Vol.2017-CE-138 No.12
2017/2/12

機能に基づくプログラム構造比較に向けた類似度の策定

間嶋　義喜[1,a)]　大賀　賢志[1,b)]　竹内　和広[1,c)]

概要：プログラム作成では同一の機能を実現するために，典型的な実装方法が存在する場合や多様な実装方法が可能な場合とがある．本発表では，オブジェクト指向型プログラミング言語のソースコードの識別子を基準に，その構造の類似度を定義することを試みる．具体的には，プログラム教育の上で，よく課題とされる機能を題材に，各種機能のプログラム構造を特徴づける上で適切なグラフ表現の抽象化について検討した．

キーワード：プログラム構造評価，プログラム類似度，プログラム教育

An assesment of similarity measure for comparing structure of program code for the same assignment

Mashima Yoshiki[1,a)]　Oga Satoshi[1,b)]　Takeuchi Kazuhiro[1,c)]

Abstract: For answering one assignment for programing skills development, there are more than one way even if typical and fundamental implementation patterns exist in programming. This article assesses a similarity of the structure of program codes based on the identifier of the source code of the object oriented programming language. Specifically, we designed a graph representation for abstracting the program structure with finding fundamental differences among sample codes in typical text books for beginners of programming.

Keywords: Evaluation of Program Structure, Similarity Measure between Programs, Programming Skills Development

1. はじめに

プログラムは目的をもって作成される．そのため，プログラムが要求される目的を達成できているかどうかは，あらかじめプログラムに与えられる入力と，要求の達成を確認するための，入力に対して求められる出力を用意しておけば，プログラムの動作検証を行うことができる．しかし，プログラム教育の文脈では，プログラムは単に要求される動作をすればよいというわけではなく，要求を達成するための方法についても評価の対象となりうる．本稿の提案は，同一目的のプログラム集合に対して，それに属する個々のプログラムをプログラム構造の観点から類似性を定義し，プログラム集合内の構造多様性を視覚的に把握することに貢献しようとするものである．

プログラム記述にはアルゴリズム選択の自由があり，特定の目的には通常，複数の典型的なアルゴリズムが存在する．また，目的を達成するためには，複数のアルゴリズムを組み合わせて使用しなければならない場合もある．そして，特定のアルゴリズムを実装するための制御順序，条件式等が多様に記述できることも少なくない．このような背景から，プログラム学習を指導する際，学習者のプログラムを人手で判断することは多大な労力を要する作業となってしまう．そのため，プログラムの記述方法の多様性を尊重した指導を行いたくても，学習者の個別的な実装について精緻な評価をすることは難しい．

他方，プログラムで特定の目的を実現するための一般的

[1] 大阪電気通信大学
Osaka Electro-Communication University
[a)] gp13a127@oecu.jp
[b)] mi16a001@oecu.jp
[c)] takeuchi@isc.osakac.ac.jp

な記述方法をどのように呼称すべきかも明らかではない．例えば，Java などのオブジェクト指向のプログラム言語では，メソッド名に自然言語の語が使われる．しかし，メソッド名に使われた語から，当該のプログラム部分が，どのような機能を担っているかを推定することは簡単ではない．それは，メソッド名を構成するプログラム部分が，手続き型言語のプログラムの 1 行 1 行の機能の複合体として構成され，その機能がプログラムの 1 行単位が実現する機能に比べ，多様であり，メソッドの実装よりも粒度が細かく，プログラムの行単位の機能よりもおおきなプログラム部分が担う機能を自然言語でラベル付けすることは困難である．このようにプログラムの部分要素は適切な言葉で呼称し共通認識を得ることが難しいため，プログラムの部分要素の違いは，何らかの観点から他のプログラムとの類似性や相違性から説明することが適切であると考えた．

