウェアの健全性評価
(6) 仮説検証

4.1 ソフトウェアの健全性評価

4.1.1 ソフトウェア変更の健全性

プログラムの変更時, 呼出し関係にある他のプログラムにも影響が及ぶ. 呼び出される回数が多いプログラムほどその影響は大きく, ソフトウェア内で重要な機能を持つと考えられる. よってプログラムを次の 2 つに分類する.

(1) コア:呼び出されている回数が多いプログラム
(2) 非コア:呼び出されている回数が少ないプログラム

コア, 非コアに対する変更回数によりソフトウェア変更の健全性を次の 4 つに分類する(表 1). さらに変更の時間的変化から大域安定性を評価できる(図 2).

表 1 ソフトウェアの変更の健全性分類

記述項目	名前
(A) コアかつ変更回数が少ないプログラム群	安定
(B) コアかつ変更回数が多いプログラム群	活性
(C) 非コアかつ変更回数が少ないプログラム群	不活性
(D) 非コアかつ変更回数が多いプログラム群	不安定

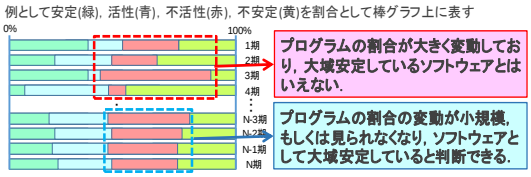

図 2 大域安定性評価

4.1.2 ソフトウェア価値の健全性

ソフトウェア価値の健全性の分類を表 2 に示す. 活性の割合が不安定の割合よりも高いことはソフトウェアの価値が増大し, 資産化していることを意味する. 一方活性の割合が低いことは価値が低く, かつコストが増大していることから負債化していることを意味する.

表 2 ソフトウェアの価値の健全性分類

記述項目	名前
活性の割合＞不安定の割合	資産化
活性の割合＜不安定の割合	負債化

5. プロトタイプ実装

図 3 に実装したプロトタイプの構成を示す.

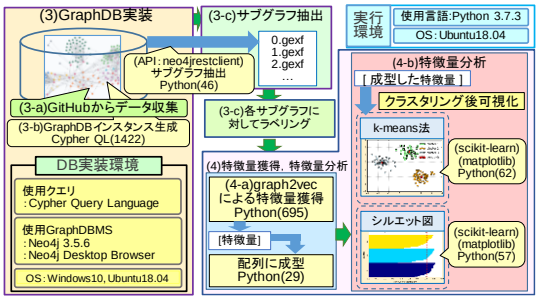

図 3　プロトタイプの構成

6. GitHub の機械学習フレームワークに適用

提案方法を GitHub の機械学習フレームワークである DGL[2]と PyTorch Geometric[3]に適用した. 適用期間を次の 5 つの期間に分けて分析した. (i)7/6~8/5, (ii)8/6~9/5, (iii)9/6~10/5, (iv)10/6~11/5, (v)11/6~12/5

本研究では, 3 回以上呼び出されているサンプルをコアとし, 3 回未満であるサンプルを非コアと定義する.

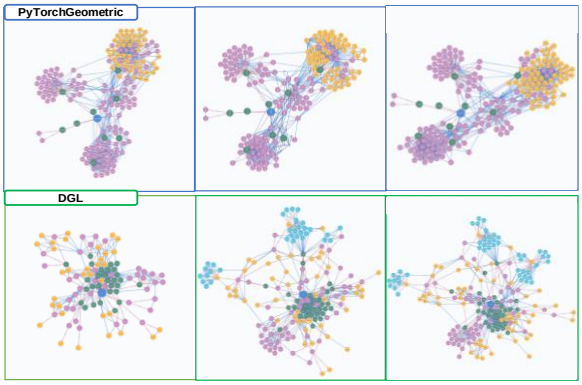

図 4　(i), (ii), (iii)期間目のインスタンス

コールグラフモデルに対して, 表現学習である graph2vec を適用してサブグラフ毎の特徴量を 1024 次元のベクトルとして特定した. この特徴量に対して k-means 法及び主成分分析による次元削減を行った結果のクラスタリングとシルエット図を図 5 に示す.

