プログラム間の類似性の定量化手法

小田悠介*　　若林茂**

Method of Similarity Quantification between Program Codes

Yusuke ODA*　　Shigeru WAKABAYASHI**

ABSTRACT

We propose a new algorithm that calculates similarity value between two computer programs. Our method applies Kernel Method to syntax trees generated from each source code. This way has following characteristics: it is able to quantify degree of syntactic difference between source codes, and user can execute additional analysis based on linear algebras. This paper describes our algorithm first, and shows a result of applying it to students' program codes actually. In addition, we tryed categorizing students' programs using its result and Kernel-modified clustering algorithm (Kernel-AHC on the Centroids: K-AHC-C). This paper also discusses the comparison between results of K-AHC-C and human-power categorizing.

Keywords : Programming Education, Similarity between Program Codes, Kernel Method, Clustering

1. はじめに

プログラミング教育の現場では, 教師がプログラミングに関する適当な問題を学生に提示し, 学生がこの問題を解くプログラムを作成し提出する, という形式をとることが多い. この方法は簡便に学生へ自学自習の時間を与える方法で効果的だが, 一方で教師の負担は非常に大きく, 数十名の学生から送信されるプログラムを逐一解析し, 評価を付けなければならない.

ところで, 学生から送られてくるプログラムは多くの場合, 非常に似ている, もしくは完全に一致してしまうことがある. これは次のような仮定を置くことにより引き起こされるものと考えられる.

- 与えられる問題は全ての学生において同一である.
- 問題の規模が小さく, 作成されるソースコードが長大にならない.
- どの学生も同じ単元まで受講している.

例えば, 図1(a), (b)は, 2011年度の神戸市立工業高等専門学校(以降, 本文では神戸高専とする) 電子工学科2年の科目「プログラミングI」において, 学生から提出されたPascal言語のプログラムを2名分抜粋したものである.

ここで出題された問題は, 入力された数値列の合計と平均を求めるものである. 両者のプログラムは全く同じ処理を行う. これらの異なる点は, 識別子(変数名, 関数名, 型名など)の命名やインデント, 空白, 空行の有無であり, このような特徴を機械的な処理で除外してしまえば, 両者が同じ処理を行うプログラムであることは容易に判明すると考えられる.

```
program TotalMean(input,output);
var n,x,ten,gokei:integer;
    heikin:real;
begin
  gokei:=0;
  n:=1; x:=0;
  while ten>=0 do
    begin
      writeln('no.',n,' =?');
      gokei:=gokei+ten;
      x:=n-1; n:=n+1
    end;
  heikin:=gokei/x;
  writeln('gokei=',gokei:5);
  writeln('heikin=',heikin:5:1)
end.
```

(a)

```
program TotalMean(input,outpit);
var n,x,gokei:integer; heikin:real;
begin
  gokei := 0;
  n := 0;
  write('No.1=?');
 read(x);

  while x>=0 do
   begin
 gokei :=gokei + x;
n := n+1;
    write('No.',n+1,' =?');
    read(x);
  end;
  heikin :=gokei / n;
  writeln('合計 =',gokei:5);
  writeln('平均 =',heikin:5:1)
end.
```

(b)

図1 学生から提出されたプログラムのソースコード

このように, 小さな問題に対して学生が提出するプログラムは, そのほとんどが「よく似た」内容であり, 全てのプログラムを同じように精査するのは効率的ではない. 2つのプログラムがどの程度似ているか, つまり, プログラム間の類似性を機械的に抽出することができれば, 抽出された情報によって教師の作業量の軽減が期待できると考えられ, また評価自体の自動化なども視野に入れることができると考えられる.

プログラム間の類似性に関して, 井上らによる従来の研究(1)では, 2つのプログラムについて複数の比較法を順番に適用してゆき, プログラムが一致したときの比較レベルをプログラム間の距離とする手法(段階的感度比較法)が提案された. しかし, この手法で求めら

*専攻科　電気電子工学専攻
**電子工学科　教授

れる距離は間隔尺度としての要件を満たしておらず, そのままでは他のクラスタリングアルゴリズムへ応用することが難しいなどの問題があった.

