

動的機能搜索における関連度と探索戦略

中野真明貴[†] 林 晋平[†] 小林 隆志[†]

[†] 東京工業大学 情報理工学院 情報工学系 〒152-8552 東京都目黒区大岡山 2-12-1

E-mail: {maaki@sa., hayashi@se., tkobaya@}cs.titech.ac.jp

あらまし 形式概念分析を利用した動的機能搜索では、機能を実現する実装箇所は概念束内の複数の概念内に分散して現れる。このため搜索時には複数の概念を探索する必要がある。本稿ではこの探索効率の向上を目的として、機能との関連度に着目した概念束の探索戦略を提案する。機能との関連度を表す指標としては、欠陥箇所特定技法で利用される類似係数を用いる。代表的なオープンソースソフトウェアの複数の機能に対し提案手法による機能搜索を行い、既存手法との有用性の比較を行った。

キーワード 動的機能搜索 形式概念分析

Investigation Strategies and Relevance Metric on Dynamic Feature Location

Maaki NAKANO[†], Shinpei HAYASHI[†], and Takashi KOBAYASHI[†]

[†] Dept. of Computer Science, School of Computing, Tokyo Institute of Technology

2-12-1, Ookayama, Meguro-Ku, Tokyo, 152-8552 Japan

E-mail: {maaki@sa., hayashi@se., tkobaya@}cs.titech.ac.jp

Abstract In dynamic feature location based on formal concept analysis, it is a hard problem that modules implementing focused feature distribute dispersively among many formal concepts. For this reason, in locating, users must investigate many formal concepts. To improve the efficiency of this feature location technique, we propose two investigation strategies using a relevance metric. We use a similarity that is used in area of fault localization as a relevance metric. We applied our investigation strategies to several features of a representative open source software product and compared them with an existing strategy.

Key words Dynamic Feature Location, Formal Concept Analysis

1. はじめに

ソフトウェアに対する機能の修正や追加といった保守工程においては、既存のプログラムを正確に理解することが重要である。開発者はプログラム理解のために、ソースコードや開発ドキュメントなど様々な情報を参照するが、約 50%の時間をソースコードの読解に費やしていることが報告されている [1]。

プログラム理解を効率よく行う方法は古くから研究されており、近年ではプログラムの機能とその機能を実装するモジュールとのマッピングを作成し、着目する機能に関係のあるソースコードを限定することでプログラム理解を支援する、機能搜索が注目を浴びている [2]。機能搜索は、現在までに静的解析や動的解析、またモジュール名を解析するものなど様々な手法が提案されている [3]。その中でも、動的解析を利用した手法は、機能を実装するモジュールの多くが実行トレース中に現れるため高い精度が期待できる。

動的解析を利用した代表的な手法に、データ解析技術である

形式概念分析（FCA）を利用した手法 [4] がある。この手法では、複数の機能を同時に解析でき、各モジュールが機能の固有のモジュールであるのか、他機能との共有モジュールであるのかを区別できるという利点がある。また各モジュールが、注目している機能を実行する時に必ず実行されるモジュールであるか、特定の場合にのみ実行される注目している機能の分岐部分のモジュールであるかの理解が可能である。我々はこれまでに、多層システムへの適用のための拡張 [5] やインクリメンタルに解析を行う拡張 [6] を行い、FCA を利用した動的機能搜索の有用性を示してきた。

しかしながら、FCA を利用する手法は各モジュールと機能の関係を詳細に分析するため、機能を実装しているモジュールの一部が、解析結果である概念束中の複数の形式概念に分散する場合がある。この場合、搜索時に形式概念間の関係を理解しながら多くの形式概念を探索する必要があり、プログラム理解の作業コストが大きくなる問題点がある。

本稿ではこの探索効率の向上を目的として、機能との関連度

に着目した概念束の探索戦略を提案する。欠陥特定の技術を機能探索に用いることにより、高い精度で機能を実装する重要なモジュールを発見できるという結果[7]を受け、機能との関連度を表す指標としてはトレースに含まれる機能集合とモジュール実行との類似係数を用いる。本稿では、類似係数を用いて探索範囲を広げる手法と、類似係数により探索範囲を予め絞る手法の2つの探索戦略を提案する。

提案手法の評価のために、オープンソースソフトウェアの複数の機能に対し、既存手法と提案手法による機能探索実験を行った。本稿では、機能探索の精度の比較と、提案手法が他機能との共通部分と注目している機能の分岐部分どちらに対し効果的であるかについて議論を行う。

