特別研究報告

題目

構造的特徴を考慮したプログラミングコンテストの解答の分類手法

指導教員
井上 克郎 教授

報告者
原口 公輔

平成 31 年 2 月 12 日

大阪大学 基礎工学部 情報科学科

平成 30 年度 特別研究報告

構造的特徴を考慮したプログラミングコンテストの解答の分類手法

原口 公輔

平成 30 年度 特別研究報告

構造的特徴を考慮したプログラミングコンテストの解答の分類手法

原口 公輔

内容梗概

現在，アルゴリズムの学習やプログラミング技術の向上のために，初級者・上級者を含む多くのユーザがプログラミングコンテストに参加している．プログラミングコンテストとは，アルゴリズムに関する問題を提示し，参加者に制限時間以内に解かせ，解くのに要した時間やその解答の精度などでスコアを競うコンテストである．解答ソースコードの正誤判定にはオンラインジャッジシステムが用いられ，用意されたテストケースと実行時間などの制約条件を全てクリアすることで，正しい解答と判定される．プログラミングコンテストは Web サービスとして提供されることが多く，過去の解答が公開されていることがある．プログラミング初級者はこの解答を参考に自身の解答改善やアルゴリズム学習に活用することが出来る．ところが，ユーザによって使用言語や実装の方法が異なるため，解答の一覧全てが学習に活用できるとは言えない．

プログラミング初級者の学習支援を目的とした研究に，藤原らのユーザの既存解答群の分類・提示システムの開発研究がある [1]．このシステムはソースコードの語彙的な特徴と実行時間や使用メモリ，ソースコードのサイズといったメトリクス値を用いて，過去の全解答を固定数のクラスタに分類する．その後，提出時間や正誤判定などの情報を用いて，クラスタを頂点とする遷移グラフを作成し，ユーザの解答に最も類似するクラスタの解答から漸次的に提示するシステムである．提示される解答を参考にすることで，ユーザは自身の解答を漸進的に改善することができる．

ところが，この研究の分類手法では既存の解答群の分類に際して，行数や循環的複雑度といった構造的な特徴が反映されていない．プログラミングコンテストの解答は使用しているアルゴリズムや関数，ユーザの癖などによって構造的な特徴が異なり，学習者が書くソースコードと似た構造を持つ解答が学習の参考にしやすい．

そこで，本研究ではソースコードの分類に際して，ソースコードの構造的特徴を表すメトリクスも用いて，分類を実行した．先行研究と分類結果を比較し，構造的なメトリクス値が同問題を解くソースコード分類に対して有用であることが分かった．

主な用語

プログラミングコンテスト
N-gram IDF
ソースコードメトリクス

目 次

1　まえがき

近年，プログラミングコンテストがプログラミング学習者の関心を集めている．プログラミングコンテストは，アルゴリズムに関する複数の問題を参加者が一斉に解き，ソースコードの提出の早さや，解答の正確さを競う競技であり，Web サービスとして提供されることが多い．

プログラミングコンテストでは，過去に提出されたソースコードがリポジトリに保存されており，ユーザは自由に閲覧することができる．プログラミングコンテストのユーザはこれらのソースコードを参考にすることで，アルゴリズムの学習や，自身のソースコード改善に活用する事ができる．ところが，同一の問題に対するソースコードの実装方法は使用言語や使用するライブラリ，アルゴリズムによって多様である．さらに，プログラミングコンテストの解答は，使用言語や判定結果，提出時間などのソースコード以外のメタ情報でしかソースコードを検索できない．そのため，ユーザは学習の参考になるソースコードを効率的に検索することが困難である．

この課題を解決するために藤原らは，プログラミング学習者向けの解答提示システム TAMBA[1] を開発した．この研究では，プログラミングコンテストのソースコードリポジトリを語彙的な特徴で複数のクラスタに分類し，各クラスタごとに，テストの実行時間やソースコードのサイズなどのメタ情報で更に細分化し，複数のクラスタを生成する．その後，提出履歴などの情報で各クラスタを頂点とする有向グラフを作成する．ユーザがシステムに対してソースコードを新たに入力すると，それに類似するソースコードが含まれるクラスタと，隣接するクラスタに含まれる全ソースコードを提示する．システムのユーザには性能面で最も優れたソースコードが初回から提示されるわけではなく，自身のソースコードに語彙的かつ性能的に類似した解答だけが提示されるので，段階的に改善できる．

ところがこの研究では，ソースコードの分類に際して構造的な特徴を考慮していない．プログラミングコンテストの解答は使用しているアルゴリズムや関数，ユーザの癖などによって構造的な特徴が異なる [2]．そのため，学習者が書くソースコードと似た構造を持つ解答が学習の参考にしやすい．そこで，本研究では，語彙的な特徴に加えて，ソースコードの構造的特徴を表すメトリクスを用いた「構造的な特徴」を考慮したプログラミングコンテストの解答の分類を試みた．さらに，TAMBA との分類の違いを評価した．

以降，2 章ではプログラミングコンテストに関する関連研究と関連技術について述べ，3 章では提案手法について説明する．4 章では提案手法による解答群の分類を評価する実験方法とその結果を示し，5 章で全体のまとめと今後の課題について議論する．

2　関連研究

本章では，プログラミングコンテストのソースコードの分析とそれに関連する研究について述べる．

2.1　プログラミングコンテスト 初級者・ 上級者間における ソースコード特徴量の比較

堤らの研究 [2] では，プログラミングコンテストの一種である Codeforces[3] の提出済みのソースコードを利用して，プログラミングコンテストにおける初級者と上級者間の，予約語の使用頻度，ソースコードメトリクスの値，編集履歴の差異を調査した．ここで，プログラミングコンテストにおける初級者と上級者とは，それぞれ，プログラミングコンテストの各コンテストの結果の累積によって定まる「レーティング」によって全ユーザをソートしたときの下位 25%と上位 25%のユーザを指す．

2.1.1　ソースコードメトリクス

堤らは Source Monitor[4] と呼ばれる Web 上で公開されているメトリクス計測ツールを用いて計測出来る 11 種類のメトリクスについて調査した．調査対象であるメトリクスの名称とその説明を表 1 に示す．また，メトリクスの一部で用いられている循環的複雑度 [5] についての説明を，下記に示す．

