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

2016SE023　可知 敬朗　2016SE049　牧野 槙一郎
指導教員　青山 幹雄

1 研究課題・背景

1.1 研究背景

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[6]．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．

1.2 研究課題

本研究では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

2.1 表現学習

機械学習の一つである表現学習とは，画像，音，自然言語等の要素を分散表現として抽象化する方法である．ネットワーク表現学習[1]とは，ネットワーク構造からノード，エッジ，サブグラフの分散表現を獲得する方法である．提案されるアルゴリズムとして，node2vec[8]，graph2vec[11]などがあげられる．これらの方法で計算された分散表現を使うことで，複雑なネットワーク構造に対する既存のグラフ分析方法よりも，ラベル推定や分類タスクを高い精度で実行できる．また，ネットワーク表現学習方法は大規模なグラフの可視化にも応用されている．

2.2 グラフデータベース(GraphDB)

GraphDB は，CRUD（Create, Read, Update, Delete）メソッドを持つデータベース管理システムである．Neo4j[12]は Java ベースで実行されるオープンソースの GraphDBMS である．Neo4j で提供される NoSQL である Cypher クエリ言語を用いることでグラフ作成，削除，検索，分析が可能である．

2.3 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[9]は OSS 開発コミュニティを SCGM(Software Community Graph Model)として定義し，機械学習を用いることで開発者の活動に関する特徴量を獲得した．特徴量に対しクラスタリングを行うことでグラフから得た特性を可視化した．

この分析により OSS 開発コミュニティにおける進化構造が明らかにした．

3 アプローチ

GitHub[7]における 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な視点での分析が必要である．ソフトウェアの健全性の定義は後述する．

次に本研究のアプローチを示す．

(I) GraphDB 生成

GitHub におけるプログラムの呼出し関係を大局的な視点から表現するために，コールグラフモデルを定義し GraphDB を生成する．

(II) 特徴量獲得プロセス

コールグラフを分析することを目的とし，表現学習によって特徴量を獲得する．局所的な方法では，ノード数やエッジ数の増大に対して，適用には限界があり，最初に選択するノードによって分析結果が異なる可能性がある．そのため，本研究では，局所的な方法の node2vec ではなく大局的な graph2vec を用いる．また，プロパティを用いることにより，全体のグラフからサブグラフ抽出した場合においても，元のグラフの情報を保持することが可能となる．

(III) 特徴量分析プロセス

特徴量からクラスタリングを行い，分析を行う．(II)と同様に，局所的な方法では，分析結果が異なる可能性があるため，k-means 法を用いる．

(IV) ソフトウェアの健全性評価プロセス

プログラムが呼び出される回数に応じて「コア」，「非コア」の名称を付与する．クラスタとプログラムを照合することで，クラスタ毎の「コア」，「非コア」の割合を分析する．その後，分析結果を時系列で可視化することで，ソフトウェアの健全性を評価する．

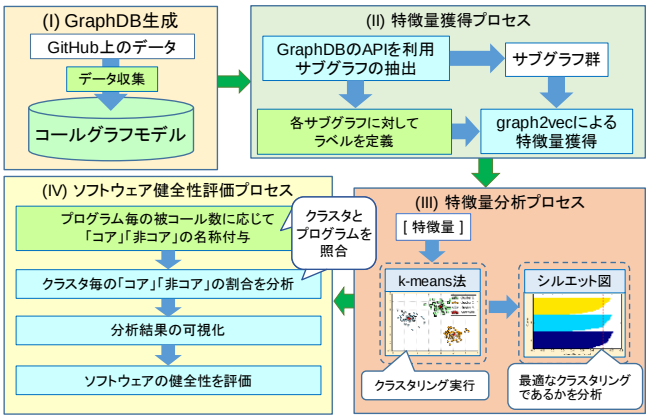

図 1　アプローチ

4 提案方法

4.1 提案プロセス

提案プロセスは次の 6 項目から成る(図 2).