そこで本研究では, 従来法のような問題が発生しない新しい類似性の尺度を提案した[2]. 従来法では特定の比較法による比較結果に注目していたが, 本手法で注目するのは, プログラムから生成できる構文木の形状である. これに, 様々なデータ構造を特徴空間に写像することなく解析することができるカーネル法を用いて, プログラム間の類似性を定義した.

本論文ではまず, 新しく提案したカーネル法に基づく類似度の定義を行い, 実際にこれを用いて学生のプログラム同士の類似度の算出を試みた. その後, カーネル法に適用できるよう変形したクラスタリングアルゴリズム, 特に今回は重心法を用いて, 学生のプログラムをクラスタに分けることを試みるとともに, 人間(プログラミングの授業を担当している教員)が手作業で行ったクラスタリング結果とも照らし合わせ, 本手法による類似度がどのような特徴を持つのかも調査した.

2. カーネル法

カーネル法はパターン認識などで用いられる手法で, データ構造同士の「内積」に相当するスカラ値を定義し, 内積から導かれる特徴空間上でデータの解析を行う. この内積をカーネルと呼ぶ. 特徴空間への写像に対して解析を行うのは通常のパターン認識と同じだが, カーネル法にはいくつかの特筆すべき点がある.

1. データ構造を選ばない.
   カーネルの値は, 内積の要件さえ満たしていればどのような計算を用いて導出しても構わない. このため, 実質的にどのようなデータ構造にもカーネル法を使用することができる.
2. 特徴空間の次元に制約がない.
   前述したように, カーネル法による特徴空間は内積により定義される. このため特徴ベクトルを厳密に求めるような方法をとることはなく, 特徴空間上の種々の値(距離, 角度など)を内積から直接計算して求める. このため, 特徴空間の次元を制限することによる制約を受けることはない.
3. 線型代数による解析が可能である.
   カーネルの値は線型代数の手法により定義されるので, 線型代数を用いる解析法への応用が可能である.

以降では特に断りがない限り, データ構造T_i, T_jに関するカーネルを$\kappa(T_i, T_j)$, T_i, T_jに対応する特徴空間上の仮想的なベクトルを$\boldsymbol{x}_i, \boldsymbol{x}_j$と書くこととする.

3. カーネル法のプログラムへの応用

ほとんどのプログラミング言語は, 記述されたプログラムの構造を構文木で表すことが可能である. 木構造に対するカーネルは既にいくつか提案されており, プログラムを構文木の形に変換すれば, 木構造カーネルを用いてデータ解析をすることが可能である. この様子を図2に示す.

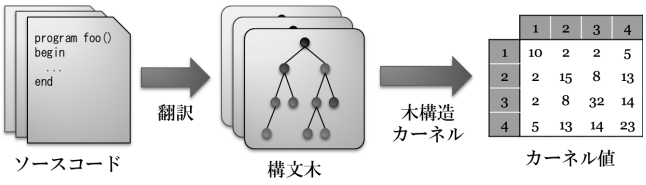

図2 プログラムへのカーネルの適用

4. 木構造カーネル

木構造カーネルに共通した特徴は, 2つの木の間に共通する構造がどの程度含まれているかを表していることである. これをプログラムの構文木に適用する場合, ある部分について同じ構造のプログラムが書かれていれば, 構文木の形状も共通部分を含むことになる. このため, 木構造カーネル上では似た構造を持つプログラムほどカーネルの値が大きくなると考えられる. 今回は木構造カーネルの中でも全部分木カーネル[3]と呼ばれるものを用いた.

まず, 次の条件を満たすものを(ある木に対する)部分木と定義する.

- 少なくとも1個以上の子を持つ任意のノードを選んで, このノード(部分木の根とする)と, 子孫ノードの組み合わせで得られる木である.
- 部分木の根以外のあるノードが木に含まれる場合, その兄弟ノードも全て木に含む.