表 1: 調査対象のメトリクス一覧 [2]

メトリクス	説明
avg_complexity	各関数の循環的複雑度の平均値
max_complexity	各関数の循環的複雑度の最大値
avg_depth	各関数のネストの深さの平均値
max_depth	各関数のネストの深さの最大値
methods_per_class	クラス当たりのメソッド数
n_class	クラス数
n_func	関数の数
n_lines	物理行数
n_statements	セミコロンで区切られた論理行数
percent_branch_statements	全体の論理行数に占める分岐文 (if, else, for, while, goto, break, continue, switch, case, default, return を用いた文) の割合
percent_comments	全体の物理行数に占めるコメント の割合

循環的複雑度

McCabe によって開発されたソフトウェア測定法の一種である．循環的複雑度は以下の式によって定義される．

$$M = E - N + 2P$$

ただし，

M := 循環的複雑度

E := 制御フローグラフ ($ControlFlowGraph : CFG$) における辺の数

N := CFG における頂点数

P := CFG におけるコンポーネント (結合しているグラフ) の数

循環的複雑度はソースコード内の分岐の数などによって増減する．この値が低いほど，メソッドの構造が単純であり，バグ混入の可能性が低いと言える．

2.1.2 分布の調査方法

堤らはソースコードのメトリクス値を調査し，比較を行った．以下の用語を用いて説明する．

U_p := 問題 p へ提出を行った参加者の集合

$m_{u,p,c}$:= 参加者 u，問題 p における初回提出ソースコードの各メトリクス c の計測値

$m'_{u,p,c}$:= $m_{u,p,c}$を各問題について平均 0，標準偏差 1 となるように標準化したもの

$M'_{c,low}$:= $m'_{u,p,c}$のうち，u が初級者に属する集合

$M'_{c,high}$:= $m'_{u,p,c}$のうち，u が上級者に属する集合

メトリクス値の調査は以下の流れで行われた．

1. 各参加者，各問題の初回提出について Source Monitor を用いて，$m_{u,p,c}$ を求める．
2. $m_{u,p,c}$ をもとに，$m'_{u,p,c}$ を計算する．
3. $m'_{u,p,c}$ をメトリクス c ごとに初級者の集合 $M'_{c,low}$ と上級者の集合 $M'_{c,high}$ に分割する．
4. “$M'_{c,low}$ と $M'_{c,high}$ の母平均が等しい” という帰無仮説を立てて Welth の t 検定を実施する．

5. $M'_{c,low}$ と $M'_{c,high}$ の差を明らかにするために，Cohen's d[6] を用いて，効果量を測定する．

Cohen's d[6]

Cohen's d は x_1 と x_2 の 2 群において，各群の標準偏差に対してどの程度平均が異なるかを示す．Cohen's d 1 は以下の式によって定義される．

$$d = \frac{\overline{x_1} - \overline{x_2}}{s}$$

ただし，

$$\begin{aligned} s &:= \sqrt{\frac{(n_1-1)s_1^2 + (n_2-1)s_2^2}{n_1+n_2-2}} \\ n_i &:= |x_i| \\ s_i^2 &:= x_i\text{の分散} \end{aligned}$$

各 c ごとに，$M'_{c,low}$，$M'_{c,high}$ の 2 群について，t 検定の実施と，Cohen's d の算出を行った．調査の結果，いずれのメトリクスについても，t 検定において有意水準 5%で有意差が認められた．

初級者と比較して，上級者の計測値が大きかったメトリクスについて，Cohen's d の値の降順に並べたものを表 2 に示す．n_statements や n_func，methods_per_class といったメトリクスが列挙されており，上級者は処理を関数やクラスなどに分割する傾向にあることが分かる．次に，上級者と比較して，初級者の計測値が大きかったメトリクスについて，Cohen's d の昇順に並べたものを表 3 に示す．avg_depth や percen_branch_statement，avg_complexity などのメトリクスが列挙されており，初級者は分岐分を多く利用し，ネストが深くなる傾向にあることが分かる．

このように堤らは，プログラミングコンテストのソースコードの特徴として，上級者は初級者と比較して，処理を関数やクラスといったブロックに分割し，複雑度が小さい傾向にあることを示した．

2.2 解答提示システム TAMBA

藤原らはプログラミング学習者向け漸進的ソースコード提示システム TAMBA[1] を開発した．このシステムの全体像を図 1 に引用して示す．TAMBA は，各問題の提出済みの解答群を対象に，N-gram IDF を用いて各ソースコードを語彙的特徴で固定数のクラスタに分類する．次に各クラスタ毎に実行時間や使用メモリサイズ，ソースコードサイズといったメトリクスを用いて 再度細かく分類する．最後に，提出履歴の情報を用いて，各クラスタを頂

表 2: 上級者において値が大きいメトリクス

c	$M'_{c,low}$	$M'_{c,high}$	初級者分散	上級者分散	p 値	Cohen's d
n_statements	-0.139	0.108	0.894	1.078	0.000e+00	0.245
n_func	-0.105	0.089	0.843	1.118	0.000e+00	0.191
n_lines	-0.065	0.043	0.931	1.022	3.037e-220	0.109
methods_per_class	-0.045	0.035	0.567	1.134	1.660e-165	0.084
n_class	-0.044	0.023	0.768	1.086	8.730e-98	0.069

表 3: 初級者において値が大きいメトリクス

c	$M'_{c,low}$	$M'_{c,high}$	初級者分散	上級者分散	p 値	Cohen's d
avg_depth	0.246	-0.202	1.034	0.944	0.000e+00	-0.457
percent_branch_statement	0.195	-0.154	0.993	0.995	0.000e+00	-0.351
avg_complexity	0.162	-0.137	1.073	0.926	0.000e+00	-0.303
max_complexity	0.113	-0.100	1.060	0.953	0.000e+00	-0.214
max_depth	0.096	-0.081	1.028	0.971	0.000e+00	-0.179
percent_comments	0.022	-0.035	1.031	0.916	8.580e-61	-0.059

