Vol.2020-SE-204 No.9
2020/3/2

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†

概要：OSS の開発は不特定多数のユーザが参画できるため，その開発の実態が不透明であるという問題がある．この問題に対し OSS 内のプログラム呼出し関係をコールグラフモデルとして定義し，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価値，及び大域安定性の分析を可能とした．価値と大域安定性に基づきソフトウェアの健全性を分析可能とした．提案した分析方法を支援するプロトタイプを，グラフデータベースである Neo4j と表現学習ライブラリである graph2vec を用いて実装した．プロトタイプを GitHub 上の PyTorch Geometric と DGL の開発データに適用し，健全性の時間的推移を明らかにした．この結果から，提案方法の有効性，妥当性を示す．

キーワード：コールグラフ，グラフモデル，表現学習，OSS，健全性，大域安定性

A Health Analysis Method of Software Systems Using Machine Leaning on Call Graph Models and its Evaluation

TAKAAKI KACHI† SHINICHIRO MAKINO† MIKIO AOYAMA†

1. 研究の背景と課題

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体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，そのネットワーク構造の時間的推移に着目してソフトウェアの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を評価することで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

本稿では研究背景を踏まえ次の 3 点を研究課題とする．

RQ1: 呼出し関係を表すコールグラフモデルはどのように定義できるか?

RQ2: コールグラフの機械学習によるソフトウェア健全性はどのように分析できるか?

RQ3: 実データに対して健全性分析は有効，妥当か?

2. 関連研究

2.1 ネットワーク表現学習

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．提案されているアルゴリズムとして，node2vec[8]，graph2vec[12]などがある．これらの方法で計算された分散表現を用いて複雑なネットワーク構造に対して，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は大規模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 はグラフ構造を管理し，CRUD（Create, Read, Update, Delete）メソッドを持つ DBMS ある．Neo4j[13]はオープンソースの GraphDBMS である．Neo4j で提供される Cypher クエリを用いてグラフ作成，削除，検索，分析が可能である．

2.3 ソフトウェアエコシステムの健全性分析

先行研究[11]ではソフトウェアプロジェクト，及びそれに依存するコミュニティをソフトウェアエコシステムの健全性を表す概念モデルとして定義し，ソフトウェアに対する技術や環境の変化が健全性に対しどのような影響を与えるかを分析した．またこの分析から，ソフトウェアエコシステムの健全性を評価するための指標や変更に対する予測モデルをガイドラインとして提案した．

2.4 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 開発コミュニティにおける開発者，及びその活動を SCGM(Software Community Graph Model)として定義し，機械学習を用いて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，OSS 開発コミュニティにおける開発者の成長パターン，及びコミュニティの進化構造を明らかにした．

† 南山大学理工学部ソフトウェア工学科
Dep, of Software Engineering, Nanzan University

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(図 1).

(I) GraphDB 生成

GitHub における OSS 内の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．局所的な方法ではグラフのスケーラビリティや初期値依存の点から適用には限界があるため，本研究では，局所的な分析アルゴリズムの node2vec ではなく大局的分析アルゴリズムである graph2vec を用いる．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，元のグラフの情報の保存が可能となる．

(III) 特徴量分析プロセス

特徴量をクラスタリングし，その構造を分析する．(II)と同様に，大局的な分析方法である k-means 法を用いる．ここで，クラスタリングの目的は変更回数の多さによりプログラム群を 2 クラスタに分離することである．

(IV) ソフトウェアの健全性評価プロセス

呼び出される回数に応じてプログラムを「コア」，「非コア」に分類する．また，(III)におけ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．

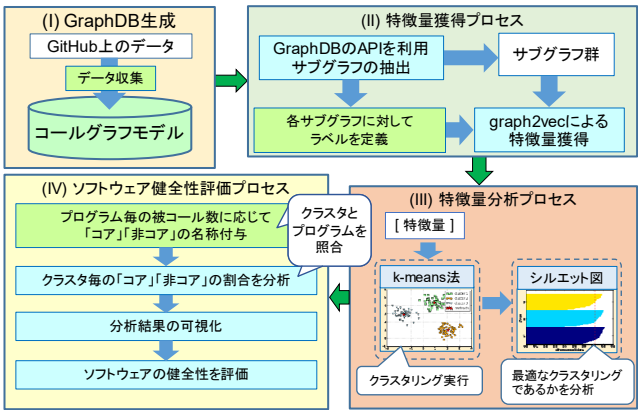


図 1 アプローチ

Figure 1 Approach

4. 提案方法

4.1 提案プロセス

提案プロセスは次の 6 つのプロセスから成る(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 つのサブプロセスから成る．