2. 関連研究

2.1 形式概念分析を利用した動的機能探索

形式概念分析 (FCA) [8] は数学における束論に基づくデータ解析手法の一つである。FCA を用いることで、解析対象となるオブジェクトとその属性との間の二項関係を表した形式文脈から概念束を構成する事ができる。解析者は概念束を見ることでオブジェクトと属性の関係だけでなく、オブジェクト同士・属性同士の関係性も理解できるようになる。

FCA を利用した動的機能探索は Eisenbarth らによって提案された [4]。この手法において FCA のオブジェクト O は機能探索の対象となるプログラムの実行トレースであり、属性 $\mathcal{A}$ は対象プログラム中のモジュールや機能である。 O と $\mathcal{A}$ の間の二項関係 I について、実行トレース o においてモジュールや機能 a が実行されていたなら、 $(o, a) \in I$ とする。三つ組 $C = (O, \mathcal{A}, I)$ を形式文脈と呼び、これを入力とし概念束を構成する。

FCA を利用した動的機能探索の例を示す。表 1 のような形式文脈が存在するとする。この表において、 $s_1 \sim s_4$ は実行トレースを、 $f_1 \sim f_3$ は機能を、 $m_1 \sim m_8$ はモジュールを表す。この形式文脈から構成される概念束は図 1 である。

図 1 の各ノードは形式概念を表している。形式概念はオブジェクトの集合である外延と、属性の集合である内包から構成される。FCA を利用した動的機能探索においては、外延は実行トレースの集合であり、内包はモジュールや機能の集合である。また形式概念 c から図の上方向に辺をたどり到達できる形式概念を c の上位概念という。逆に形式概念 c から図の下方向に辺をたどり到達できる形式概念を c の下位概念という。図 1 において、形式概念 $(\emptyset, \{f_1, m_1\})$ の上位概念は $(\emptyset, \{m_4\})$ と $(\emptyset, \{m_8\})$ であり、下位概念は $(\{s_4\}, \{m_6\})$ と $(\{s_1\}, \{m_7\})$ と $(\emptyset, \emptyset)$ である。

機能 f_1 について、機能 f_1 が存在する形式概念 $c_{f_1} = (\emptyset, \{f_1, m_1\})$ を見ると、モジュール m_1 は機能 f_1 の実装に深く関わっている事が推測できる。モジュール m_1 は機能 f_1 が利用された場合は必ず実行され、それ以外の場合では実行されない。次に、 c_{f_1} の上位概念の内包から、モジュール m_4 と m_8 は機能 f_1 の実装に用いられているが、他の機能とも共有していることが推測できる。また c_{f_1} の下位概念の内包から、モジュール m_6 と m_7 は機能 f_1 の実装に用いられているが、分岐部分で利用されているため常に呼び出されるわけではないことが推測できる。

表 1 機能探索における形式文脈の例

	f_1	f_2	f_3	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
s_1	✓	✓		✓	✓		✓	✓		✓	✓
s_2			✓			✓		✓			✓
s_3		✓	✓		✓	✓	✓	✓			✓
s_4	✓			✓			✓		✓		✓

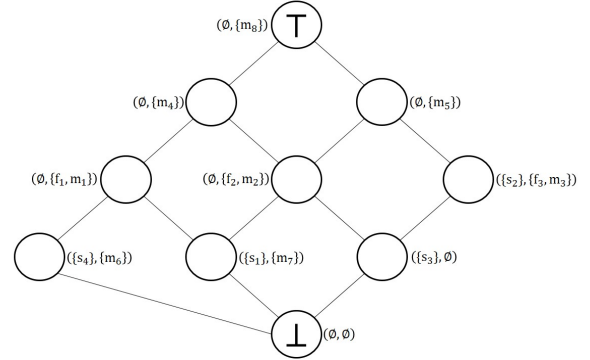


図 1 表 1 から構成される概念束

この様に機能の存在する形式概念の上位概念は他機能との共有モジュールを含んでおり、下位概念は特定の場合のみ実行される分岐部分のモジュールを含んでいる。FCA を利用した動的機能探索においては注目している機能の存在する形式概念と、その上位下位の概念を調査することが重要となる。しかし一般に上位と下位の概念の数は多く、そのすべてを調査する事は困難であるため探索戦略が必要となる。