点とする有向グラフを作成する．この有向グラフがシステムの根幹になる．このシステムの入力は，同じ問題に対するユーザのソースコードである．TAMBA は入力したソースコードと最も特徴が近いソースコードが含まれているクラスタと，有向グラフ上でそのクラスタと隣接しているクラスタを特定する．それらのクラスタに含まれている全てのソースコードを類似したソースコード として提示する．最も性能が良いと考えられる解答ソースコードを初回から提示するのではなく，他のユーザの提出履歴に沿って解答ソースコードを提示することで，ユーザは漸進的にソースコードの改善を行えると考えられる．

N-gram IDF と有向グラフ作成までのプロセスについて，次小節より説明する．

2.2.1 N-gram IDF

文書の大域的な語の重みつけ技術である IDF(逆文書頻度) は，式の簡潔さとロバスト性から，多くの代表的な語の重み付け手法に利用されている．IDF の代表的な応用例に TF-IDF が挙げられる．これは，文章中に含まれる各単語の重要度を評価する一つの手法であり，文書 d における単語 t の TF-IDF 値は下記式で表される．

$$TF - IDF(t, d) = tf(t, d) \cdot idf(t) \tag{1}$$

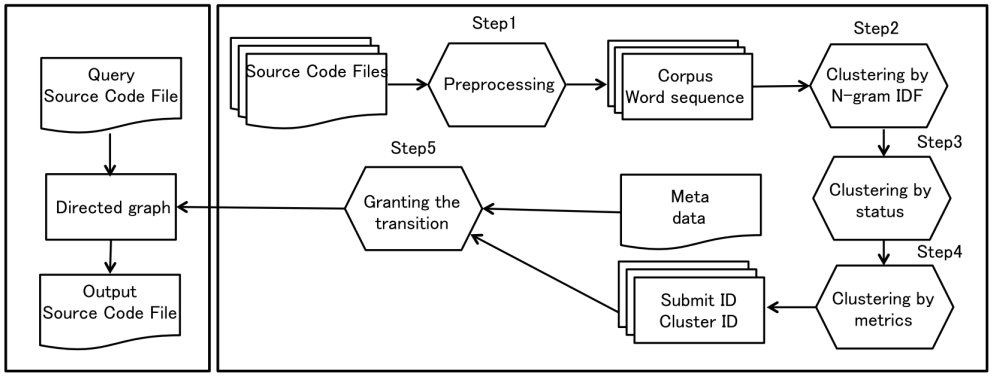

図 1: TAMBA の全体像

上記式において，$tf(t,d)$ はその文書における 1 つの単語の出現頻度を表す値であり，下記の式で表される．

$$tf(t,d) = \frac{n_{t,d}}{\sum_d n} \tag{2}$$

ここで，$n_{t,d}$ は文書 d における単語 t の出現回数，$\sum_d n$ は文書 d における全ての単語の出現回数である．

また，$idf(t)$ は対象とする文書に関わらず一定の値を取り，文書の全集合における，単語 t の重要度を示す指標になり，下記式で表される．

$$idf(d) = \log \frac{|D|}{df(t)} \tag{3}$$

ここで，$|D|$ は文書の全集合 D の要素数を，$df(t)$ は単語 t の現れる文書の数を表す．単語 t が現れる文書の数が多いほど，$idf(t)$ の値は小さくなり，$TF-IDF(t,d)$ の値によって示される単語重要度も小さくなる．

IDF の欠点として，N-gram が適用できない点が挙げられる．不自然な単語のつながりをもつ単語 N-gram の IDF は各文書における出現頻度が低くなるので，IDF が大きくなる．この課題に対して，白川ら [7] によって単語 N-gram に対する大域的重み付け手法が提案され，単語 N-gram に対する IDF，IDF_{N-gram} の算出が可能になった．単語 N-gram g に対する $IDF_{N-gram}(g)$ は以下の式より表される．

$$IDF_{N-gram}(g) = \log \frac{|D| \cdot df(g)}{df(\theta(g))^2} \tag{4}$$

ここで，$|D|$ は文書の全集合 D の要素数，$df(g)$ は単語 N-gram の現れる文書の数，$\theta(g)$ は単語 N-gram $g = w_1 \cdots w_N$ から単語の論理積 $\bigwedge_N^{i=1} w_i$ に分解する写像関数，$df(\theta(g))$ は D のうち，$\theta(g)$ を満たす文書の数である．

2.2.2 TAMBA のソースコード分類手法

藤原らによるソースコードの分類手法は下記に示すステップに分かれる．各ステップについて説明する．

1. ソースコードの前処理
2. N-gram IDF による解答ソースコードの分類
3. 正誤判定による細かい分類
4. ソースコードのメタ情報による細かい分類

ステップ 1　最初のステップでは，ソースコードの特徴量作成のための前処理を行う．以下に示す処理を全ソースコードに対して行う．

- アルファベットに含まれる大文字を全て小文字に変換する
- 記号文字を空白文字に変換する
- 連続する 2 つ以上の空白文字を 1 つの空白文字に変換する

上記の前処理により，ソースコードを単語列として扱えるようになる．

ステップ 2　このステップでは，ステップ 1 で前処理を施したソースコードの集合に対して，N-gram IDF 値を用いた特徴的な単語 N-gram の抽出と，その出現回数を用いた特徴ベクトルの作成を行う．

白川らによる研究 [7] によって，単語 N-gram の重みのみを用いて，全文書における特徴的な単語 N-gram の抽出が可能になり，これを実行するツール Weighting Scheme[8] が公開されている．藤原らによる研究では文書全体の特徴的な単語 N-gram の抽出に Weighting Scheme を使用している．Weighting Sheme は，特徴的な N-garm を抽出したいファイル群が含まれるディレクトリを引数に実行することで，ディレクトリ全体における特徴的な N-gram の出力を行う．

ステップ 2 ではこの Weightinh Scheme を用いて語彙的な特徴を表すベクトル (以下，語彙ベクトルと表記する) を作成する．ステップ 1 で前処理を行なった全解答ソースコードを

Weighting Scheme に入力し，特徴的な N-gram の抽出を行う．その後，抽出された N-gram の出現回数を各成分，次元を抽出された N-gram の数とするベクトルを各解答ごとに作成し，これを語彙ベクトルとする．

次に，各語彙ベクトル間の類似度を求め，凝集型階層クラスタリングを実行する．凝集型階層クラスタリングとは，各クラスタが 1 つずつ特徴ベクトルを持つ状態から，再帰的にクラスタを併合していくクラスタリング手法である．ベクトル間の距離の定義及び，クラスタ併合時のメソッドは複数存在するが，TAMBA ではベクトル間の距離にはコサイン距離を，クラスタ併合時のメソッドは各クラスタ間の最も距離が遠いベクトルを採用する最長距離法を採用している．