2つの木構造T_i, T_jに対して, これらの部分木が一致する個数を数えたものを全部分木カーネル$\kappa_A(T_i, T_j)$と定義する. これは次式により表現される.

$$\kappa_A(T_i, T_j) \equiv \sum_{\nu_i \in T_i} \sum_{\nu_j \in T_j} \kappa_R(\tau(\nu_i), \tau(\nu_j)). \qquad (1)$$

ここで, ν_i, ν_jはT_i, T_jに含まれるノード, $\tau(\nu)$はνとその全ての子孫ノードで構成される木である. また$\kappa_R(T_i, T_j)$は共通ルート部分木カーネルと呼ばれるもので, 次の再帰式により定義される.

$$\kappa_R(T_i, T_j) \equiv \begin{cases} 0, & \text{if condition} \\ \prod_{n=1}^{\mathrm{d}(\mathrm{root}(T_i))} \kappa'_n(T_i, T_j), & \text{otherwise.} \end{cases} \qquad (2)$$

$$\begin{aligned} \text{condition} &\equiv \mathrm{d}(\mathrm{root}(T_i)) \neq \mathrm{d}(\mathrm{root}(T_j)) \\ &\quad \vee \ \mathrm{d}(\mathrm{root}(T_i)) = 0 \\ &\quad \vee \ \mathrm{label}(\mathrm{root}(T_i)) \neq \mathrm{label}(\mathrm{root}(T_j)) \\ \kappa'_n(T_i, T_j) &\equiv \kappa_R(\tau(\mathrm{ch}_n(\mathrm{root}(T_i)), \tau(\mathrm{ch}_n(\mathrm{root}(T_j))) + 1 \end{aligned}$$

ここで, $\mathrm{d}(\nu)$はノードνの出次数, $\mathrm{ch}_n(\nu)$はνのn番目の子, $\mathrm{root}(T)$は木Tの根, $\mathrm{label}(\nu)$はνのラベル(ノードに関連付けられたデータ) である. プログラムの構文木では, ラベルの値は識別子名, 制御構造名, 演算子といった情報を表す文字列や数値となる.

5. 余弦類似度

次にカーネルを用いて, 2つのデータ構造間の類似性を測る尺度を定義する. まず, カーネルとは特徴空間上の内積である, と定義されるため, 内積の定義より次の式が常に成り立つ.

$$\begin{aligned}\kappa(T_i, T_j) &= \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\ &= |\boldsymbol{x}_i||\boldsymbol{x}_j| \cos\theta_{T_i,T_j} \end{aligned} \tag{3}$$

ここで, θ_{T_i,T_j}は特徴空間上で$\boldsymbol{x}_i, \boldsymbol{x}_j$がなす角である. これを$\cos\theta_{T_i,T_j}$について書き直すと,

$$\begin{aligned}\cos\theta_{T_i,T_j} &= \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{|\boldsymbol{x}_i||\boldsymbol{x}_j|} \\ &= \frac{\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle}{\sqrt{\langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle \langle \boldsymbol{x}_j, \boldsymbol{x}_j \rangle}} \\ &= \frac{\kappa(T_i, T_j)}{\sqrt{\kappa(T_i, T_i)\kappa(T_j, T_j)}} \end{aligned} \tag{4}$$

となる. この$\cos\theta_{T_i,T_j}$は, $\boldsymbol{x}_i, \boldsymbol{x}_j$のなす角に関する情報を$[-1, 1]$もしくは$[0, 1]$の値域で与える. 後者の値域となるのは, 特徴ベクトルの全ての要素が正値のみを取る場合である. 全部分木カーネルは各要素について「ある部分木を含む場合は1, 含まない場合は0」という特徴空間に写像することに等しいので, 値域$[0, 1]$に該当する.