2.2 欠陥特定を用いた動的機能探索

スペクトラムに基づく欠陥特定 (SFL) は、ソフトウェアの各モジュールに対して欠陥を含む可能性を計算することで、欠陥箇所を特定するデバッグ技術である [9]。どのテストが失敗したかを記録した欠陥ベクトル e と、モジュールがどのテストで実行されたかを記録したヒットスペクトラム A の各行との類似度が欠陥を含む可能性を表すため、これを類似係数で数値化し欠陥特定を行う。

この SFL を用いて動的機能探索を行う手法 [7] が Perez らによって提案されている。この手法では、各実行トレースで機能探索の対象となっている機能が利用されたかどうかを記録したベクトル e を欠陥ベクトルとして用い、実行トレースと各実行トレースで実行されたモジュールからヒットスペクトラム A を構成する。計算された類似係数が 1 であるモジュールは注目している機能の実装のみに関わっていることを示し、類似係数が 0 であるモジュールは注目している機能には全く関係のないことを示す。Perez ら手法では、機能の実行の有無とテスト実行の成否を対応付けるため、注目する 1 つの機能のみしか機能探索できず、他の機能との共有するモジュールと特定の場合のみ実行されるモジュールを区別できないという問題がある。

一般に SFL で利用される類似係数は様々なものが提案されている。本稿では優れた精度を持つ Ochiai の類似係数 [10], [11] を用いる。モジュール j の Ochiai の類似係数 $s_o(j)$ は以下の式であらわされる。

$$s_o(j) = \frac{n_{11}(j)}{\sqrt{(n_{11}(j) + n_{01}(j)) \cdot (n_{11}(j) + n_{10}(j))}} \quad (1)$$

ただし $n_{pq}(j)$ は以下のように定義されているとする。

$$n_{pq}(j) = ||i \mid a_{ij} = p \wedge e_i = q|| \quad (2)$$

つまり $n_{11}(j)$ は機能検索の対象となっている機能を利用したトレースのうち、モジュール j を実行したトレースの数を表している。

2.3 既存手法の問題点

2.1 節で述べた FCA を利用した動的機能検索は機能同士の関係性も解析できる点で優れているが、解析対象が大きなプログラムになると解析結果である概念束が非常に煩雑になり、機能を実現する実装箇所が概念束中の複数の形式概念に分散してしまうという問題点が存在する。そのため検索時に複数の形式概念の探索が必要となる。

ここで特に工夫せず、機能 F の存在する形式概念 c_F から N 回以下の辺の遷移で到達できる c_F の上位概念および下位概念のみを探索する方法を考える。この手法は c_F から構造的に近い c_F の上位下位概念のみを探索するという戦略である。 c_F の上位下位ではない形式概念に含まれているモジュールは、機能 F が利用される場合に実行されないモジュールであるか、機能 F の分岐部分で実行される他機能との共有モジュールである。このようなモジュールは c_F の上位下位概念に含まれているモジュールに比べ、機能 F を対象とした機能検索において重要度が低い [4] ため、探索範囲から除外している。この c_F から構造的に近い上位下位概念のみを探索する手法は、概念束を探索する戦略として素直な方法である。本稿ではこの手法を既存手法として扱い、提案手法との比較対象として用いる。

2.2 節で述べた SFL を用いた動的機能検索は類似係数が 1 や 0 であるものに対しては高い精度で機能検索が可能であるが、値が 0 と 1 の中間であるものに対しては機能に関係しているか曖昧であるという問題点が存在する。これは機能検索を行う際に、検索の対象としている一機能のみから重要であるかどうかを計算しているためであると考えられる。一機能のみに注目しているため類似係数が 0 と 1 の中間であることが共有モジュールであるのか、機能の分岐部分のモジュールであるのか、それとも機能には関係の薄いモジュールであるのか判別できない。

提案手法では先行研究のそれぞれの問題点を解決するため、複数の機能を同時に解析した結果である FCA を利用した動的機能検索の結果に対し、SFL で用いられた類似係数を適用し、機能との関連度に着目した概念束の探索戦略を提案する。

3. 提案手法

3.1 概要

本研究の目的は FCA を利用した動的機能検索の探索効率を向上させることであり、この目的を達成するために本稿では機能との関連度に着目した概念束の探索戦略を提案する。この章ではまず FCA と類似係数の関係性について述べ、機能との関連度に着目した 2 通りの探索戦略を提案する。