ステップ 3　ステップ 2 で作成した各クラスタを，そのソースコードがコンテストにおける正誤情報を用いて 2 つのクラスタに再度分類する．

ステップ 4　ステップ 3 の処理の実行の後，正誤情報が正である全クラスタに対して，ソースコードのメタ情報を用いて，さらに複数のクラスタに細かくクラスタ分割を行う．コンテストから取得できる，ソースコードのファイルサイズ，テストの実行時間，メモリ使用量の 3 つのメタ情報を用いて各ソースコードの特徴ベクトルを作成する．さらに，それぞれの次元において，平均値が 0，分散が 1 になるように線形変換し，正規化する．この特徴ベクトルを用いてステップ 2 と同じ凝集型階層クラスタリングを行い，8 つの各クラスタを 3 つに再度分類する．

ステップ 5　ステップ 5 では，プログラミングコンテストの履歴情報を元に，有向グラフを作成する．作成する有向グラフは，各クラスタがノード，連続する 2 つの提出がどのクラスタからどのクラスタに遷移したかの情報がエッジとなる．

2.2.3　実験結果

TAMBA[1] では誤答の修正において，TAMBA を用いた場合に誤答の原因特定に役立つか，学習に役立つかの面で評価した．

評価実験では，問題文の内容理解の後に，2 つの誤答ソースコードの修正を行う．その際に，TAMBA の出力かプログラミングコンテストの解答一覧のどちらかを，原因特定の参考にする．対象の問題は，プログラミングコンテストの一種である AOJ[9] の，複数の実装方法があると判断した 3 つの問題である．その後，“プログラミングの学習に役立つと思いますか？”，“誤答原因の 特定に役立ちましたか？” と質問し，“役立つ” から “役立たない “までの 5 段階のリッカート尺度で回答してもらう．

実験の結果，どちらの質問に対しても，TAMBA を用いた場合のほうが，役立つを示す 1 と 2 と答えた割合が高く，誤答の原因を特定できた件数も TAMBA を用いた場合のほうが高かった．

3　提案手法

3.1　研究の背景と目的

前章で述べた [1] は，プログラミング学習者の支援を目的とした，ソースコードの提示を行うシステムである．ソースコードの全てが参考になるわけではないため，N-gram による語彙的な情報でソースコードを分類し，ユーザの実装に近いソースコードを提示する．また，堤らの研究 [2] によりユーザのレートによってメトリクス値の平均分布が異なり，レートごとにソースコードの構造にある傾向があることが示された．

ところで，TAMBA はソースコードの分類において，構造的な特徴が反映されていない．よりユーザの実装に近いソースコードを提示するために，ソースコードの分類に構造的な特徴を反映するべきである．よって本研究では，ソースコードメトリクス値を用いた，構造的な特徴を考慮したソースコードの分類手法を提案し，その分類結果を既存手法による分類結果と比較する．

3.2　提案手法の概要

本章では，構造的特徴を考慮したソースコードの分類手法を説明する．図 2 にその一連の流れを示す．各問題の提出されたソースコード群を対象に，下記の 3 ステップの処理を適用する．

なお，分類結果の傾向の比較を行う際に，誤答のソースコードのクラスターは変化がないため，分類対象のソースコードは正答のみとしている．

3.3　ステップ 1

最初のステップでは，ソースコードに前処理を施し，語彙ベクトルと構造的な特徴を表すベクトル (以下，構造ベクトルと呼ぶ) を作成する．

3.3.1　前処理

プリプロセッサと単記号の削除　プログラミングコンテストに提出されるソースコードの特徴として，過剰なプリプロセッサの存在が挙げられる．提出するソースコードの制限として，そのサイズは対象外であることが多い．そのため，不必要な include 文や define 文によるエイリアスを数十行含むソースコードが存在する．このプリプロセッサの有無はステップ 2 の N-gram IDF 値を用いた特徴語の抽出に大きな影響を及ぼすと考えられる．

また，先行研究では使用ツールの都合上，＋や％といった単記号を前処理で削除していた．単記号を削除してしまうと，識別できない式が存在する．例えば，C 言語における，ビット

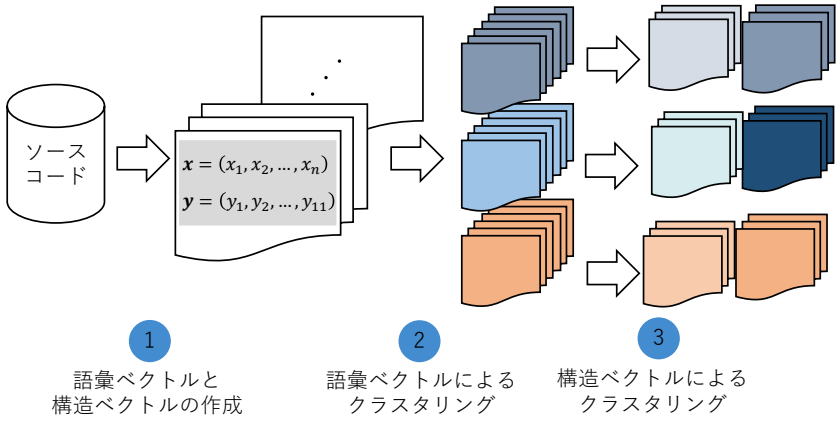

図 2: 提案手法の全体像

演算子&と条件式における論理演算子&&の違いなどである．単記号の有無も N-gram IDF 値を用いた特徴語の抽出において影響を及ぼすと考える．

本研究ではこれら 2 つの前処理による分類結果の違いを比較するために，次の 4 パターンで前処理を行う．

- プリプロセッサと単記号のどちらも削除する
- プリプロセッサを削除し，単記号を残す
- プリプロセッサを残し，単記号を削除する
- プリプロセッサと単記号のどちらも残す

連続した空白，改行の削除　ソースコードに含まれる連続した空白と改行を 1 つの空白に変換する．これは，N-gram IDF 値の計算に用いるツールの入力フォームに合わせるためであり，先行研究と同様である．