(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 つのサブプロセスから成る．

(5) ソフトウェアの健全性評価

一定期間毎に 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．

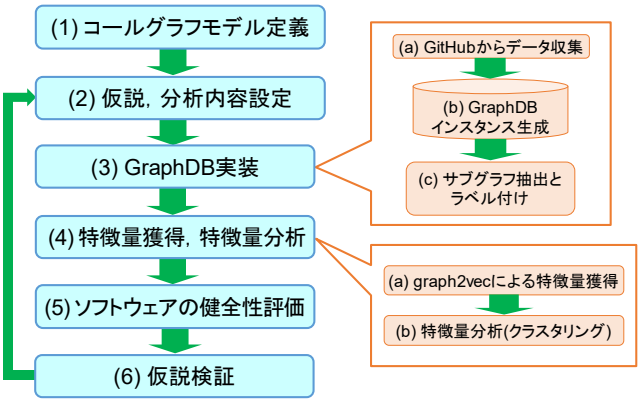


図 2 提案プロセス

Figure 2 Proposed Process

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．次の(1)，(2)にグラフモデルにおけるノード，エッジの定義を示す．ノードのプロパティ定義を表 1，エッジの型定義を表 2 にそれぞれ示す．

(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

Table 1 Definition of Node's Properties

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　エッジの型定義
Table 2　Definition of Edge's Types

記述項目	型
あるディレクトリ下に他のディレクトリ，プログラムが存在する	CONTAIN
あるプログラムが他のプログラムを 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はソフトウェア内で重要な機能を持つと考えられ，変更時の影響も大きいと考えられる．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する(図 3)．

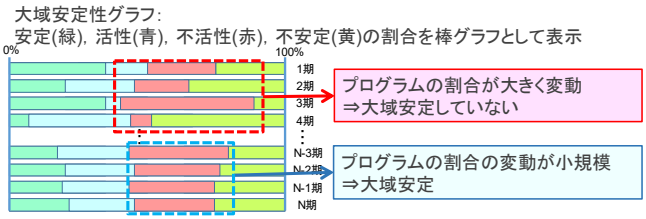


図 3　大域安定性評価
Figure 3　Evaluation of Global Stability

表 3 ソフトウェア変更の健全性分類
Table 3　Health Classification of Software Modification

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

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．

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
Table 4　Health Classification of Software Values

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

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．

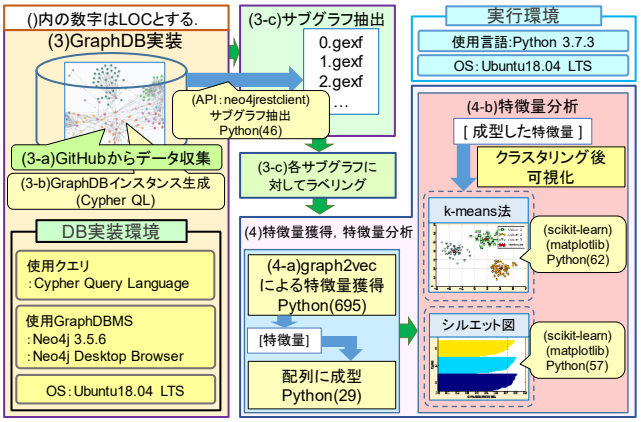


図 4　プロトタイプの構成
Figure 4　Structure of Prototype

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．

6.1　GraphDB インスタンス生成

次の 5 つの期間に分けて各インスタンスを生成した．2019 年の 8/5，9/5，10/5，11/5，12/5 時点における GraphDB 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．図 5，図 6 においてグラフのノードを root ノードからの深さで色分けした．root を青，深さ 1 を緑，深さ 2 を紫，深さ 3 を黄，深さ 4 を水色で表現している．また本研究のコア，非コアは被 CALL 数が 3 回以上か未満かで分類した．