- ・ 注目している機能に対する類似係数が閾値以上の形式概

念のみを探索する戦略（提案手法 1）

- ・ 他機能に対する類似係数との比較を行い、予め探索範囲を絞る戦略（提案手法 2）

3.2 FCA と類似係数の関係性

この節では FCA と類似係数の関係性について説明する。

まず形式概念の内包に含まれている各モジュールは実行されたトレースの集合が一致するため、各形式概念の内包に属するモジュールはすべて同じ値の類似係数を持つ事が言える。このことから形式概念ごとに一つの類似係数が算出される事が言える。本稿では内包に属するモジュールがとる類似係数の値を形式概念の類似係数とする。

機能 f が存在している形式概念 c_f の機能 f に対する類似係数は 1 になる。これは c_f の内包に属するモジュールが実行されたトレースの集合が機能 f が利用されたトレースの集合に一致するからである。また形式概念の機能 f に対する類似係数の値は c_f から上位に進めば進むほど、また下位に進めば進むほど小さくなる。これは上位や下位に進むほど、その形式概念に属するモジュールが実行されたトレースの集合と機能 f が利用されたトレースの集合との違いが大きくなるためである。

以上のことより c_f の上位下位概念が c_f にどの程度近いかを表す指標として類似係数が利用できると本稿では考える。この考えのもと本稿では本来比較できない他機能に対する類似係数と比較する。これは類似係数が近さを表すと考えており、各形式概念がどの機能に近いかを評価する為である。

3.3 提案手法 1

提案手法 1 について、注目している機能 F に対し形式概念の探索を行う場合、既存手法と同様に機能 F が存在する形式概念 c_F 自身と c_F の上位下位概念のみから探索を行う。

提案手法 1 においては、注目している機能 F に対する類似係数のみを用いる。本手法では、機能 F に対する類似係数が閾値 T 以上の形式概念 c のみを探索する。3.2 節の最後で述べたように、類似係数は各形式概念の c_F との近さを表すため、類似係数が高い形式概念を探索することで、提案手法 1 では注目している機能に密接に関係する形式概念のみを探索する事が可能であると考えられる。

3.4 提案手法 2

提案手法 2 においても、既存手法や提案手法 1 と同様に注目している機能 F に対し形式概念の探索を行う場合、機能 F が存在する形式概念 c_F 自身と c_F の上位下位概念のみから探索を行う。

提案手法 2 について説明するために、記号の定義を行う。まず FCA において属性として用いた機能の集合を $\mathcal{F}$ 、機能検索の対象として注目している機能を F とする。現在注目していない機能の集合を $\mathcal{F}_{other}$ とする。 $\mathcal{F}_{other} = \mathcal{F} \setminus \{F\}$ である。機能 f に対する形式概念 c の類似係数を $s_o(f, c)$ で表す。また、 $topOtherScore(c) = \max_{f \in \mathcal{F}_{other}} s_o(f, c)$ とする。

以上の記号を用いて、提案手法 2 における探索の条件を以下に示す。

$$topOtherScore(c) \leq 1 - rate * (1 - s_o(F, c)) \quad (3)$$

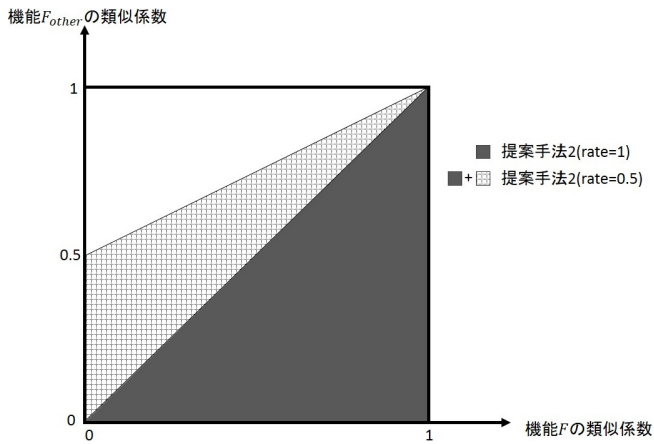


図2 提案手法2 (rate = 1) が探索する類似係数の範囲

提案手法2では各形式概念が式(3)を満たすか予め計算し、式(3)を満たす形式概念 c のみに探索範囲を絞る。 $rate$ は条件の厳しさを変化させるための正変数であり大きいほど条件が厳しくなる。

式(3)は少々直観的ではないため、機能集合が $\mathcal{F} = \{F, F_{other}\}$ の2機能から構成される場合に、機能 F に対して式(3)が真になる範囲を図2に図示する。図2の横軸は機能 F の類似係数を表し、縦軸は F_{other} の類似係数を表している。図2で灰色の範囲が提案手法2 (rate = 1) を、灰色と網掛け部を合わせた範囲が提案手法2 (rate = 0.5) を表している。