3.3.2　語彙ベクトルの作成

次に，前処理を施したソースコードの集合に対して，N-gram IDF 値を用いた特徴的な単語 N-gram の抽出と，その出現回数を用いた特徴ベクトルを作成する．この一連の処理は先行研究と同様である．ここで作成したベクトルが図 2 の $\boldsymbol{x}$ に該当する．

3.3.3　構造ベクトルの作成

先行研究では，ソースコードメトリクスではなく，テストの実行時間，使用メモリ，コードサイズの 3 成分を持つベクトルを作成していた．

本研究では，Source Monitor[4] によって得られたメトリクス値を各成分とする構造ベクトルを作成する．ここで作成したベクトルが図 2 のベクトル $\boldsymbol{y}$ に該当する．対象のメトリクスは表 1 に示す全てのメトリクスであり，作成するベクトル $\boldsymbol{y}$ の次元は 11 次元になる．

3.4　ステップ 2

ステップ 2 では，各語彙ベクトル間の類似度を求め，凝集型階層クラスタリングを実行する．ベクトル間の距離，クラスタの併合にはそれぞれコサイン距離，最長距離法を採用し，作成するクラスタの個数は TAMBA[1] と同じ 8 とした．

3.5　ステップ 3

ステップ 3 では，ステップ 2 で作成した各クラスタごとに構造ベクトル間の類似度を求め，凝集型階層クラスタリングを実行する．ベクトル間の距離，クラスタ併合にはそれぞれベクトル距離，最長距離法を採用し，作成するクラスタの個数は TAMBA[1] と同じ 3 とした．

以上の手順により，入力されたソースコード群は 24 個のクラスタに分類される．

4　ケーススタディ

前章で述べた提案手法について，その分類の傾向を調査するためにケーススタディを行なった．

4.1　評価方法

4.1.1　ソースコードの組み合わせの分布調査

問題の分類対象ソースコードにおいて，${}_n\mathrm{C}_2$(n はソースコードの数) 個の，2 つの異なるソースコード組み合わせの集合 D を考える．本研究の提案手法と TAMBA[1] の手法の分類の結果によって，D の各要素を表 4 に示す 5 タイプのいずれかに分類する．提案手の異なる 4 種類の前処理で調査を行う．それぞれのタイプの組み合わせをカウントし，前処理の比較を行う．

表 4: ソースコード組み合わせの種類

No.	組み合わせの種類
1	提案手法，TAMBA の手法ともに， 2 つのソースコードを同じ 1 つのクラスタに分類した
2	提案手法は同じクラスタに分類したが， TAMBA の手法は異なる 2 つのクラスタに分類した
3	提案手法は異なる 2 つのクラスタに分類したが， TAMBA の手法は同じ 1 つのクラスタに分類した
4	提案手法，Tmaba の手法ともに，異なる 2 つのクラスタに分類した

4.1.2　メトリクス値による箱ひげ図の作成

分類を実行した問題ごとに箱ひげ図を作成する．箱ひげ図の縦軸に採用するデータは，[2] で示されている cohen's d の値の上位 1 つと下位 1 つのメトリクスである．これは，上級者と初級者間で平均値の差異が大きかったメトリクスを箱ひげ図の軸にすることで，構造的な特徴を分類に反映できている場合に，箱の分布が大きく異なると考えられるからである．

箱ひげ図は，提案手法の分類で作成される 24 個のクラスタを横軸に取る．' ステップ 2 のクラスタ番号'-' ステップ 3 のクラス他番号' のようにして各クラスタを 1-1，1-2，...，8-3 というようにラベリングし，ステップ 2 で作成されるクラスタごとに色分けを行う．

作成する箱ひげ図の一覧を表 5 に示す．

表 5: 箱ひげ図作成に用いたメトリクスの種類

No.	メトリクス名	メトリクスの説明
1	n_statements	物理行数
2	avg_depth	平均的なネストの深さ

4.2 データセットの準備

本研究では，分類するソースコードデータに [2] のデータセットを用いた．これは，プログラミングコンテストの一種である Codeforces に提出されているソースコード及び，そのメタ情報を格納したデータベースによって構成されるデータセットである．データセットの統計情報を表 6 に示す．このデータセットの中から，特定の問題のソースコードを抽出して，分類を行う．

表 6: データセットの統計情報

収集期間	ファイル数	参加者数	コンテスト数	問題数	DB サイズ
2016/5/19~2016/11/15	1,644,636	14,520	739	3,218	357MB

さらに，先行研究の分類結果と提案手法の結果を比較するために，既存手法による分類の再現を行う．そのため，[2] のデータベースに新たにテーブルを追加し，メタ情報を保存した．追加したテーブル SubmissionStatus の詳細な説明を表 7 に示す．

表 7: SubmissionStatus テーブルの説明

カラム名	キー	説明	データ型
submission_id	PK FK	Codeforces における提出 ID	Integer
verdict		提出の正誤	String
time_conumed_millis		各テストの実行に かかった最大の時間 (ms)	Integer
memory_consumed_bytes		各テストの実行で 消費した最大のメモリサイズ (byte)	Integer

次に，分類を実行した問題の情報を表 8 に示す．

4.3 分類結果

4.3.1 ソースコードの組み合わせの分布

ステップ 1 の前処理の種類を便宜上，以下のようにラベル付けする．

表 8: 調査対象の Codeforces の問題

問題 ID	ファイル数
691A	1705

- A:プリプロセッサと単記号のどちらも削除する
- B:プリプロセッサを削除し，単記号を残す
- C:プリプロセッサを残し，単記号を削除する
- D:プリプロセッサと単記号のどちらも残す

実験の結果，表 9 に示す分布になった．プリプロセッサと単記号のどちらも削除した A は，タイプ 1 の数が最も多くなっている．他の前処理と比較して，多くの単語 N-gram が前処理の時点で削除されるため，構造的な特徴の分類に必要な要素が削除されてしまい，既存手法との差異が小さくなったと考えられる．

プリプロセッサを削除し，単記号を残した B は，タイプ 1 の数が最も少なかった．単記号を TAMBA の手法では含んでいないため，語彙ベクトルによるクラスタリングの時点で，分類結果が大きく異なっていると考えられる．