図 5 から, (ii)期間目の PyTorchGeometric と DGL は適切なクラスタに属しているサンプル数が多いことから, クラスタリングは妥当であるといえる. さらにシルエット図からクラスタ数 2 が妥当であるといえる.

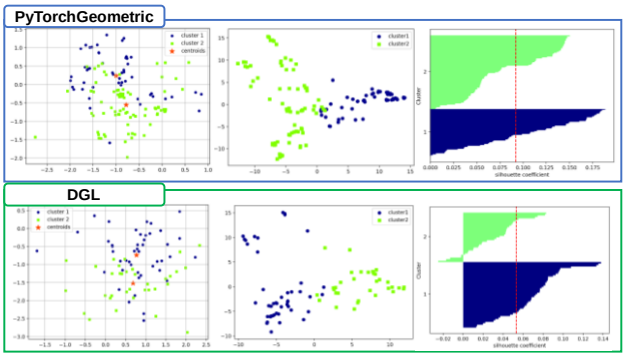

図 5　(ii)期間目のクラスタリング結果

7. 評価結果

価値(図 6), 大域安定性(図 7)より健全性を評価する.

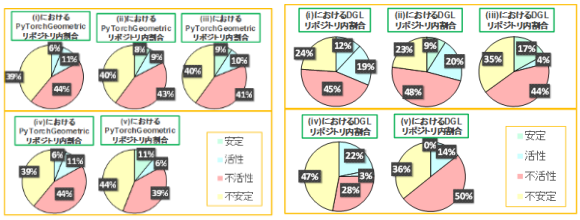

図 6　価値評価結果(左 PyTorchGeometric, 右 DGL)

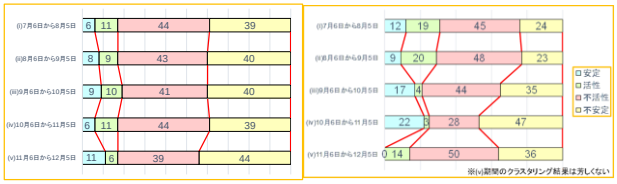

図 7　大域安定性評価結果(左 PyTorchGeometric, 右 DGL)

8. 考察

提案した分析方法により, 次の 3 点が明らかになった.

(1) PyTorchGeometric は全期間において, 大域安定していた. 一方 DGL は(ii)から(iv)の期間において大域安定していなかった. この原因として GitHub 上のコミットログから, DGL は PyTorchGeometric より, コミットされたコード行数の増加が大きいことが分かった. 従って, その期間中に加えられた変更及び機能追加がソフトウェア全体に対して大きな影響があったと推測できる.

(2) (ii), (iii)期間において変更が集中しているプログラムが大域安定を阻害していると推定できる.

(3)先行研究[4]と比較すると, 本研究はソフトウェア内の呼出し関係から価値や大域安定性の変化を明らかにしたことで, 健全性の直接的な分析を可能にした.

9. まとめ

本研究ではプログラムの呼出し関係をコールグラフモデルで定義し, そのグラフに表現学習を適用することでソフトウェアの健全性の分析を可能とした. この分析により, 開発の良さの時間的推移が明らかとなるため, ソフトウェア構造の変化に対する理解支援や, 開発の健全性を高めることが期待できる.

今後適用対象を拡大し提案方法の有効性を確認する.

10. 参考文献

[1] 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, 情報処理学会研究報告, Vol. 2017-ICS-186 No. 6, Mar. 2017, pp. 1-6.

[2] Distributed (Deep) Machine Learning Community, DGL (Python Package Built to Ease Deep Learning on Graph, on Top of Existing DL Frameworks), https://github.com/dmlc/dgl/.

[3] M. Fey, PyTorch Geometric, https://github.com/rusty1s/pytorch_geometric/.

[4] 加藤 聖也, 他, 機械学習によるグラフモデル OSS 開発コミュニティ構造の特徴量分析方法の提案と評価, 情報処理学会第 81 回全国大会講演論文集, No. 6N-03, Mar. 2019, pp. 281-282.

[5] Narayanan, et al., graph2vec: Learning Distributed Representations of Graphs, Proc. of MLG 2017, https://arxiv.org/abs/1707.05005/.