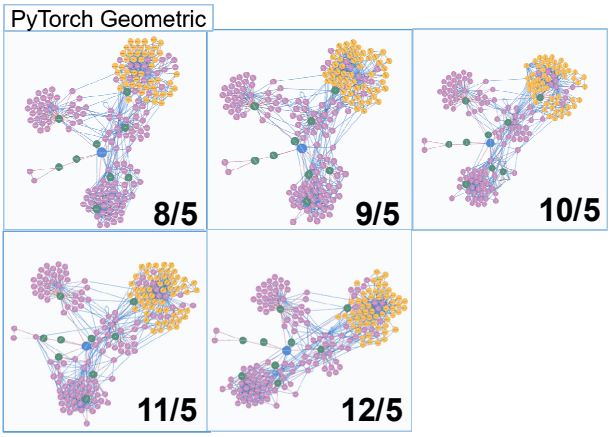


図 5　PyTorch Geometric DB インスタンス
Figure 5　PyTorch Geometric DB Instance

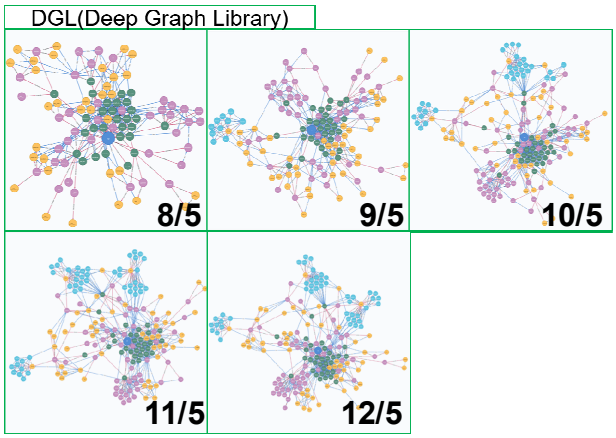


図 6　DGL DB インスタンス
Figure 6　DGL DB Instance

表 5　各期間におけるインスタンス(PyTorchGeometric)
Table 5　Instance Details of Each Period(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)
Table 6　Instance Details of Each Period(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

6.2 特徴量分析

各グラフインスタンスに対して，(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 におけるソフトウェアへの変更回数をラベルと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 法を用いてクラスタリングを行った．しかし，この方法ではクラスタを分離することができなかった．原因として，抽出したベクトルの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 法を適用した．また，クラスタ数の妥当性を確認するためにシルエット分析を行った．

6.2.1 主成分分析適用後に k-means 法によるクラスタリング

主成分分析(PCA:Principal Component Analysis)を用いて抽出したベクトルに対し，k-means 法を適用した結果について次の図 7，及び図 8 に示す．

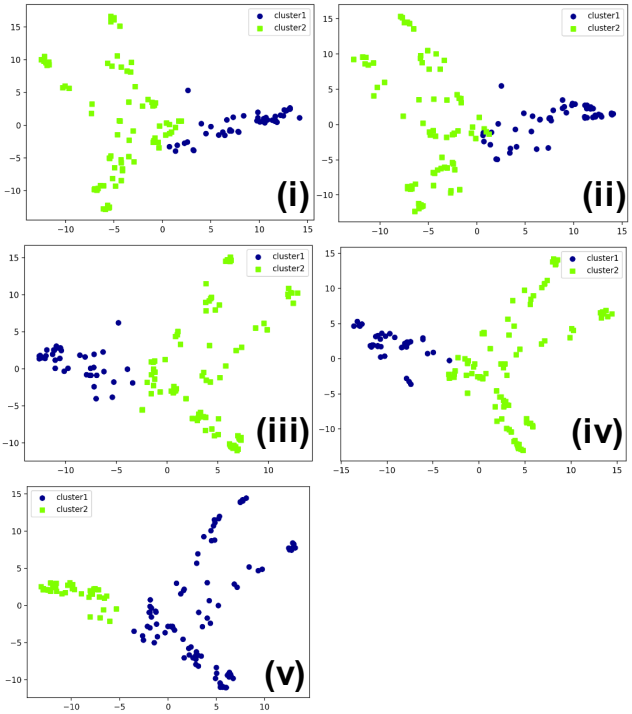


図 7　主成分分析後の k-means 法適用結果(PyTorch Geometric)
Figure 7　PCA and K-means Analysis (PyTorch Geometric)