図2からもわかるように式(3)は他機能の類似係数と比較して大きい範囲を表す条件式である。類似係数が各形式概念と c_F との近さを表すため、類似係数の比較を行うことで各形式概念が c_F に近いのか、それとも他機能 F_{other} が存在する形式概念 $c_{F_{other}}$ に近いのかを判別できる。これにより類似係数が1にならない c_F の上位下位概念についても機能と密接に関係しているかどうかの判別ができる。

4. 実験

4.1 概要

提案手法の評価のために、代表的なオープンソースソフトウェアに対しFCAを用いた動的機能探索を行い、出力された概念束に対して既存手法と提案手法を適用し、探索範囲に含まれた形式概念が機能に密接に関係しているかの実験を行った。実験を行うにあたり、まず対象プログラムにFCAを利用した動的機能探索を行った(4.2節)。その後出力された概念束に対し既存手法および提案手法1, 2を適用し、有用性の比較検証を行った(4.3節)。

4.2 実験対象

本節では手法の適用対象となる概念束を構築する手順および実験の際に正解として用いるデータセットの作成方法について説明する。

4.2.1 対象ソフトウェア

本稿では、実験の対象のプログラムとして、オープンソース

表2 実験で用いたRhinoのテストと機能の形式文脈

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13
S1-1	✓	✓				✓		✓					
S1-2	✓	✓				✓					✓		
S2-1	✓		✓						✓				
S2-2	✓		✓						✓				
S2-3	✓								✓				✓
S3-1	✓		✓	✓		✓	✓						✓
S3-2	✓		✓	✓		✓	✓						✓
S3-3	✓		✓	✓		✓	✓						✓
S3-4	✓		✓	✓		✓	✓						✓
S4	✓			✓	✓					✓			
S5	✓				✓	✓							
S6	✓			✓		✓							
S7-1	✓		✓	✓		✓							✓
S7-2	✓		✓	✓		✓							✓
S7-3	✓		✓			✓							✓
S7-4	✓		✓	✓		✓							✓
S7-5	✓		✓	✓		✓							✓

プロジェクトのRhino^(注1)を選択した。その理由は、Rhinoが代表的なオープンソースソフトウェアであり、ソースコードのサイズが十分な大きさを持っており、十分な量のJUnitで記述されたテストが存在したからである。RhinoはMozilla Foundationによって管理配布されている、Javaで記述されたJavaScriptの実装である。本実験ではRhinoのバージョン1.7.7を用いた。Rhinoのバージョン1.7.7のソースは16パッケージ、241クラス、約75KLOCで構成されている。

4.2.2 形式文脈

著者らのうち1名が選んだRhinoの13個の機能についてFCAを行った。FCAの入力である実行トレースを取得するために、Rhinoの7個のJUnitテストを利用した。機能とトレースを対応付けるため、テストクラス名やテスト内容を参考にし、テストクラスのテストメソッドごとにどの機能を利用したかをチェックした。各テストと機能の対応の形式文脈を表2に示す。表2においてS1からS7はテストクラスを、ハイフンの後の数字はクラス内のテストメソッドを表す。

4.2.3 FCAの結果

FCAを行うためにConcept Explorer^(注2)のバージョン1.3を利用した。またテストの実行トレースを取得するためにyebisu^(注3)を利用した。yebisuはJavaのバイトコードにプログラムを埋め込むことで実行時に呼ばれたメソッドを出力する。

これらのツールを使い13個の機能に対してFCAを行った。今回の機能探索においてモジュールにはメソッドを用いた。各テストにおいて実行されたメソッドを、yebisuを用いて記録し、Concept Explorerを用いて概念束を構築する。