(1)　コールグラフモデル定義

ノードとエッジ，それぞれのプロパティを定義する．

(2)　仮説分析内容設定

設定した仮説に基づいて，仮説ごとに明らかにするべき分析項目を設定する．

(3)　GraphDB 実装

GraphDB 実装は，(a) GitHub からのデータ収集，(b) GraphDB インスタンス生成，(c) サブグラフ毎に.gexf 形式に変換，及びラベル付けの 3 つのプロセスから成る．

(4)　特徴量獲得，特徴量分析

特徴量獲得，特徴量分析は，(a) graph2vec による特徴量獲得，(b) 特徴量分析の 2 つのプロセスから成る．

(5)　ソフトウェア健全性の評価

期間毎に GraphDB インスタンスを生成し，分析結果を時系列で可視化，評価する．

(6)　仮説検証

分析で得られた結果から，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．

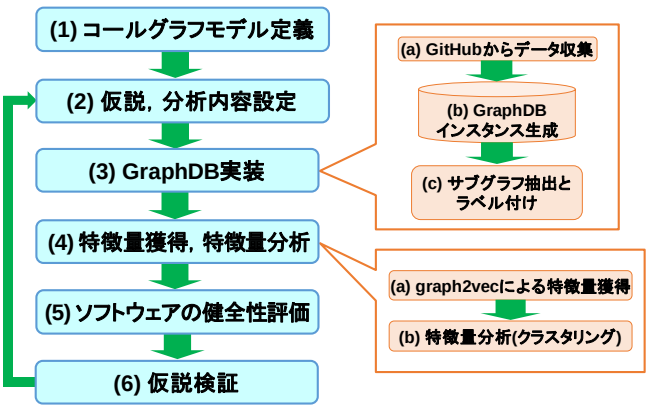

図 2　提案プロセス

4.2 コールグラフモデル定義

ソフトウェアの健全性を分析するために，プログラムの呼出し関係全体をコールグラフモデルとして定義する．

(1) ノード定義

分析対象は，Python のプログラム及び，Python のプログラムが含まれるディレクトリとする．

表 1 ノードのプロパティ定義

記述項目	属性
ディレクトリ名 or ファイル名	name
ディレクトリ形式 or ファイル形式	def
分析対象ディレクトリからのパス	path

(2) エッジ定義

表 2 エッジ定義

記述項目	型
あるディレクトリ以下に他のディレクトリ，プログラムが存在する	CONTAIN
あるプログラムがプログラムを import している	CALL

4.3 ソフトウェアの健全性評価

4.3.1. 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 3)．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 3)．

表 3 ソフトウェア変更の健全性分類

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

表 3 に分類した理由を次に示す．

(A) 安定：

コアプログラムは多く呼び出され，変更点が表面化しやすい状況にあるが，(A)に属するプログラム群は変更回数が少ないことから，変更する必要がある箇所が少ないと判断でき，「安定」しているといえる．

(B) 活性：

(B)に属するプログラム群は呼び出される回数が多く，変更点も多いことから，利用される回数が多いと判断でき，「活性」であるといえる．

(C) 不活性：

(C)に属するプログラム群は呼び出される回数が少なく，変更点も少ないことから，利用される回数も少ないと判断でき，「不活性」であるといえる．

(D) 不安定：

(D)に属するプログラム群は呼び出される回数が少ないため，変更点は表面化し辛いが，(D)に属するプログラム群は変更回数が多いことから，潜在的に変更箇所の多いプログラムが多く存在し，「不安定」であるといえる．

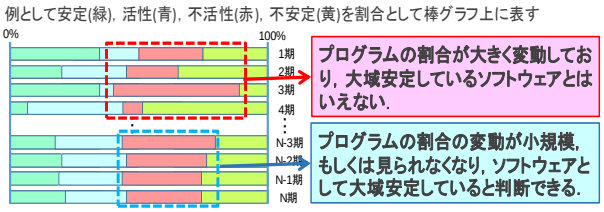

図 3　大域安定性評価

4.3.2. 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 4　ソフトウェア価値の健全性分類

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 4 に実装したプロトタイプの構成を示す.

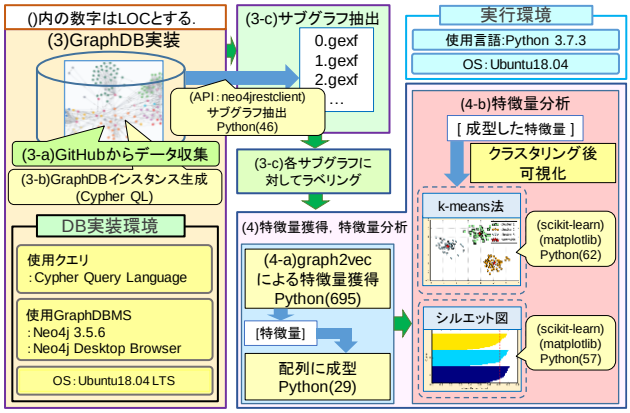

図 4　プロトタイプの構成

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[4]と PyTorch Geometric[5]に適用した．8/5，9/5，10/5，11/5，12/5 時点の GraphDG インスタンスを図 5，図 6 に示す．また表 5，表 6 にノード数，エッジ数を示す．

本研究では，3 回以上呼び出されているサンプルをコア，3 回未満であるサンプルを非コアとして定義し，分析期間を次の 5 つの期間とする．(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.