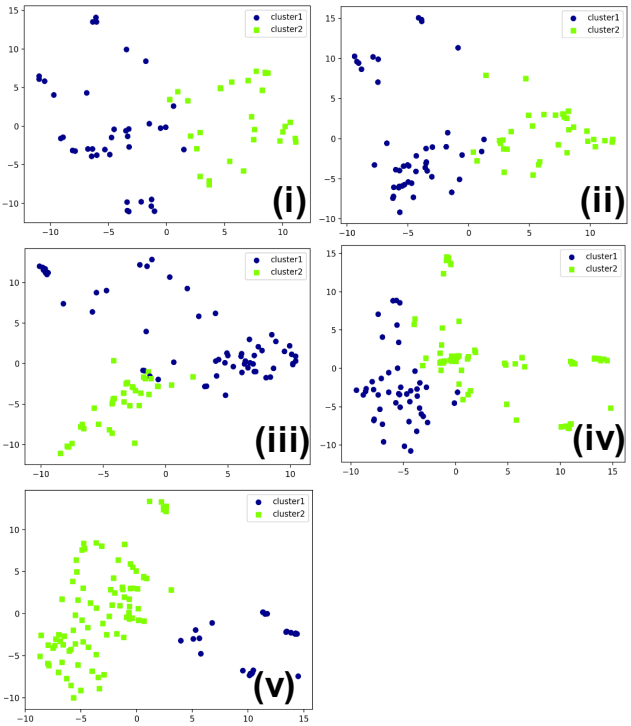


図 8　主成分分析後の k-means 法適用結果(DGL)
Figure 8　PCA and K-means Analysis (DGL)

図 7，図 8 の結果より，PyTorch Geometric 及び DGL の各グラフインスタンスの特徴量分析において，2 クラスタを明確に分離できた．

また学習データのラベルに基づき，クラスタ 1 には変更回数が少ないプログラム，クラスタ 2 には変更回数が多いプログラムが属していると推測できる．

6.2.2 シルエット分析によるクラスタリング

6.2.1 で行った主成分分析後の k-means 法適用結果に対し，クラスタリングの妥当性，及びクラスタ数の妥当性を確認するため，抽出したベクトルに対しシルエット分析を適用した(図 9，図 10)．

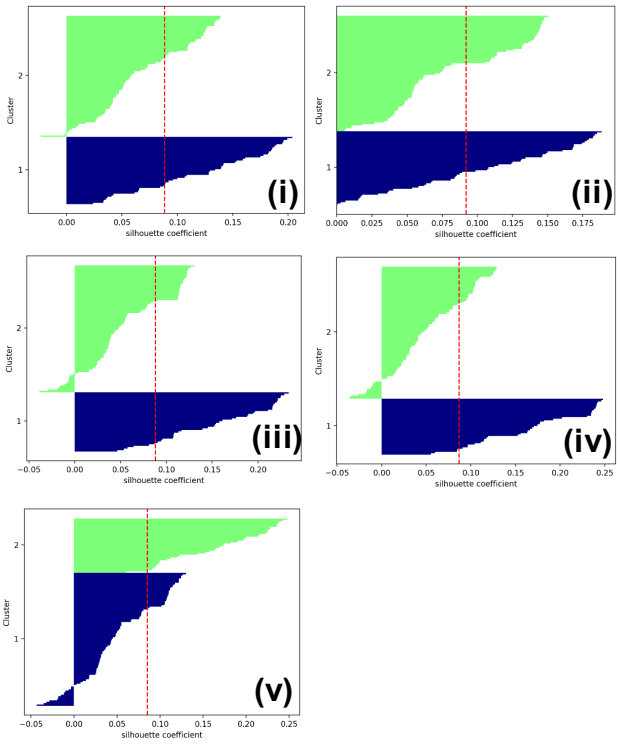


図 9　シルエット分析の結果 (PyTorch Geometric)

Figure 9　Silhouette Analysis (PyTorch Geometric)

(1) PyTorch Geometric のシルエット分析結果考察

図 9 より PyTorch Geometric のシルエット分析について，各期間とも適切なクラスタに属するサンプル数が多く，間違ったクラスタに属するサンプル数が少な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，各シルエット図よりクラスタ数 2 が妥当であるといえる．

また，(i)～(iv)期間においてはクラスタ 1 に属するサンプルが多いのに対し，(v)期間目ではクラスタ 2 に属するサンプル数が多くなり，間違ったクラスタに属するサンプルも増加していることが分かった．この原因として，(v)期間目におけるソフトウェアへの変更回数が少ないことによる学習のためのラベリングの不足が挙げられる．