T_iとT_jが同じ部分木を多く持っている場合, $\boldsymbol{x}_i, \boldsymbol{x}_j$は特徴空間上で近い方向を向く, つまり$\theta_{T_i,T_j}$が0に近付く. よって, $\cos\theta_{T_i,T_j}$が1に近付けば近付くほどT_iとT_jが似た要素である, と言うことができる. 逆にT_iとT_jが全く同じ要素を持っていない場合, $\boldsymbol{x}_i, \boldsymbol{x}_j$は直交する. これは$\cos\theta_{T_i,T_j}$が0となる場合に対応する. この$\cos\theta_{T_i,T_j}$を余弦類似度と名付け, 特徴空間上の類似度を測る指標として用いることとする. ここで重要なのは, $\cos\theta_{T_i,T_j}$の値を特徴ベクトルを用いず, その内積(カーネル)のみを用いて計算できることである.

6. アルゴリズムの実装

図3に, 本研究で実装したアルゴリズムの概略を示す. 今回対象とした言語は, 神戸高専電子工学科2年の科目「プログラミングI」で用いられるPascal言語である. 言語仕様については同科目で用いられる教科書[4]に従った.

まず, 対象のPascalプログラムを前処理として翻訳器に入力し, 構文木を得る. これらの構文木の全てのペアに対して全部文木カーネル$\kappa_A(T_i, T_j)$を計算し, グラム行列Kを得る. グラム行列とはi, j要素が$\kappa(T_i, T_j)$となるような行列のことである. Kの要素を用いて, 余弦類似度$\cos\theta_{T_i,T_j}$を計算する.

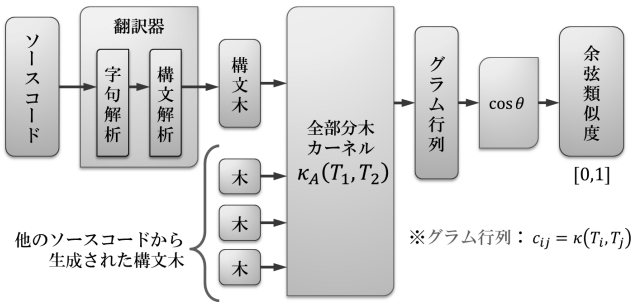

図3 アルゴリズムの内部構造

7. 余弦類似度の算出結果

実際に学生から収集したプログラムに対して, 本アルゴリズムを適用した. 収集したプログラムは神戸高専電子工学科2年の科目「プログラミングI」において出題された練習問題に対する回答であり, 1問に対する学生40人分の回答を1データセットとして, 合計12データセットを得た. 図4に示すのは, その中の1問,

> 1000以下の素数の一覧表を作成しなさい.
> ただし, 1行に表示する素数は10個とする.

に対するデータセットについて, 余弦類似度を計算した結果である.

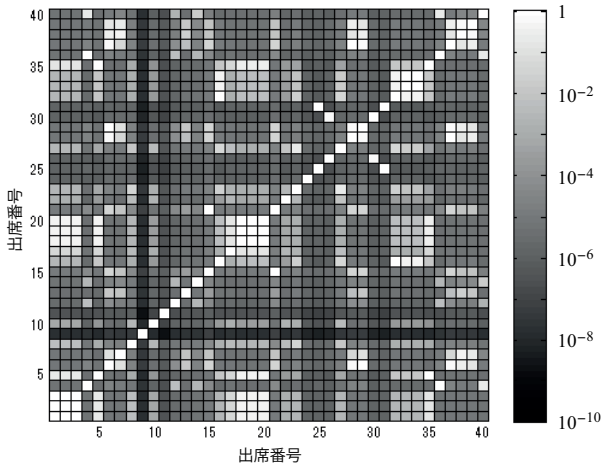

図4 学生のプログラム同士の余弦類似度

図4の縦軸, 横軸は学生の出席番号である. 出席番号が交差する点の色はそれらの学生が提出したプログラム同士の余弦類似度を示し, 色が明るいほど余弦類似度が大きい(1に近い)ことを示す.

図4から, いくつか読み取れる事実がある. 左下か

ら右上への対角線上のマスは全て白であるが, これは同じ学生のプログラム同士の余弦類似度が1であることを示しており, 自明な結果である. また一般的に$\kappa_A(T_i,T_j)=\kappa_A(T_j,T_i)$となる性質があるが, これは図が対角線について対称であることからも確認できる.