TAMBA と同じ前処理を行っている C は，タイプ 2 の数が最も多くなっている．2 回目のクラスタリングにおいて互いに異なる特徴量を用いているため，TAMBA が異なるクラスタと分類した組み合わせについても，提案手法で同じクラスタと分類した組み合わせが多くなったと考えられる．また，タイプ 4 の数についても全前処理の中で最も少ない．

プリプロセッサも単記号も残した D は，B と同様でタイプ 1 の数が少ない．同じく，単記号の影響だと類推する．

表 9: 各前処理のおけるソースコードの組み合わせの分布

前処理の種類	タイプ 1 の数	タイプ 2 の数	タイプ 3 の数	タイプ 4 の数
A	102,083	92,489	79,238	1,155,823
B	45,230	135,829	136,093	1,112,489
C	73,420	270,724	107,582	976,227
D	49,024	190,863	132,287	1,057,256

4.3.2 メトリクス値による箱ひげ図

A，B，C，D のそれぞれの前処理の，n_statements の箱ひげ図，TAMBA による手法の n_statements の箱ひげ図について，それぞれ，図 3, 4, 5, 6，7 に示す．また，A，B，C，D のそれぞれの前処理の，avg_depth の箱ひげ図，TAMBA による手法の avg_depth の箱ひげ図について，それぞれ，図 8, 9, 10, 11，12 に示す．

TAMBA の箱ひげ図と比較して，前処理 A，B はともに各色ごとの箱の分布が各色内でに大きく異なっている．構造ベクトルによるステップ 3 のクラスタリングにおいて，プリプロセッサの削除が有効だと類推する．

また，TAMBA の箱ひげ図と比較して，前処理 C，D の箱の分布は，各色内でそれほど大きくない．プリプロセッサを残すと，N-gram によるステップ 2 の分類で，構造的な特徴が吸収されてしまっていると類推する．

4.4 提案手法の問題点

提案手法では TAMBA[1] と同様固定数のクラスタ数を採用したが，問題数や，分類するソースコードの互いの類似度によって，クラスタ数は動的に設定するべきである．実際，691A の問題を分類した結果，ステップ 2，ステップ 3 のそれぞれのクラスタリングにおいて，ソースコードの数が極端に少ないまたは，ソースコードの数が 0 のクラスターが存在した．問題数や，各特徴ベクトルの分布に応じて，クラスタ数を決定する必要がある．

また，今回クラスタリングのアルゴリズムに cosine 距離と最長距離法による凝集型階層クラスタリングを採用したが，ベクトル間の類似度やクラスタの併合法は他にも存在する．さらには，k-means 法 [10] や自己組織化マップ [11] といったクラスタリング手法が存在する．各クラスタリング手法の分類結果と比較して，採用するクラスタリングのアルゴリズムを定める必要がある．