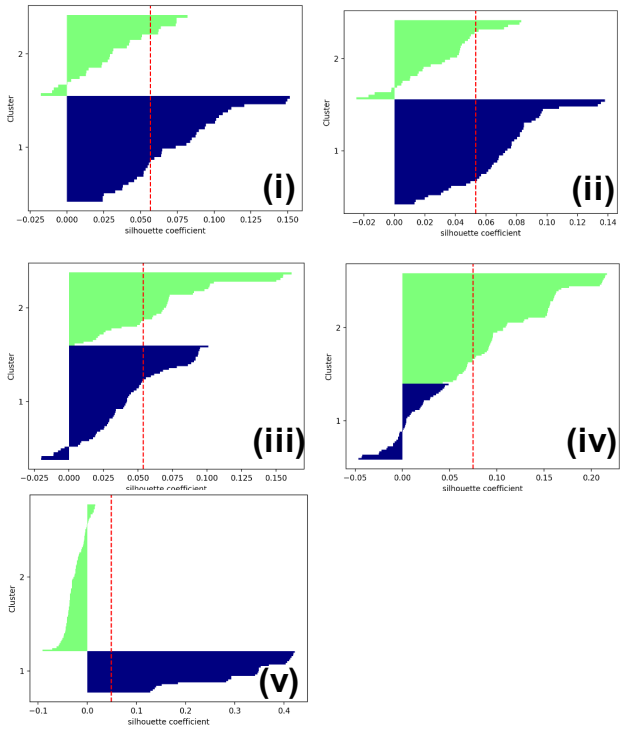


図 10　シルエット分析の結果 (DGL)

Figure 10　Silhouette Analysis (DGL)

(2) DGL のシルエット分析結果考察

図 10 より DGL のシルエット分析について，(i)～(iii)期間において適切なクラスタに属するサンプル数が多く，間違ったクラスタに属するサンプル数が少ないことから，(i)～(iii)期間のクラスタリングは妥当であり，クラスタ数 2 も妥当であるといえる．(iv)，(v)期間目においては，間違ったクラスタに属するサンプル数が増加していることが分かる．また，(v)期間目においてクラスタ 2 に属するサンプルのほぼ全てが間違っているという結果になり，学習不足と判断できる．これにより(v)期間目における分析は，提案方法におけるプログラムを 2 クラスタに分離するという前提条件を満たさなかった．これらの原因として(iv)，(v)期間目においてはソフトウェアの変更回数がきわめて少なく，学習のためのラベリングが適切ではなかった可能性が挙げられる．

7. ソフトウェアの健全性評価

各グラフインスタンスから得られるノードの被 CALL 数，及び特徴量分析(6.2)の結果を元に，ソフトウェア変更による健全性分類(表 3)を行い，ソフトウェアの価値(7.1)，ソフトウェアの大域安定性(7.2)を評価する．また，これら 2 つの評価を以てソフトウェアの健全性を評価する．

7.1 ソフトウェアの価値評価

6.2 の分析期間(i)～(v)におけるソフトウェアの価値について評価指標を次の式(1)の V として定義する．

$$V = (\text{活性の割合}) - (\text{不安定の割合}) \quad (1)$$

図 11 における縦軸を V の値，横軸を期間としてソフトウェア価値の推移を示す．表 4 の健全性分類より，V の値が正であれば分析期間においてソフトウェアが資産化，負であれば負債化していると判断できる．

また, DGL の(v)期間目は学習が失敗し提案方法の前提条件を満たさないと判断できたため評価の対象としなかった．

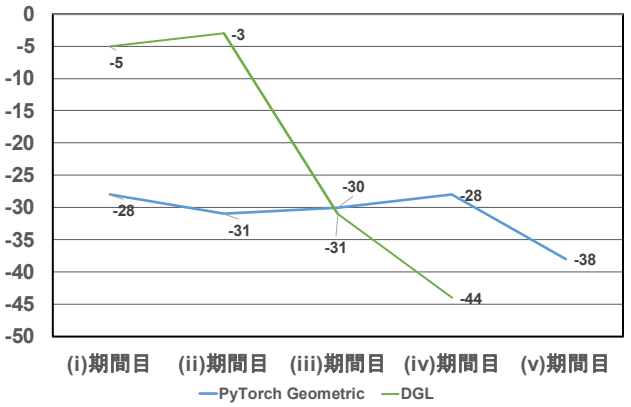


図 11　ソフトウェア価値評価結果

Figure 11　Evaluation of Software Value

ソフトウェアの価値について，PyTorch Geometric，及び DGL 全期間において(DGL の(v)期間目は除く)活性のプログラム群より不安定のプログラム群の割合が大きくなったので「負債化」していると判断できる．しかし PyTorch Geometric は活性の割合と不安定の割合の変動が小さいのに対し，DGL においては(ii)〜(iii)期間目，(iii)〜(iv)期間目に見られるように割合の変動が大きいことから負債化が急速に進んでいると推測できる．