7テスト合計で実行されたメソッド数は1206メソッドで、機能と合わせた1219個の属性に対しFCAを行った。解析結果中

(注1) : <https://developer.mozilla.org/ja/docs/Rhino>

(注2) : <http://conexp.sourceforge.net/>

(注3) : <http://github.com/salab/yebisu>

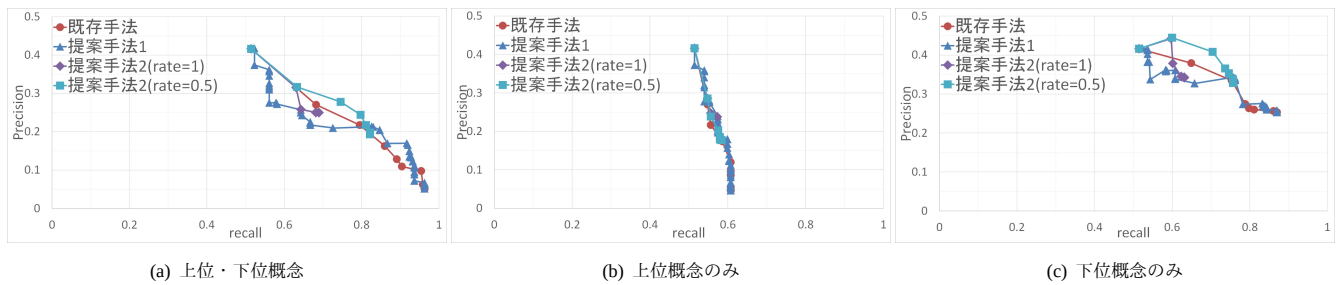


図3 10機能平均 precision と平均 recall

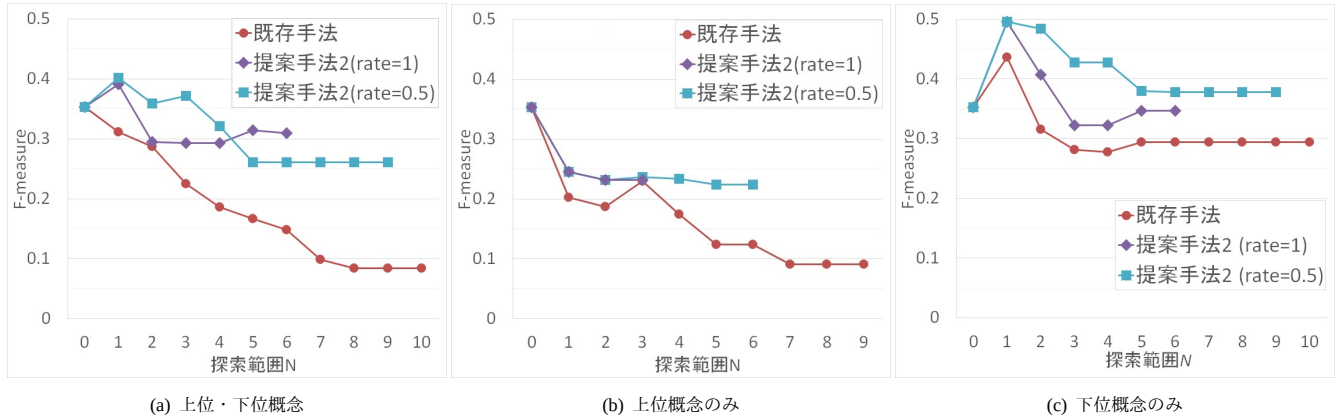


図4 10機能の F-measure の中央値

の形式概念の数は 381 個であった。機能検索を行う際には多数の形式概念間の関係を理解しながら探索をする必要があり、プログラム理解の作業コストが大きくなる。そのため、本稿で提案する探索戦略が必要不可欠である。

4.2.4 正解とするメソッドの作成

提案手法の有用性を比較検証するために、各機能に対して密接に関係しているメソッドの一覧を作成する。Rhino のチュートリアル^(注4)やソースコードおよび各テストの情報から、各トレースで呼び出されたメソッドが機能に密接に関係しているかを著者らのうちの 1 名が判断した。作成したメソッドの一覧は実験結果を検証する上で探索範囲に含むべき正解として扱う。

4.3 実験手順

4.2 節で構成した概念束に対して、既存手法および提案手法 1, 2 を適用した。本稿では 2.3 節で説明した素直な方法を既存手法として扱う。既存手法の探索範囲 N に関しては 0 から 10 まで変化させる。 $N = 0$ の場合は c_f のみが探索される。

提案手法 1 の閾値 T に関して 1 から 0.01 まで 0.01 刻みで変化させて適用する。提案手法 2 については $rate \in \{1, 0.5\}$ で適用する。また提案手法 2 については既存手法と同様に c_f から N 以下の遷移で到達できる概念を探索するとし、 N を 0 から 10 まで変化させ適用する。

各手法の探索範囲に含まれたメソッドが機能と密接に関係しているかを 4.2.4 項で作成した正解セットで判定し、precision, recall および F-measure で各手法を評価した。