以上のような背景から，本稿では，プログラムのソースコードの特徴を抽象的なグラフ構造で表現し，そのグラフ表現を数理的に扱いやすくするために，プログラム 1 つ 1 つをプログラム構造の観点から類似性と相違性を計量できるベクトル空間上のベクトルとして表現することを提案する．

2. プログラム構造のグラフ表現

プログラムの初学者は，各種のプログラムに関する教科書を参考にしつつ，プログラム学習を行っていくものと考えられる．そういった初学者向けの教科書の多くには，プログラムが特定の要求を実現するための，基本的なプログラムの部分要素を例として掲載している．

本稿では，教科書にあるような基本的なプログラムに対して，そのプログラム構造を反映したクラスタリング表示を行うことを目的とする．そのためには，クラスタリングの対象となるプログラム集合を設定した時に，集合の要素となる個々のプログラムをベクトルで表現できることが望ましい．すなわち，プログラム集合中のプログラム構造の類似点や相違点をベクトル空間上の類似度で表現する．

2.1 採用データ構造に基づくプログラム構造の特徴付け

本稿では，プログラムをベクトルで表現するための中間点として，プログラムをグラフ構造で表現する．その際，当該のプログラムがどのようなデータ構造を採用し，そのデータ構造をどのように扱っているかをグラフ表現することを考える．

特に基本的なプログラムにおいて，データ構造はそのプログラムがどのようなプログラム構造を採用しているかを特徴付ける．ソートのアルゴリズムを例に挙げると，ソートのアルゴリズムの目的は，あるデータ列の数値の並び替えを行うことであり，そのために値を記憶する変数や配列といったデータ構造が必要となる．そして，それらデータ構造の扱い方や操作の仕方などが，ソートのアルゴリズムにバリエーションを持たせている．

具体的には，バブルソートは，与えられたデータ列の端から参照を行い，隣り合った数値を比較することで並び替えを行うアルゴリズムであるが，クイックソートは，データ列の数値中から特定の数値を基準として選択し，残りのデータを大小で分割する．そして，この一連の操作を分割したデータ列ごとに行うアルゴリズムである．このように，バブルソートとクイックソートでは，データ列に対する参照の違いがアルゴリズムという機能の違いを生んでいる．以上のように，プログラムを実装する際に，どのようなデータ構造をどのように扱っているかを，プログラム構造を表現する上で極めて重要であると位置付けた．

実際のグラフ表現では，同じ型のデータ構造であっても，それをいくつインスタンス化して利用するかも，プログラム構造の特徴として位置付ける．

例えば，図 1 のような Java 言語で記述された，与えられた配列の中にある 2 つの値を交換する swap プログラムが定義されていたとする．1 行目において宣言されている int 型の変数 x と int 型の変数 y は，同じく 1 行目に宣言されている int 型配列 Array のインデックスとして扱われている．また，2 行目において宣言されている int 型の変数 tmp は，交換する値を一時的に記憶するために使用されている．これにより，同じデータ構造である変数が，その宣言ごとに異なる目的に沿って使用されていることがわかる．

2.2 提案するプログラム構造グラフ

前節の議論を踏まえて，本稿が提案するプログラム構造を反映するグラフ表現の詳細を説明する．

図 1, 図 3 のソースコードを例に，二部グラフを構築する．それぞれ構築されるグラフは図 2, 図 4 であり，各ノードの番号とソースコードの識別子との対応を表 1 に示す．

二部グラフとは，ノードの集合を 2 つの集合 V_1, V_2 に分割し，エッジは集合 V_1 のノードと集合 V_2 のノードの間で引かれるグラフのことを指す．ノードの集合 V_1 は宣言されたデータ構造の種類とその宣言名の組であり，V_2 は参照されているメソッドや制御構造における引数や条件に合致した場合の処理などの取り扱われ方となっている．またエッジの重みは，宣言されたデータ構造に対する参照回数を示している．