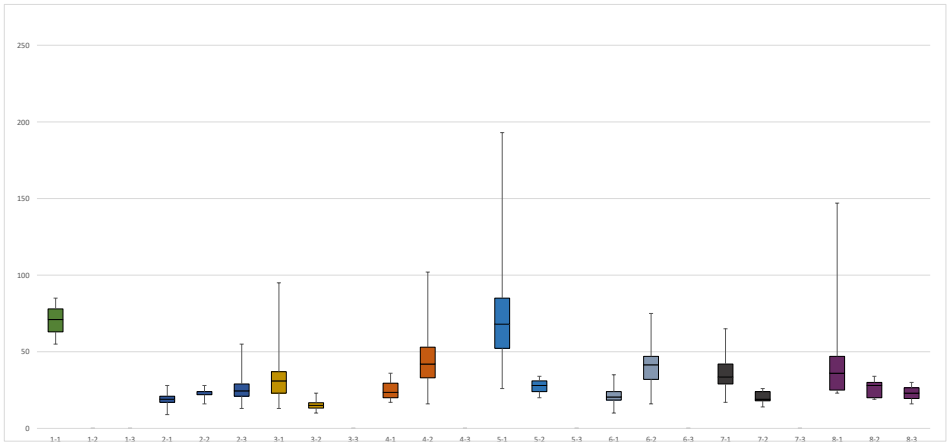

図 3: 前処理 A の n_statements による箱ひげ図

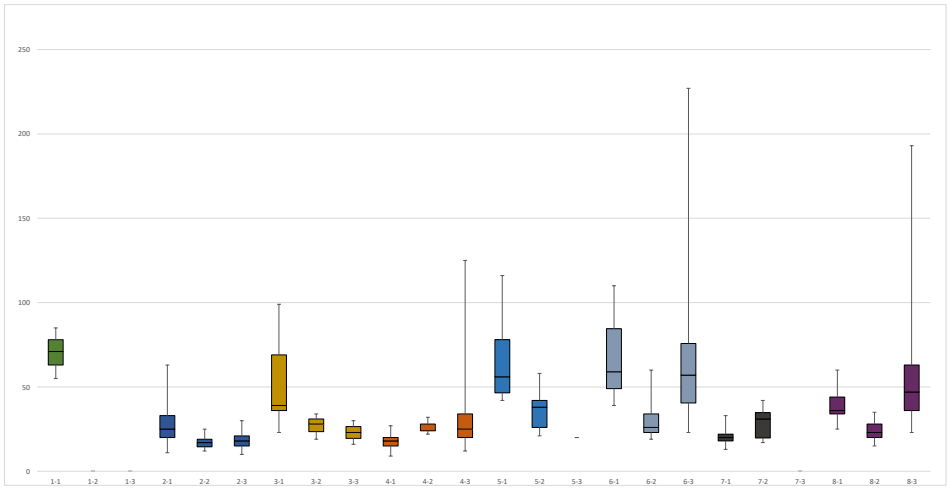

図 4: 前処理 B の n_statements による箱ひげ図

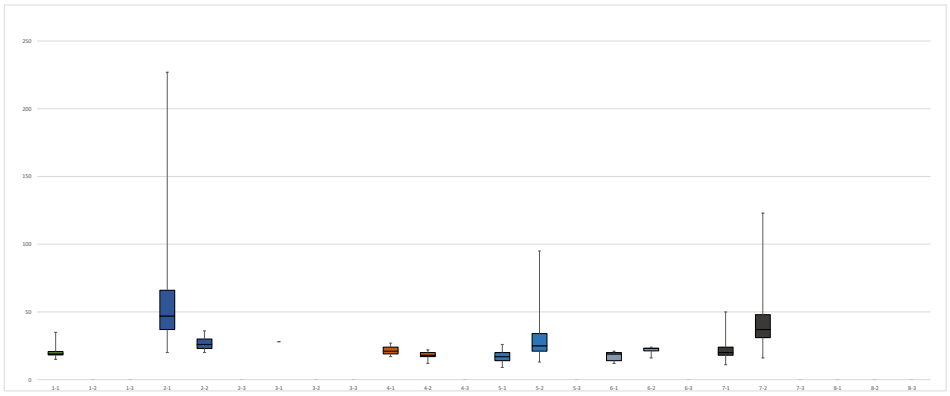

図 5: 前処理 C の n_statements による箱ひげ図

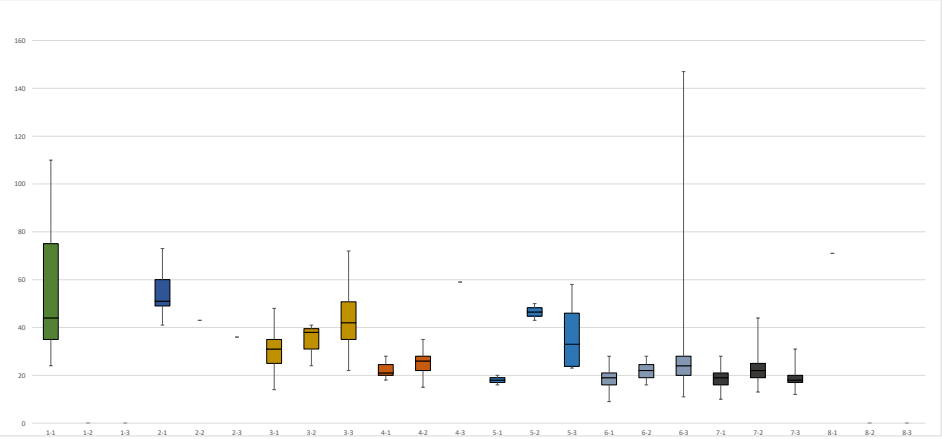

図 6: 前処理 D の n_statements による箱ひげ図

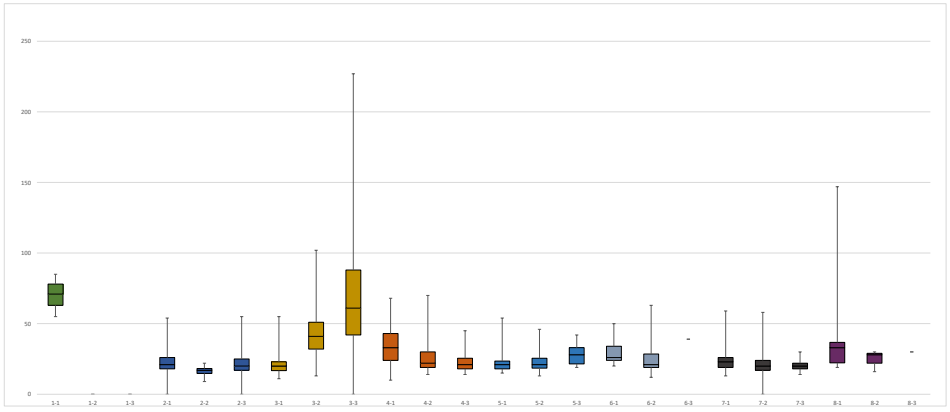

図 7: TAMBA の手法での分類の n_statements による箱ひげ図

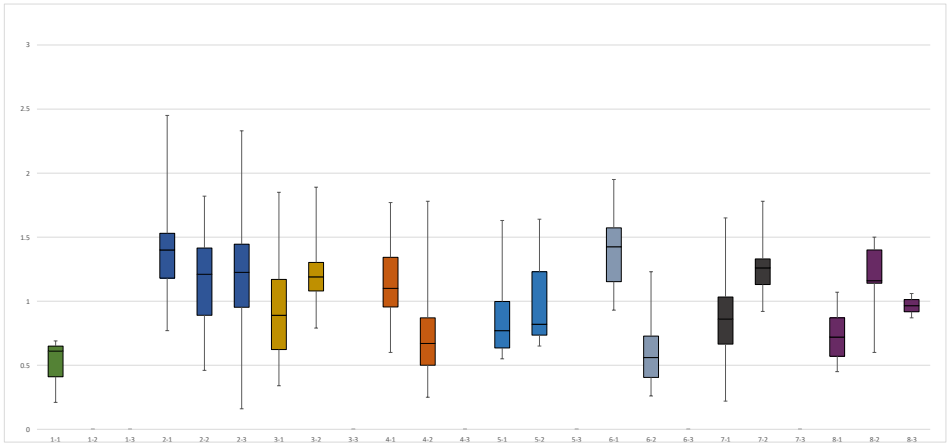

図 8: 前処理 A の avg_depth による箱ひげ図

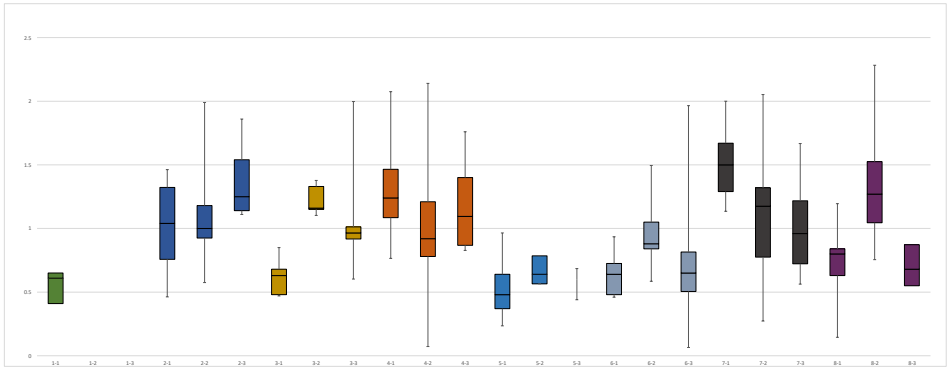

図 9: 前処理 B の avg_depth による箱ひげ図

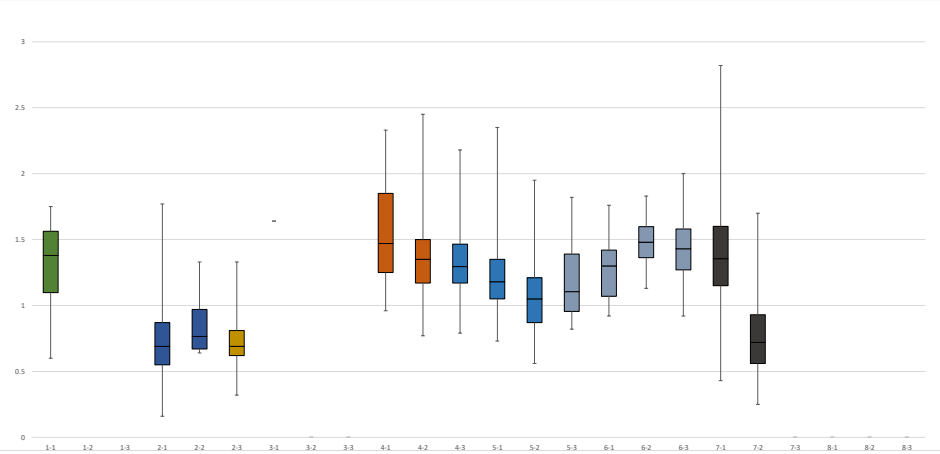

図 10: 前処理 C の avg_depth による箱ひげ図

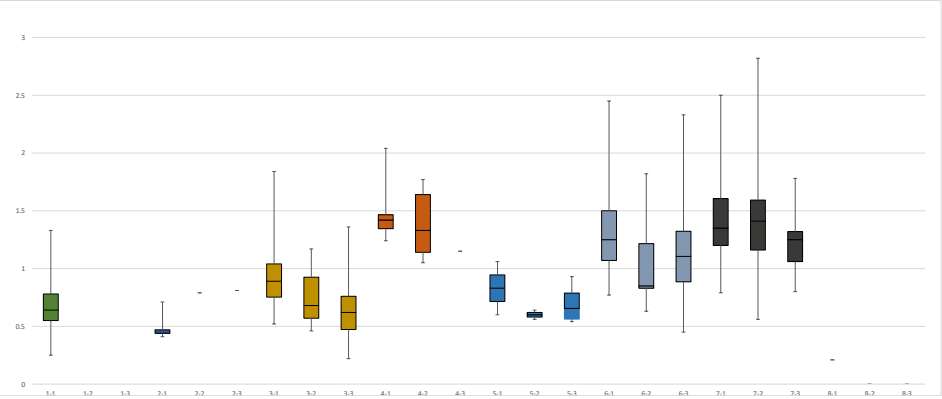

図 11: 前処理 D の avg_depth による箱ひげ図

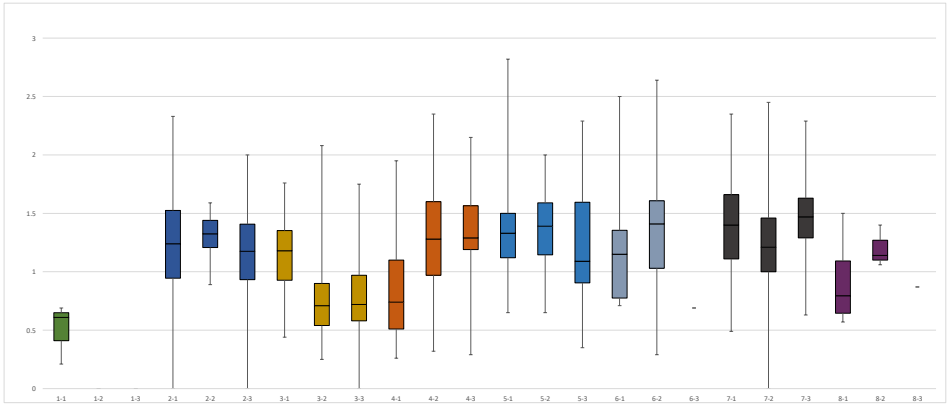

図 12: TAMBA の手法での分類の avg_depth による箱ひげ図

5　まとめと今後の課題

本研究では，構造的特徴を表すメトリクスを考慮したソースコードの分類手法を提案し，プログラミングコンテストのソースコードリポジトリに対して，実験を行った．既存手法と分類の結果を比較した結果，ソースコードの組み合わせごとのクラスタ分布が大きく異なり，構造的な特徴が反映されていると判断した．