4.4 実験結果

機能 f7, f8, f11 に関して正しく機能検索を行うことができ

なかったため実験結果から取り除いた。正しく機能検索を行うことができなかった理由については 4.5 節で議論する。

提案手法 1 の閾値 T を動かした場合および既存手法と提案手法 2 ($rate \in \{1, 0.5\}$) の遷移回数 N を動かした場合における、10 機能での平均 precision, recall の遷移を図 3(a) に示す。また提案手法が上位と下位どちらに効果があったかを評価するために、探索範囲を上位下位に絞ったグラフを図 3(b), 図 3(c) に示す。図 3(a) において、各線の左上が閾値 $T = 1$ または $N = 0$ での平均で右下に行くほど閾値の値が低くなる (N が大きくなる)。

次に既存手法と提案手法 2 ($rate \in \{1, 0.5\}$) の N を動かした場合における、F-measure の 10 機能の中央値を図 4(a) に示す。各手法において探索範囲が広がらなくなった N でグラフの描画を止める。また先ほどと同様に、探索範囲を上位下位に絞ったグラフを図 4(b), 図 4(c) に示す。

4.5 考察

図 3(a) において提案手法 2 ($rate = 0.5$) の $1 \leq N \leq 4$ 部分のグラフから、提案手法 2 ($rate = 0.5$) が既存手法と比べ precision を大きく下げずに recall を向上させている事がわかる。また、図 4(a) において、提案手法 2 ($rate = 0.5$) は探索範囲 N の全ての値において既存手法と比べ等しいか上回っていることがわかる。これは提案手法 2 ($rate = 1$) においても同様であり、以上のことから提案手法 2 が既存手法より優れていることが示された。

提案手法 2 の $rate$ については、図 3(a) において $rate = 1$ の場合の recall が 0.7 以下に留まることや、図 4(a) の $1 \leq N \leq 4$ において $rate = 1$ の場合の F-measure が $rate = 0.5$ の場合と比べ下回っていることがわかる。これらのことから、 $rate = 0.5$ の方が $rate = 1$ の場合より優れていることがわかる。 $rate = 0.5$ の

(注4) : <http://www.mozilla-japan.org/rhino/tutorial.html>

場合の方が優れていた理由としては、 $rate = 1$ では式 (3) の条件が厳しく、探索範囲が狭くなりすぎたためと考えられる。

また図 3(a) の precision が 0.2 より大きい範囲において提案手法 1 が既存手法より悪い結果を示していることがわかる。このことから類似係数を使い探索範囲を広げていく戦略よりも既存手法のように概念束の構造を利用して探索範囲を広げていく戦略のほうが有用であることが分かった。また以上より類似係数は探索の段階で範囲を広げるために利用するのではなく、予め他機能との比較によって探索範囲を絞るために利用する方が効果的であると考えられる。

図 3(c), 図 4(c) から、提案手法 2 が下位において有用であることが分かった。下位のみでの最大 F-measure も既存手法では 0.44 であるが、提案手法 2 ($rate = 0.5$) では 0.50 に上昇しており、提案手法 2 は下位概念に対し良い効果があるといえる。次に図 3(b), 図 4(b) から、提案手法 2 は上位に対しては効果が薄いことが分かった。図 4(b) から既存手法と比べ F-measure が下がりきらない点では優れているものの、 $N = 0$ の場合が最大でありそこから N を増やすと下がってしまうことがわかる。以上より提案手法 2 は共有モジュールを含む上位概念より分岐部分のモジュールを含む下位概念に対して効果があると考えられる。上位に対して効果が薄かった理由としては、図 3(b) からわかるように上位概念の範囲に正解が少なく、探索範囲を広げると precision が急激に下がってしまうことなどが挙げられる。

提案手法で正しく機能検索を行うことができなかった機能について考察を行う。機能 f8 と f11 は提案手法 2 では探索範囲にモジュールを含めることができなかった。これはこの二つの機能が機能 f2 と非常に近い機能でありかつ、f2 に完全に包含されてしまっていることが原因であると考えられる。実際機能 f8 や f11 を実装しているのは機能 f2 を実装する上で重要な役割を果たしているメソッドであり、f8 と f11 の 2 機能の区別はメソッド内の if 分岐で実装されている。このような非常に近い包含関係のある機能（例えば「決済」と「クレジットカード決済」）が両方機能検索の対象に入っている場合や、二つの機能を分離している部分が機能検索の粒度（今回ならばメソッド）よりも細かい粒度で実装している場合、提案手法 2 では包含している機能の類似係数が、包含されている機能の類似係数より大きくなってしまうため正しく機能検索を行うことができない。