まず図 1 のソースコードを見ると，1 行目において int 型配列 Array，int 型変数 x，y，2 行目において int 型変数 tmp が宣言されている．次に 2，3，4 行目において int 型配列 Array が計 4 回配列参照され，2，3 行目で int 型変数 x が計 2 回，3，4 行目で int 型変数 y が計 2 回，4 行目で int 型変数 tmp が計 1 回変数参照されている．

これらを基にグラフを構築すると，図 2 のグラフとなる．

```
1: static void swap (int[] Array, int x, int y){
2:         int tmp = Array[x];
3:         Array[x] = Array[y];
4:         Array[y] = tmp;
5: }
```

図 1　Java 言語で記述された swap プログラム

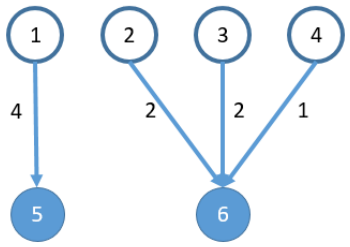

図 2　図 1 の swap プログラムによるグラフ表現

表 1　ノード番号と識別子との対応表

ノード番号	識別子
1	int[]_Array
2	int_x
3	int_y
4	int_tmp
5	配列参照
6	変数参照
7	String[]_Array
8	String_data
9	int_i
10	equals
11	length
12	for_if_Argument
13	for_if_Body

宣言された配列や変数から，配列参照と変数参照の 2 つのノードに重み付き有向エッジが引かれている．このときの重みは，宣言された各データ構造の参照回数となっている．

次に図 3 のソースコードを見ると，1 行目において String 型配列 Array と String 型 data が宣言され，2 行目において int 型変数 i が宣言されている．そして 4，6 行目で String 型配列 Array が計 2 回配列参照されており，4 行目で String 型変数 data が計 1 回変数参照，4，5，7，9 行目で int 型変数 i が計 4 回変数参照されている．

次にデータ構造の操作に対して，4 行目で String 型配列 Array が equals メソッドで使用されており，6 行目で同じく String 型配列 Array が length メソッドで使用されている．

またデータ構造に対する制御構造での取り扱われ方においては，3，4 行目，3，6 行目と繰り返し構文の入れ子になっている条件分岐の引数として String 型配列 Array が計 2 回使用されており，同じく 3，4 行目の繰り返し構文の入れ子になっている条件分岐の処理の部分と，3，8 行目

```
1: static int search(String[] Array, String data) {
2:         int i = 0;
3:         while (true) {
4:                 if (Array[i].equals(data))
5:                         return i;
6:                 if ((Array.length)−1 == i)
7:                         return −1;
8:                 else
9:                         i++;
10:        }
11: }
```

図 3　Java 言語で記述された search プログラム

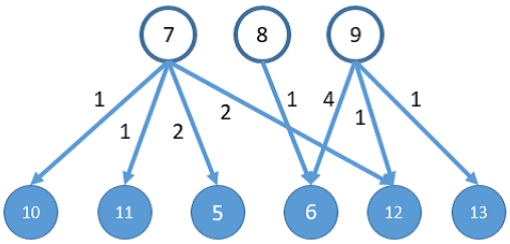

図 4　図 3 の search プログラムによるグラフ表現

の繰り返し構文の入れ子になっている条件分岐の引数に，int 型変数 i が 1 回ずつ使用されている．よってこれらから，図 4 に示すグラフが構築される．

制御構造において参照される箇所は条件部である引数と，条件に合致した場合に処理される部分の 2 か所である．このとき複数の制御構造が入れ子構造になっており，かつ参照されている宣言されたデータ構造がある場合，新しくノードを作成する．

このようにグラフ構造を定義することにより，ソースコードにおいて定義されたデータ構造がどのように使用され，制御構造上でどのように利用されているかを表現することができる．