7.2 ソフトウェアの大域安定性評価

ソフトウェアの変更時における健全性分類の結果(表 3)を時系列に沿って観測することで，多期間にわたるソフトウェアの全体の大域安定性を評価し，その結果を次の図 13, 図 14 に示す．また，7.1 のソフトウェアの価値評価と同様に DGL の(v)期間目は評価の対象としなかった．

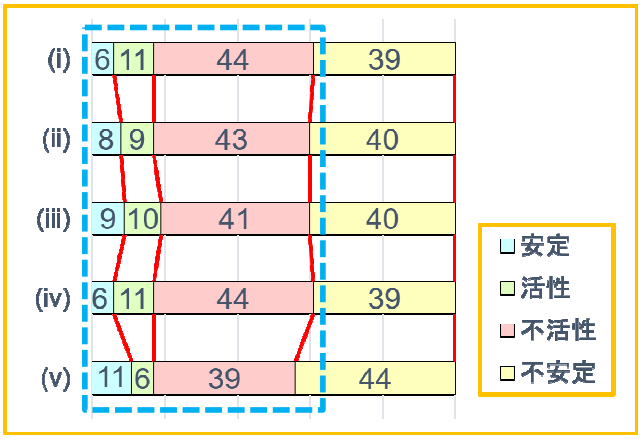


図 12　大域安定性評価 (PyTorch Geometric)

Figure 12　Evaluation of Global Stability (PyTorch Geometric)

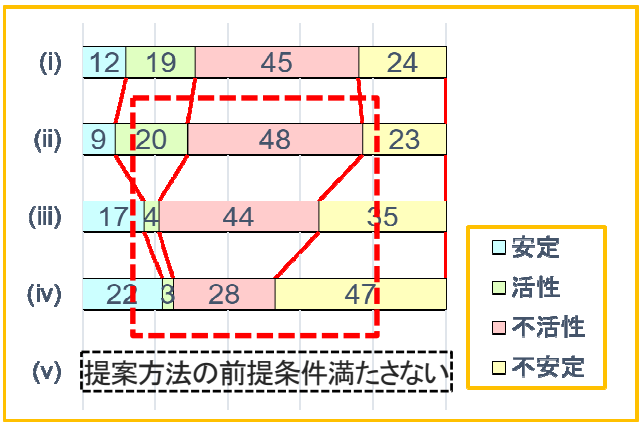


図 13　大域安定性評価 (DGL)

Figure 13　Evaluation of Global Stability (DGL)

図 13 より PyTorch Geometric は全期間においてソフトウェアが大域安定していることが分かる．これに対して，図 14 より DGL は(ii)期間目から大域安定性が損なわれていることが分かった．よって(ii)〜(iii)期間目に行われたソフトウェアへの変更が DGL の大域安定性を阻害したと推測できる．

8. 考察

以上の分析，評価により，次の 3 点が明らかとなった．

(1) PyTorch Geometric は全期間において大域安定している．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 15，図 16，図 17)．

(2) DGL の(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)先行研究と比較すると，本研究はソフトウェア内の呼出し関係をコールグラフモデルとして捉え，ソフトウェアの変更における価値や大域安定性の時間的変化を明らかにしたことで，ソフトウェア全体の健全性の直接的な分析を可能にした．

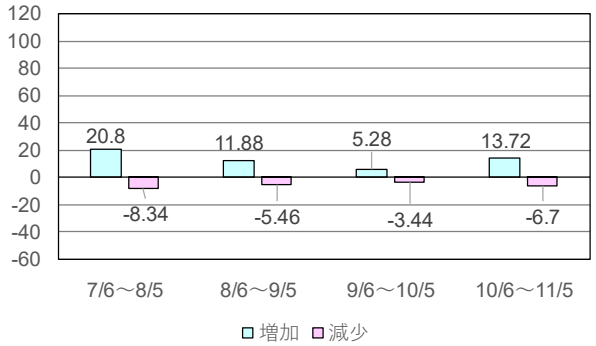


図 14　ファイル単位のコード行数増減 (PyTorch Geometric)