全部分木カーネルの定義から, 余弦類似度が1となるのは構文木が全く同一である場合に限られる(異なる形状の木には必ず異なる部分木が含まれるため). つまり, 交点のマスが白い学生同士, 例えば左下の出席番号1, 2, 3番の学生などは, 改行やインデント, 空白などの違いを除いて同じプログラムを提出している可能性が高い. ただし, 識別子名のみが異なる場合については, 今回の手法では検出されず, 類似度に影響は出ないことが分かっている. これは構文木の部分木の数や形状に, ノードのラベルとして扱われる識別子名が直接影響しないためである.

さらに図4からは, 「プログラムA, B, Cがあって, AとBの類似度が高く, BとCの類似度が高い場合, AとCの類似度も同じように高い」という規則性が見受けられる. つまり, 類似度の高いプログラム同士が一定のグループ性を持っていると推測できる. 学生を何らかのクラスタリング法を用いて並べ替えることで, より明確な情報が得られると考えられる.

8. 重心法

前章までに示した学生のプログラム間の類似性を詳しく調べるために, 重心法をカーネル法へ応用したカーネル重心法を用いて, 学生のプログラムのクラスタリングを行った. 重心法とは凝集型階層クラスタリング(Agglomerative Hierarchical Clustering: AHC)と呼ばれる手法の一種であり, 類似性の高いデータ群(クラスタ)2個を単一のクラスタにまとめる作業を繰り返すことにより, データ間の類似性を階層的な形で可視化することができる.

特徴空間上のベクトル$\boldsymbol{x}_i$, $\boldsymbol{x}_j$についてのユークリッド距離を$\mathrm{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j)$と書くこととする. ベクトル$N_\alpha$個が属するクラスタ$\alpha$と, N_β個が属するクラスタβがあるとき, それぞれのクラスタの重心ベクトル$\boldsymbol{g}_\alpha$, $\boldsymbol{g}_\beta$は

$$\boldsymbol{g}_\alpha = \frac{1}{N_\alpha}\sum_{\boldsymbol{x}_i\in\alpha}\boldsymbol{x}_i \tag{5}$$

$$\boldsymbol{g}_\beta = \frac{1}{N_\beta}\sum_{\boldsymbol{x}_j\in\beta}\boldsymbol{x}_j \tag{6}$$

であり, これら重心間の距離$\mathrm{dist}(\boldsymbol{g}_\alpha,\boldsymbol{g}_\beta)$は,

$$\mathrm{dist}(\boldsymbol{g}_\alpha,\boldsymbol{g}_\beta) = \mathrm{dist}\left(\frac{1}{N_\alpha}\sum_{\boldsymbol{x}_i\in\alpha}\boldsymbol{x}_i,\frac{1}{N_\beta}\sum_{\boldsymbol{x}_j\in\beta}\boldsymbol{x}_j\right) \tag{7}$$

で表される.

重心法のアルゴリズムは次の通りである.

1. 全ての特徴ベクトルを要素数1のクラスタと見なす. この時点では, クラスタの重心座標は特徴ベクトルと一致する.
2. 現在見えているクラスタの全ての組み合わせに対して重心間の距離$\mathrm{dist}(\boldsymbol{g}_i,\boldsymbol{g}_j)$を求める. ただし, i, jはクラスタを表す添字である.
3. $\mathrm{dist}(\boldsymbol{g}_i,\boldsymbol{g}_j)$の値が最も小さいクラスタの組を探し, これをまとめて新たなひとつのクラスタとする.
4. クラスタ数が1になるまで(2)〜(3)を繰り返す.

9. 重心法のカーネル法への適用

重心法はそのままでは特徴ベクトルの値を必要とするため, カーネル法へ適用することはできない. 特徴ベクトルを用いる計算はクラスタの重心間の距離$\mathrm{dist}(\boldsymbol{g}_i,\boldsymbol{g}_j)$のみであるので, この式を変形してカーネルのみを用いるようにする.