2.3　グラフ表現から生成されるベクトル表現

前節の二部グラフを基にプログラムごとにベクトル表現を記述する．ベクトルの次元は，重みのある有向エッジを持つ二部グラフにおける 2 つのノード集合の組み合わせとし，その要素をエッジの重みとしてベクトル表現する．このように表現することで二部グラフを多次元のベクトル表現で記述することができる．例えば，図 2，4 をベクトル表現で記述するためには，ベクトルの次元を正規化することが必要となる．ここではベクトルの次元は，図 2，図 4 の二部グラフにおける 2 つのノード集合の組み合わせとなるので，ノードの集合を $V = \{V_1, V_2\}$ とするとそれぞれの要素は，$V_1 = \{1, 2, 3, 4, 7, 8, 9\}$，$V_2 = \{5, 6, 10, 11, 12, 13\}$ となり，V_1 のノードから V_2 のノードへ引かれている重み

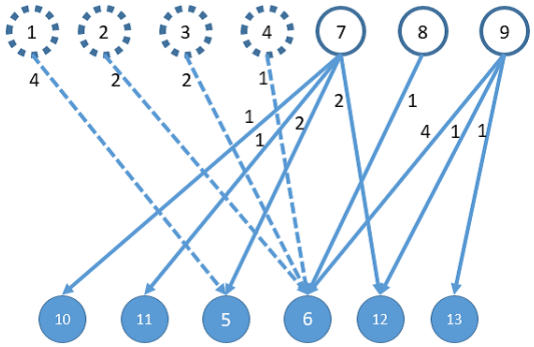

図 5 図 2, 図 4 の合成グラフ

付き有向エッジがその要素となる．図 2，図 4 のそれぞれの二部グラフにおいて，対応する次元が存在する場合はその同じ次元にエッジの重みを要素とするが，対応する次元が存在しない場合，要素は 0 となる．よって，図 2，図 4 それぞれにおけるベクトル表現は，式（1）,（2）となる．これらのベクトルを用いて，各プログラムごとによるクラスタリングを行うために行列を作成する．この行列に基づいて作成されるグラフは図 5 となる．

$$\boldsymbol{v}_{swap} = (4, 2, 2, 1, 0, 0, 0, 0, 0, 0, 0)^T \quad (1)$$

$$\boldsymbol{v}_{search} = (0, 0, 0, 0, 1, 1, 2, 2, 1, 4, 2)^T \quad (2)$$

3. 教科書のソートプログラム例に対するクラスタリング実験

本実験の目的は，特定の分野のアルゴリズムにおける教科書プログラムにおいてクラスタリングを行うことにより，プログラム構造の違いによって，差異が生まれるかを確認することである．今回の実験では Java 言語を利用したプログラムを用いる．使用するアルゴリズムは，プログラミング初学者向けの教科書に記載されているソートのプログラムを使用した．

実験に用いた教科書 [3] では，単一のアルゴリズムに複数のプログラム例が示されていることが特徴である．例えば，プログラムの構造に再帰構造を使用しているなどの実装方法に違いがあるため，今回の実験に利用した．表 2 は，使用したプログラムデータの一覧である．表中の bubbleSort1, bubbleSort2, bubbleSort3 は，全てバブルソートを実装した別構造を持つプログラム例である．

実験の手順について説明する．まず初めに，ソート機能を持つプログラムを収集した後，提案した二部グラフに変換する．その後，二部グラフからプログラムごとにベクトルを作成し，データ構造の操作メソッドで正規化を行った後，各プログラムごとに連結を行い，行列を作成する．この行列を用いて主成分分析を行った後，2 次元空間に図示し，プログラム間の類似性を確認する．

また，階層的クラスタリングによるデンドログラムを用いて，距離関数に基づく各教科書プログラムの類似性を確認する．距離関数は，Ward 法を利用した．

表 2 教科書プログラムのデータ一覧

id	プログラム	プログラムの実装における特徴
0	bubbleSort1	比較回数を抑えた機能を持つ実装
1	bubbleSort2	走査範囲を限定した実装
2	bubbleSort3	標準的な実装
3	fSort	標準的な実装
4	heapSort	標準的な実装
5	insertionSort	標準的な実装
6	mergeSort	標準的な実装
7	quickSort1	再帰構造による実装
8	quickSort2	非再帰構造による実装
9	qucikSort3	標準的な実装
10	selectionSort	標準的な実装
11	shellSort1	特定のデータ列に対応した実装
12	shellSort2	標準的な実装