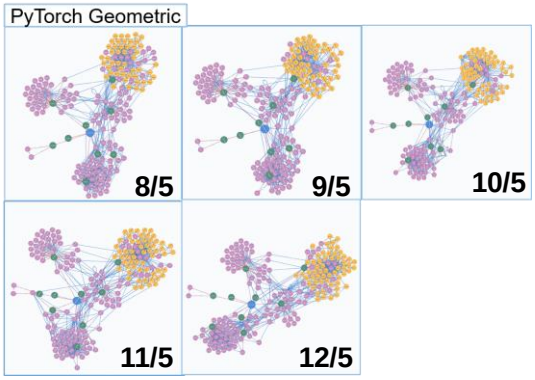

図 5　PyTorch Geometric DB インスタンス

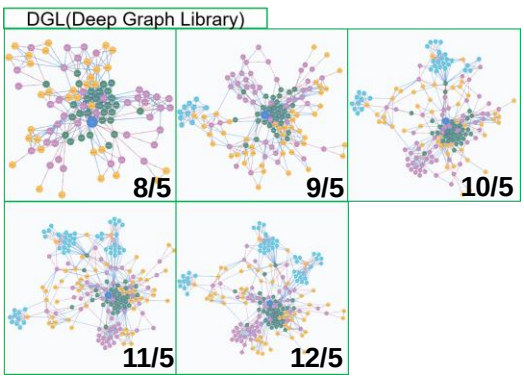

図 6　DGL DB インスタンス

表 5　各期間におけるインスタンス(PyTorchGeometric)

構成要素	8/5	9/5	10/5	11/5	12/5
ノード数	183	186	186	192	195
エッジ数(CALL)	485	501	501	511	523
エッジ数(CONTAIN)	182	185	185	191	194

表 6　各期間におけるインスタンス(DGL)

構成要素	8/5	9/5	10/5	11/5	12/5
ノード数	103	134	169	185	186
エッジ数(CALL)	220	299	365	406	412
エッジ数(CONTAIN)	103	134	167	183	184

各グラフインスタンスに対して，表現学習である graph2vec を適用しサブグラフ毎の特徴量を 1,024 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 法のクラスタ分析を行った(図 7 左)．この方法ではクラスタに分離できなかった．原因として次元数が大きいことが挙げられる．そのためベクトルの成分を合成し主成分を抽出することで次元数を削減する主成分分析を適用した後に k-means 法のクラスタ分析を行った(図 7 中央)．また，クラスタ数の妥当性を評価するためにシルエット分析を行った(図 7 右).

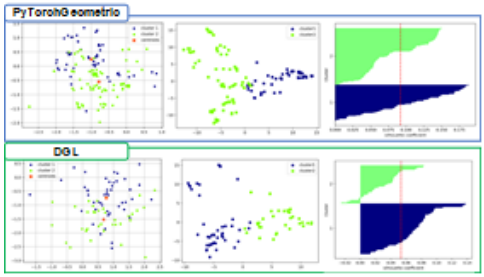

図 7　(ii)期間目のクラスタリング結果

図 7 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.

7 評価結果

価値(図 8)と大域安定性(図 9)より健全性を評価する.

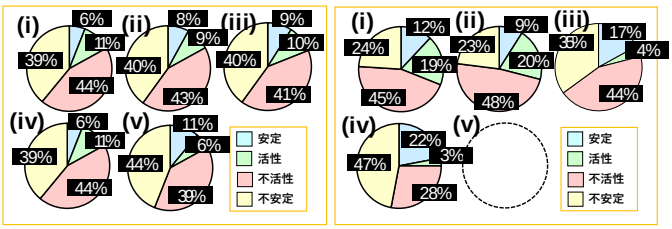

図 8　価値評価結果(左 PyTorchGeometric，右 DGL)

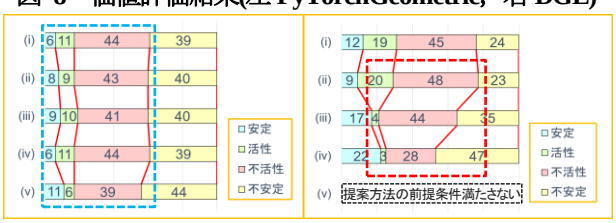

図 9　大域安定性評価結果(左 PyTorchGeometric，右 DGL)