特徴空間上のベクトル$\boldsymbol{x}_i$, $\boldsymbol{x}_j$について, そのユークリッド距離$\mathrm{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j)$は余弦定理を用いて,

$$\mathrm{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \sqrt{|\boldsymbol{x}_i|^2+|\boldsymbol{x}_j|^2-2|\boldsymbol{x}_i||\boldsymbol{x}_j|\cos\theta} \tag{8}$$

で表される. ここで根号内の各項は$\boldsymbol{x}_i$, $\boldsymbol{x}_j$の内積で置き換えられるので,

$$\mathrm{dist}(\boldsymbol{x}_i,\boldsymbol{x}_j) = \sqrt{\langle\boldsymbol{x}_i,\boldsymbol{x}_i\rangle+\langle\boldsymbol{x}_j,\boldsymbol{x}_j\rangle-2\langle\boldsymbol{x}_i,\boldsymbol{x}_j\rangle} \tag{9}$$

となる. $\boldsymbol{x}_i$, $\boldsymbol{x}_j$としてクラスタα, βの重心$\boldsymbol{g}_\alpha$, $\boldsymbol{g}_\beta$を代入すると, 根号内の各項はそれぞれ,

$$\begin{aligned}\langle\boldsymbol{g}_\alpha,\boldsymbol{g}_\alpha\rangle &= \frac{1}{N_\alpha^2}\left\langle\sum_{\boldsymbol{x}_i\in\alpha}\boldsymbol{x}_i,\sum_{\boldsymbol{x}_j\in\alpha}\boldsymbol{x}_j\right\rangle\\ &= \frac{1}{N_\alpha^2}\sum_{\boldsymbol{x}_i\in\alpha}\sum_{\boldsymbol{x}_j\in\alpha}\langle\boldsymbol{x}_i,\boldsymbol{x}_j\rangle\\ &= \frac{1}{N_\alpha^2}\sum_{T_i\in T_\alpha}\sum_{T_j\in T_\alpha}\kappa(T_i,T_j)\end{aligned} \tag{10}$$

$$\begin{aligned}\langle\boldsymbol{g}_\beta,\boldsymbol{g}_\beta\rangle &= \frac{1}{N_\beta^2}\left\langle\sum_{\boldsymbol{x}_i\in\beta}\boldsymbol{x}_i,\sum_{\boldsymbol{x}_j\in\beta}\boldsymbol{x}_j\right\rangle\\ &= \frac{1}{N_\beta^2}\sum_{\boldsymbol{x}_i\in\beta}\sum_{\boldsymbol{x}_j\in\beta}\langle\boldsymbol{x}_i,\boldsymbol{x}_j\rangle\\ &= \frac{1}{N_\beta^2}\sum_{T_i\in T_\beta}\sum_{T_j\in T_\beta}\kappa(T_i,T_j)\end{aligned} \tag{11}$$

$$\begin{aligned}\langle\boldsymbol{g}_\alpha,\boldsymbol{g}_\beta\rangle &= \frac{1}{N_\alpha N_\beta}\left\langle\sum_{\boldsymbol{x}_i\in\alpha}\boldsymbol{x}_i,\sum_{\boldsymbol{x}_j\in\beta}\boldsymbol{x}_j\right\rangle\\ &= \frac{1}{N_\alpha N_\beta}\sum_{\boldsymbol{x}_i\in\alpha}\sum_{\boldsymbol{x}_j\in\beta}\langle\boldsymbol{x}_i,\boldsymbol{x}_j\rangle\\ &= \frac{1}{N_\alpha N_\beta}\sum_{T_i\in T_\alpha}\sum_{T_j\in T_\beta}\kappa(T_i,T_j)\end{aligned} \tag{12}$$

となる. ここでT_α,T_βは, クラスタα,βに属する特徴ベクトルに対応するデータ構造の集合である. これらを

距離の定義に代入すると, 最終的に,

$$\mathrm{dist}(\boldsymbol{g}_\alpha, \boldsymbol{g}_\beta) = \sqrt{\begin{aligned} &\frac{1}{N_\alpha^2}\sum_{T_i\in T_\alpha}\sum_{T_j\in T_\alpha}\kappa(T_i,T_j) \\ &+\frac{1}{N_\beta^2}\sum_{T_i\in T_\beta}\sum_{T_j\in T_\beta}\kappa(T_i,T_j) \\ &-\frac{2}{N_\alpha N_\beta}\sum_{T_i\in T_\alpha}\sum_{T_j\in T_\beta}\kappa(T_i,T_j) \end{aligned}} \quad (13)$$