Figure 14　Change of Codes by Files (PyTorch Geometric)

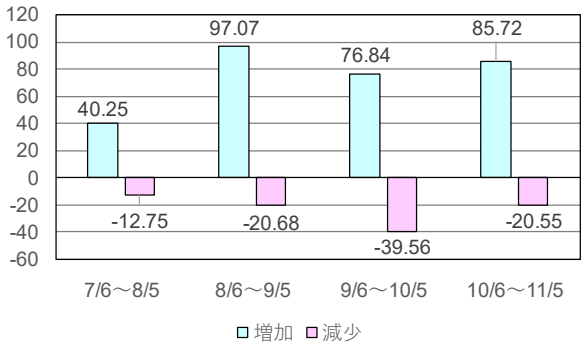


図 15　ファイル単位のコード行数増減 (DGL)
Figure 15　Change of Codes by Files (DGL)

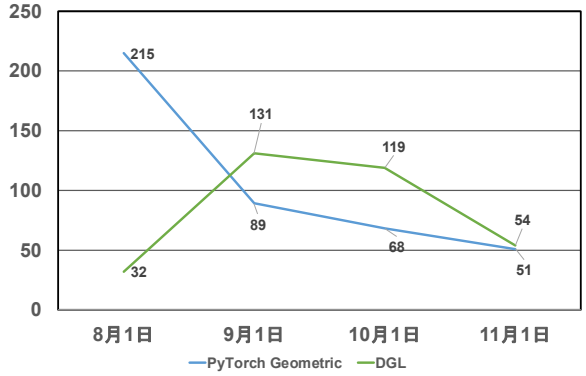


図 16　各期間におけるコミットされたファイル数
Figure 16　Number of Files Commited by Each Period

表 7　大域安定を阻害しているファイル
Table 7　Files Obstructing Global Stability

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大域的な特性を獲得

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可能となる．

価値の観点，及び大域安定性の観点から得られた評価を総合的に判断することにより，ソフトウェアの健全性を評価することが可能となる．

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，扱うデータの増大と複雑化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．

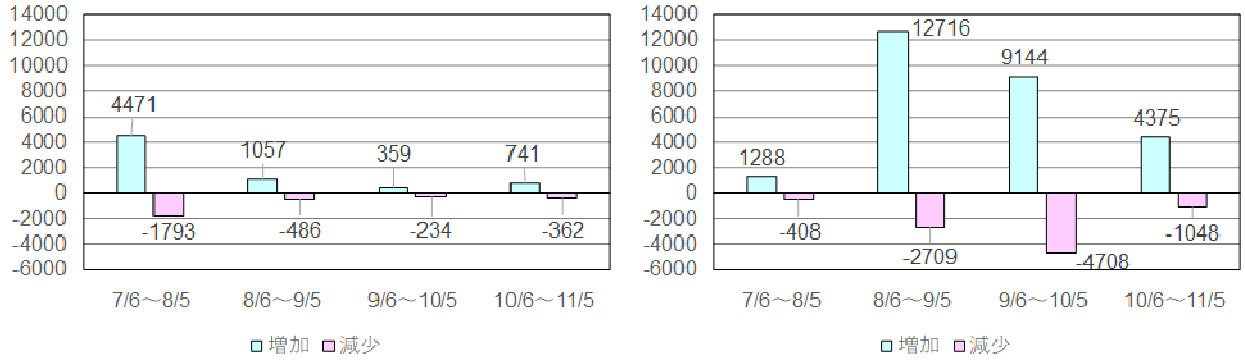
11. まとめ

本研究では OSS 内の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，グラフ表現学習を適用することで対象のソフトウェアの特徴量を獲得し，ソフトウェア変更の価値，及び大域安定性を評価する方法を提案した．これにより，対象としたソフトウェアの健全性の分析を可能とした．プロトタイプを実装し，実データに適用することで提案方法の有効性，妥当性を評価した．提案方法を用いることで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大域的な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