3.1 Principal Component Analysis

主成分分析 (Principal Component Analysis:PCA) とは統計的解析手法の一つで，データの相関の情報に基づいて多くの変数が持つ，データの情報損失を抑えながら少ない変数に集約して分析する方法である．例として変数 a,b,c の 3 変数における PCA を考える．まず各変数の相関係数行列を作成する．式（3）中の r_{ab} は変数 a と変数 b における相関係数を示している．

各相関係数を行列として表したものを相関係数行列と呼ぶ．この際，同じ変数同士の相関係数は 1 となる．式 3 に a，b，c の三つの変数の相関係数行列を記す．

$$R = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} 1 & r_{ab} & r_{ac} \\ r_{ab} & 1 & r_{bc} \\ r_{ac} & r_{bc} & 1 \end{pmatrix} \end{array} \quad (3)$$

こうして求めた相関係数行列より，固有値，固有ベクトルを求める．相関係数行列 R に対して以下の式（4）を満たす数 λ を固有値，p を固有ベクトルと呼ぶ．

$$Rp = \lambda p \quad (4)$$

求めた固有値 λ を使い，式（5）を作成し，固有ベクトルを求める．p_i は対応する固有ベクトルの要素を表す．

$$\begin{pmatrix} \lambda - 1 & r_{12} & r_{13} \\ r_{21} & \lambda - 1 & r_{23} \\ r_{31} & r_{32} & \lambda - 1 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \lambda \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \quad (5)$$

本稿では PCA で求めた固有ベクトルをもとに解析を行う．

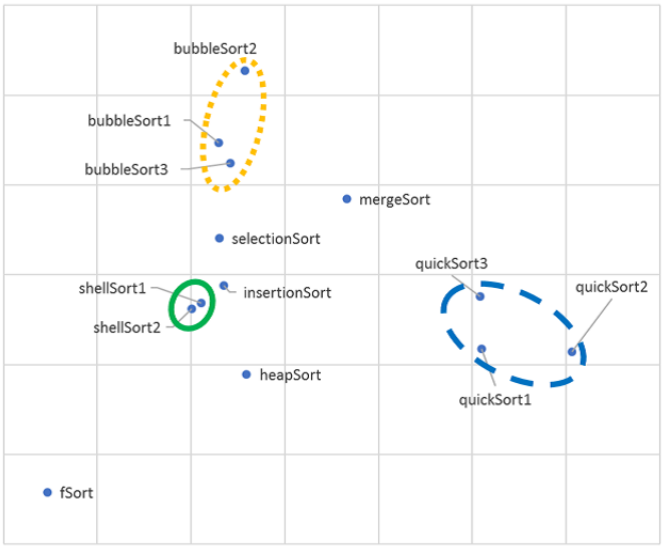

図 6　教科書プログラムにおける主成分分析の結果

3.2　Ward 法

Ward 法とは，特定クラスター内に存在するデータの平方和が最小となるクラスタを併合することでデータの近似を表す統計的解析手法の一つである．クラスタ i における平方和 S_i は式（6）で求める．式（6）中，n は変数の数を表し，$\bar{x_k}$ は平均を表している．

$$S_i = \Sigma_{k=1}^{n}(x_{ik} - \bar{x_k})^2 \qquad (6)$$

式（6）を用いて，併合対象であるクラスタ分の平方和を求めて平方和距離の値をそれぞれ算出する．クラスタ i とクラスタ j の平方和距離は $S_{ij} = S_i + S_j$ となる．また併合後の平方和をそれぞれ算出する．その後，併合前と併合後の平方和距離の値が最も小さくなるクラスタを併合する．以上の手順を繰り返すことで，クラスタリングを行う手法である．