が得られる. これによって, カーネルによる特徴空間上のベクトル間の距離を求めることができる. この距離の定義を用いて行う重心法をカーネル重心法(Kernel-AHC on the Centroids: K-AHC-C)と呼ぶ. 上記の式については, 鶴田による研究[5]でも同様のものが紹介されている.

10. カーネル重心法の適用結果

カーネル重心法により生成されたクラスタの順に, 図4を並べ替えたものを図5に示す.

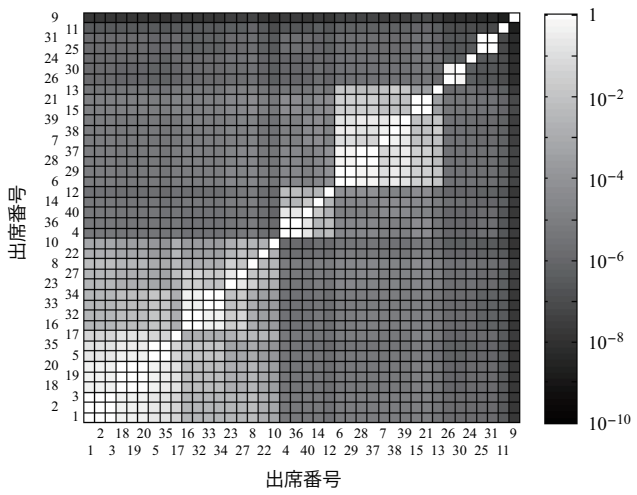

図5 カーネル重心法による並べ替え後

この図を見ると, 対角線上に3つの大きな明るい矩形を見て取れる. これは, 学生のプログラムがおよそ3つのグループに大別できることを示唆している.

11. 教員による分類結果との比較

図5の結果が実際に有用なものであるかどうかを確かめる必要があるため, 人間の手作業による分類結果との比較を行った. 表1は, 神戸高専電子工学科のプログラミング系科目を担当する教員に依頼して得た図5と同じデータセットの分類結果に, 図5に見られたクラスタを合わせて表示したものである.

基準1〜4は教員側が独自に考案した比較基準であり, それぞれ明確にプログラムを分類することができるものである. 具体的には表2のような分類法であった. 基準1はプログラムの計算量に基づく分類, 基準2〜4はプログラムの全体的な構成による分類である. 備考については, 基準とは別にプログラムに何らかの特徴が

表1 教員による分類結果と図5の比較

	基準					
出席番号	1	2	3	4	備考	図5でのクラスタ
1	C	A	A	A	—	X
2	C	A	A	A	—	X
3	C	A	A	A	—	X
18	C	A	A	A	—	X
19	C	A	A	A	—	X
20	C	A	A	A	—	X
5	C	A	A	A	—	X
35	C	A	A	A	—	X
17	A	A	A	A	—	X
16	C	A	A	A	—	X
32	C	A	A	A	—	X
33	C	A	A	A	—	X
34	C	A	A	A	—	X
23	B	A	A	A	—	X
27	A	A	A	A	—	X
8	B	A	A	A	—	X
22	A	A	A	A	—	X
10	A	A	A	A	—	X
4	A	B	A	B	—	Y
36	A	B	A	B	—	Y
40	C	B	A	B	—	Y
14	C	B	A	B	—	Y
12	A	B	A	B	—	Y
6	A	A	A	B	—	Z
29	A	A	A	B	—	Z
28	A	A	A	B	—	Z
37	A	A	A	B	—	Z
7	C	A	A	B	—	Z
38	C	A	A	B	—	Z
39	C	A	A	B	—	Z
15	A	A	A	B	—	Z
21	A	A	A	B	—	Z
13	C	A	A	B	—	Z
26	C	A	A	A	あり	—
30	C	A	A	A	あり	—
24	C	A	A	A	あり	—
25	A	B	B	A	あり	—
31	A	B	B	A	あり	—
11	C	A	A	A	あり	—
9	A	B	B	A	あり	—