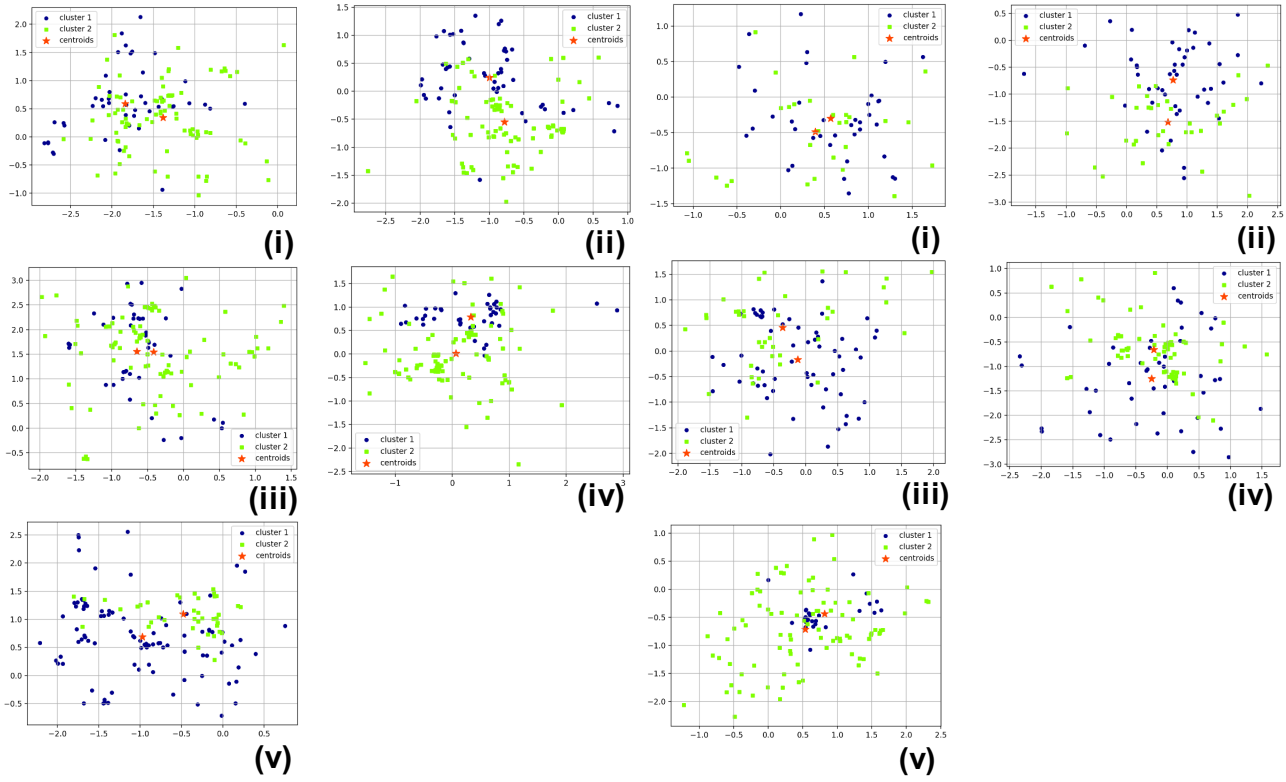
[1] 浅谷 公威，他，ネットワーク表現学習によるネットワークの成長可視化，情報処理学会研究報告, Vol. 2017-ICS-186 No. 6, Mar. 2017, pp. 1-6.
[2] J. P. Bagrow, and E. M. Bolt, A Local Method for Detecting Communities, Phys. Rev. E, Vol. 72, No. 4, Oct. 2005, pp. 1-16, https://arxiv.org/abs/cond-mat/0412482/.
[3] A. Clauset, Finding Local Community 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] 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.
[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] T. Mens, et al, Towards an Interdisciplinary, Socio-Technical Analysis of Software Ecosystem Health, Proc. of BENEVOL 2017, Dec. 2017, pp. 7-9, http://ceur-ws.org/Vol-2047/.
[12] A. Narayanan, et al., graph2vec: Learning Distributed Representations of Graphs, Proc. of MLG 2017, Jul. 2017, https://arxiv.org/abs/1707.05005/.
[13] Neo Technology, Neo4j, https://neo4j.com/.

付録



A. 1　各期間におけるコード行数増減 (PyTorch Geometric，DGL)

A. 1　Change of Codes (PyTorch Geometric，DGL)



A. 2　k-means 法適用結果 (PyTorch Geometric，DGL)

A. 2　K-means Analysis (PyTorch Geometric，DGL)