3.3　実験結果

図 6 は，主成分分析による 2 次元プロットの実験結果を示し，各点はソートのプログラムを示している．この図より，似たような名称のプログラム同士が，近い位置にプロットされていることがわかる．点線で囲ったところは bubbleSort，破線で囲ったところは quickSort，実線で囲ったところは shellSort が近くに位置している．

また，図 7 は階層的クラスタリングにおけるデンドログラムを示す．横軸は表 2 の id と対応した教科書プログラムを示し，縦軸は教科書プログラム同士の離れ具合を示している．すなわち下位の階層において近い位置にいる教科書プログラムほど，類似性が高いことを示している．この図より，7,8,9 番目は quickSort，0,1,2 番目は bubbleSort が近くに位置していることがわかる．故に図 6，7 より，プログラム構造を基に作成したグラフ構造が，類似性を持つことがわかる．

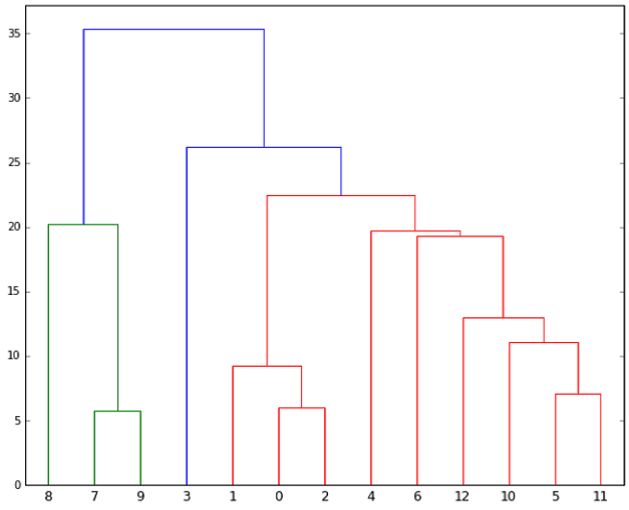

図 7　教科書プログラムにおける階層的クラスタリングの結果

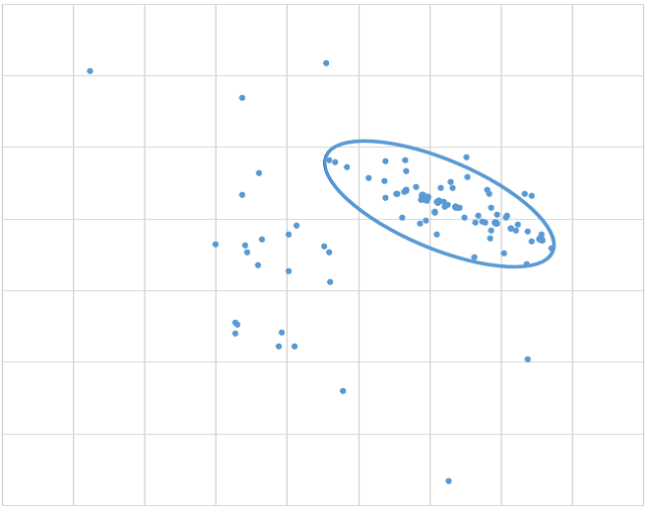

図 8　課題型プログラムにおける主成分分析の結果

4.　課題型プログラムに対するクラスタリング実験

次に，複数人によって同一の課題に対して実装されたプログラムのクラスタリングを行う．これにより，プログラムの構造を確認せずに，実装されている機能による部分構造の違いや，書き方の違いによって分類ができるかを確認する．課題内容は，5 つの数値データが与えられたときに，昇順に並び替えを行うものであり，プログラムデータの総数は 140 サンプル [2] 使用した．