表2 教員による比較基準

基準	概要
1	素数判定の繰り返し回数が, A: $n-1$, B: $\frac{n}{2}$, C: $\sqrt{n}$
2	A: 今までに出力した総数を10で割った余りで現在の行に出力した個数を計算 B: 現在の行に出力した個数を変数で保持し, 10になる度に0へ戻す
3	A: 素数の出力にwrite(改行なし)とwriteln(改行あり)を場合によって選択 B: 素数の出力にはwriteを使い, 改行のみ別に出力
4	冗長な(ひとつにまとめることができる)if文が, A: ある, B: ない

見られた場合に付記されたようである. また, 表全体は図5の左から右の順となるように出席番号を並べ替えている. 表の最も右の列は図5に見られた3つのクラスタを示しており, それぞれ図5の左下から最初の18名をクラスタX, 次の5名をクラスタY, 続く10名をクラスタZとした.

表1を見ると, 備考を付記された学生のプログラムについては図5のどのクラスタにも属していないことが分かる. つまり, 他人と明らかに異なるプログラムは本手法で明確に区別することができると考えられる. また基準2と基準4に注目すると, 両方がAに分類されているプログラム, 基準2がAで基準4がBに分類されているプログラム, 両方がBに分類されているプログラムが, それぞれ図5におけるX, Y, Zの各クラスタに対応していることが分かる. これより, プログラムの全体的な構造に着目している分類法には本手法が比較的よく一致した結果を出していると考えられる.

一方で基準1に関しては, 図5におけるそれぞれのクラスタに分散してしまっている. これは, 基準1による分類先を決定する部分がプログラム中の1ヶ所しかなかった(素数判定時のループ条件のみ)ため, 共通な部分木の数にはあまり影響が出なかったからであると推測できる.

12. まとめ

本研究では, プログラム間の類似性を定量化するために, 新たにカーネル法に基づく類似度を提案した. その結果,

- 本手法では, 数学的な手続きを用いて後段の解析を追加できることを示した. たとえば本論文ではカーネル重心法によるクラスタリングを行い, 学生のプログラム間に見られるクラスタを観察することができた.
- 全部分木カーネルによる解析結果が, 教員の手作業による分類結果のうち, プログラムの全体的な構造に着目したものとよく一致することが確認できた. 一方, プログラムの特定の部分のみに着目した分類結果に関しては, 今回の手法ではあまり合致しないことが分かった.

という事実が得られた.

これらを踏まえて, 今後取り組むべき課題としては,

- 単純な構文木の使用では明確になりにくいと分かった差異, たとえば制御構造の条件判定の違いなどについて, より明確にクラスタリングできるようにデータ構造を変形すること
- 現状では考慮されていない識別子の差異などを組み込む方法の開発
- 手法のより厳密な評価

が挙げられる.

参考文献

(1) 井上晴喜, 若林茂：プログラム間の類似性に関する研究, 教育システム情報学会第31回全国大会講演論文集, 教育システム情報学会, p219-220 (2006)

(2) 小田悠介, 上村康輔, 若林茂：プログラム間の類似性の定量化手法, 教育システム情報学会第37回全国大会講演論文集, 教育システム情報学会, pp.178-179 (2012)

(3) John Shawe-Taylor, Nello Cristianini, 大北剛 訳：Kernel Methods for Pattern Analysis, 共立出版, p470-482 (2010)

(4) 真野芳久：Pascalプログラミングの基礎, サイエンス社, p.234-254 (1994)

(5) 鶴田育緒：An application of kernels for clustering, 筑波大学大学院システム情報工学研究科修士論文, p.20 (2006)
URL: http://www.sk.tsukuba.ac.jp/SSE/degree/2005/thesis/200105351.pdf (2012年9月6日閲覧)