機能 f7 はどの手法でも正しく機能検索を行うことができなかった。これは機能 f7 に密接に関係しているメソッドが共有メソッドであったことが原因である。ソースコード中から見つけた機能 f7 の正解メソッドは概念束のトップにあたる形式概念に存在していた。本稿の手法では機能に密接に関係しているメソッドは他の機能とは密接に関係していないという前提で探索を行っているため、正しく機能検索を行うことはできなかった。このような機能に密接に関係しているメソッドを見つけるためには、本稿の手法のような形式概念ごとに探索範囲を広げる手法ではなく、形式概念の内包に含まれるメソッドを評価してメソッドごとに探索する手法を考案しなければならない。

4.6 妥当性への脅威

構造的妥当性としては評価のために作成した正解セットは著

者らのうちの 1 名が判断し決めたものであるため、誤ったものを含む可能性がある。また今回の解析の実行トレースに含まれなかったメソッドに関しては調査を行っていないため、調査範囲外に探索の正解となる機能に密接に関係しているメソッドが存在している可能性がある。

外的妥当性としては単一のプログラムの限られた機能に対してのみ実験を行ったため、ほかのプログラムや機能セットに対しては異なる結果を得る可能性がある。

5. ま と め

FCA を利用した動的機能検索における、機能を実現する実装箇所が概念束内の複数の形式概念に分散するという問題点を解決するために、機能との関連度に着目した概念束の探索戦略を提案した。機能との関連度を表す指標としては欠陥箇所特定技法で利用される類似係数を用い、類似係数を用いて探索範囲を広げる手法と、他機能に対する類似係数との比較により探索範囲を予め絞る手法の 2 つの探索戦略を提案した。

代表的なオープンソースソフトウェアの複数の機能に対し提案手法による機能検索を行い、他機能との比較により予め探索範囲を絞る手法が有用であること、および概念束上位よりも下位に対し効果が高いことを示した。

謝 辞

本研究の一部は科研費（#24300006, #25730037, #26280021, #15H02683, #15K15970）の助成を受けた。

文 献

- [1] R. Tiarks, “What programmers really do: An observational study,” *Softwaretechnik Trends*, vol.31, no.2, pp.36–37, May 2011.
- [2] V. Rajlich and N. Wilde, “The role of concepts in program comprehension,” *Proc. IWPC2002*, pp.271–278, 2002.
- [3] B. Dit, M. Revelle, M. Gethers, and D. Poshypanyk, “Feature location in source code: a taxonomy and survey,” *Journal of Software: Evolution and Process*, vol.25, no.1, pp.53–95, 2013.
- [4] T. Eisenbarth, R. Koschke, and D. Simon, “Locating features in source code,” *IEEE TSE*, vol.29, no.3, pp.210–224, 2003.
- [5] H. Kazato, S. Hayashi, S. Okada, S. Miyata, T. Hoshino, and M. Saeki, “Feature location for multi-layer system based on formal concept analysis,” *Proc. CSMR2012*, pp.429–434, 2012.
- [6] H. Kazato, S. Hayashi, T. Kobayashi, T. Oshima, S. Okada, S. Miyata, T. Hoshino, and M. Saeki, “Incremental feature location and identification in source code,” *Proc. CSMR2013*, pp.371–374, 2013.
- [7] A. Perez and R. Abreu, “A diagnosis-based approach to software comprehension,” *Proc. ICPC2014*, pp.37–47, 2014.
- [8] B. Ganter and R. Wille, *Formal concept analysis: mathematical foundations*, Springer Science & Business Media, 2012.
- [9] R. Abreu, P. Zoetewij, and A.J. Van Gemund, “On the accuracy of spectrum-based fault localization,” *Proc. TAICPART-MUTATION '07*, pp.89–98, 2007.
- [10] R. Abreu, P. Zoetewij, and A.J. Van Gemund, “An evaluation of similarity coefficients for software fault localization,” *Proc. PRDC'06*, pp.39–46, 2006.
- [11] A.d.S. Meyer, A.A.F. Garcia, A.P.d. Souza, and C.L.d. Souza Jr, “Comparison of similarity coefficients used for cluster analysis with dominant markers in maize (zea mays l),” *Genetics and Molecular Biology*, vol.27, no.1, pp.83–91, 2004.