図 8 に，主成分分析による実験結果を示す．同一課題を解くプログラムにおいて，実線で示したクラスタでは図 10 に示すバブルソートに似た構造を持つプログラムが集中していることを確認できた．

また，図 9 に階層的クラスタリングにおけるデンドログラムを示す．これにより，大きく 3 つの階層的なクラスタを確認することができる．プログラム構造を確認すると，

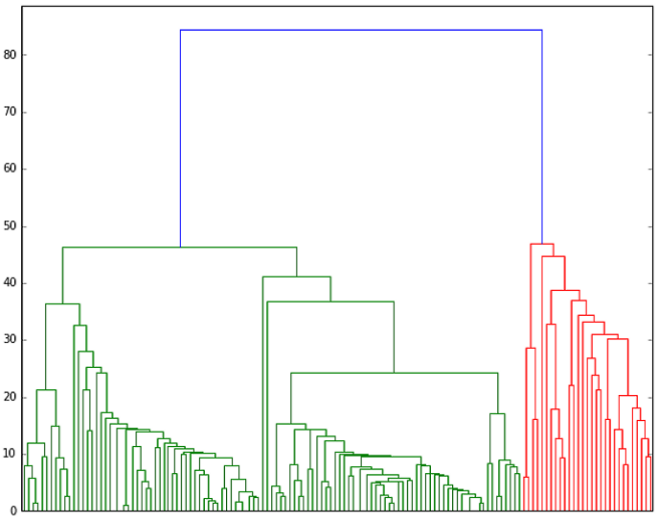

図 9　課題型プログラムにおける階層的クラスタリングの結果

```
1: Set numbers in the array x
2: for each of array index A
3:         for each of array index B
4:                 Set variable to x[A]
5:                 Set x[A] to x[B]
6:                 Set x[B] to variable
```

図 10　特徴的なプログラム構造

中心のクラスタにおいて，図 10 のようなバブルソートに似た構造がよく使用されていることが確認できた．よって同一の課題を解くプログラムにおいて，実装されているプログラム構造の違いによって分類されていることを確認した．

5. おわりに

本稿では，プログラムのソースコードをグラフを用いた抽象的表現に変換し，その抽象的表現間の類似性を数理的に扱うことを試みた．具体的には，ソートのアルゴリズムにおける教科書プログラムを用いてクラスタリングを行うことにより，プログラム構造の違いによってプログラムごとに分類ができることを確認した．また同一の課題に対して行われたプログラムのクラスタリングを行うことにより，プログラムの構造を確認しなくても実装されている機能による部分構造の違いや，書き方の違いによって分類されることを確認した．今後の課題としては，大規模なプログラムにおいても同様に分類することが可能であるかを検証したい．

謝辞　本研究の一部は科研費（基盤 (C) 課題番号:15K01100）の助成を受けて実施したものです．

参考文献

[1] 独立行政法人大学入試センター, センター試験用手順記述標準言語（DNCL）の説明, 東京 (2011)
[2] http://judge.u-aizu.ac.jp/onlinejudge/solution.jsp?pid=0018&lid=2
[3] 柴田望洋，明解 Java によるアルゴリズムとデータ構造，ソフトバンククリエイティブ (2007)
[4] Raymond, John W., Eleanor J. Gardiner, and Peter Willett. ”Rascal: Calculation of graph similarity using maximum common edge subgraphs.” The Computer Journal, Vol45, No.6 ,pp.631-644(2002)
[5] Yan, Xifeng, Philip S. Yu, and Jiawei Han. ”Graph indexing: a frequent structure-based approach.” Proceedings of the 2004 ACM SIGMOD international conference on Management of data, ACM(2004)
[6] Yan, Xifeng, Philip S. Yu, and Jiawei Han. ”Substructure similarity search in graph databases.” Proceedings of the 2005 ACM SIGMOD international conference on Management of data, ACM(2005)
[7] JDT-AST-Parser, Eclipse Java development tools (JDT), http://www.eclipse.org/jdt/