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．

ただし 11/6～12/5 は DGL の変更が極めて少なかったため，分析の対象外とした．

(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，DGL の当該期間におけるコミットされたコード行数の増加が，PyTorch Geometric よりも大きいことが挙げられる．従って，当該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を及ぼしたと推測できる(図 10，図 11)．

(2) (ii)，(iii)期間において変更が集中しているファイルが大域安定を阻害していると推定できる．表 7 にその機能と共に示す．

(3) 先行研究[9]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．

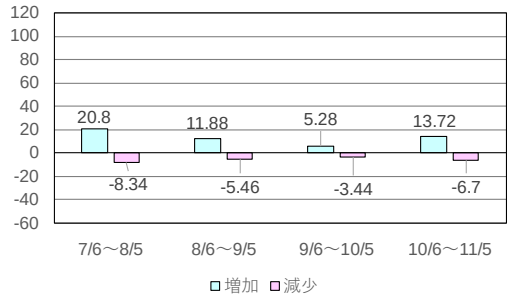

図 10 ファイル単位のコード行数増減(PyTorchGeometric)

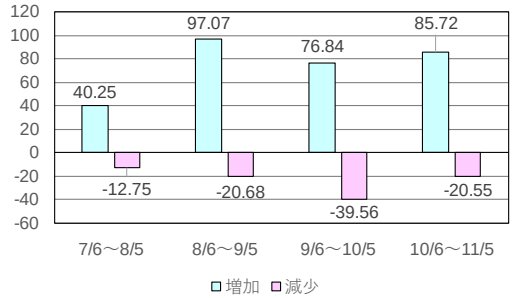

図 11 ファイル単位でのコード行数増減(DGL)

表 7 大域安定を阻害しているファイル

パス/ファイル名	機能
python/dgl/contrib/sampling/sampler.py	近傍ノード探索，GNN
python/dgl/convert.py	グラフ生成，演算処理
python/dgl/backend/pytorch/tensor.py	多次元配列形成，勾配の整形
python/dgl/contrib/dis_kvstore.py	サーバ構築，pull，push 操作
python/dgl/data/chem/tox21.py	重み付与，計算

9 期待効果

提案方法を用いることでソフトウェア開発において次の効果が期待できる．

(1) ソフトウェア構造の特性を獲得

プログラム群に対する変更や，それに伴う影響に着目した分析をすることで，ソフトウェア全体の特性が獲得可能である．

(2) ソフトウェアの健全性の評価

ソフトウェアが資産化，負債化している期間が観測できるため，特定の期間における変更が対象のソフトウェアにとって有効であったか否かが判断できる．また，ソフトウェアの大域的な変動により，大規模な変更が行われたか否かが特定できるため，対象のソフトウェアの健全性が損なわれたかどうかを判断できる．

10 今後の課題

今後の課題は次の 2 点である．

(1) クラスタリング精度の改善

本研究において k-means 法やシルエット分析を適用した機械学習ライブラリによっては分析結果が仮説を満たさなかった．そのため，graph2vec による特徴量獲得やクラスタリング方法を見直す必要がある．

(2) 提案方法を実現するアーキテクチャの改善

本研究の提案方法を実現するグラフ生成では，扱うデータが複雑になった時にスケールすることが困難である．したがって，グラフ生成を中心にアーキテクチャを見直す必要がある．

11 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．

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] J. P. Bagrow, and E. M. Bolt, A Local Method for Detecting Communities, Phys. Rev. E, Vol. 72, No. 4, Oct. 2005, pp. 1-16, https://arxiv.org/abs/cond-mat/0412482/.

[3] A. Clauset, Finding Local Community Structure in Networks, Phys. E, 2005, Vol. 72, No. 2, Aug. 2005, pp. 1-7, https://arxiv.org/abs/physics/0503036/.

[4] 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/.

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

[6] M. Girvan, and M. E. J. Newman, Community Structure in Social and Biological Networks, Proc. of the National Academy of Science of U.S.A., Vol. 99, No. 12, Jun. 2002, pp. 7821-7826.

[7] GitHub, GitHub, https://github.com/.

[8] A. Grover, et al., node2vec: Scalable Feature Learning for Networks, Proc. of KDD 2016, ACM, Aug. 2016, pp. 855-864.

[9] S. Kato, et al., A Structural Analysis Method of OSS Development Community Evolution Based on A Semantic Graph Model, Proc. of COMPSAC 2018, IEEE, Jul. 2018, pp. 292-297.

[10] 松島 裕, 他, ネットワーク構造の推移性に着目した局所的クラスタリング方法の提案, 第 74 回全国大会講演論文集, 情報処理学会, Mar. 2012, pp. 469-470.

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

[12] Neo Technology, Neo4j, https://neo4j.com/.