Codeforces[3] には，多数の問題が存在し，問題の難易度，提出数などがそれぞれ異なる．採用するクラスタリングのアルゴリズムやクラスタ数の違いなども合わせた比較実験が，今後の課題として挙げられる．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上克郎 教授には，研究活動において貴重な御意見及び御指導を賜りました．井上 教授に心より感謝申し上げます．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下誠 准教授には，研究において貴重な御助言を賜りました．松下 准教授に深く感謝感謝申し上げます．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 神田哲也 助教には研究の方針から本論文の執筆に至るまで，多くの貴重な御指導及びご助言を賜りました．神田 哲也 助教の親身な御指導のおかげで本論文を完成させることができました．心より深謝申し上げます．

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上研究室の先輩方におきましては，たくさんの研究に関する助言をいただきました．特に，伊藤薫，嶋利一真，小笠原康貴 諸氏には，研究に関する相談に乗っていただき，様々なご支援をいただきました．心より感謝申し上げます．

最後に，私を常に支えてくださった大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上研究室の皆様に心より感謝いたします．

参考文献

[1] 新藤原, 尊雄中川, 秀明畑, 健一松本. プログラミング学習者向けソースコード提示システム tamba. ソフトウェアエンジニアリングシンポジウム 2016 論文集, 第 2016 巻, pp. 34–41, aug 2016.

[2] プログラミングコンテスト初級者・上級者間におけるソースコード特徴量の比較.

[3] Codeforces. http://codeforces.com/. (Accessed on 02/08/2019).

[4] Sourcemonitor v3.5. http://www.campwoodsw.com/sourcemonitor.html. (Accessed on 02/08/2019).

[5] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 4, pp. 308–320, Dec 1976.

[6] Jacob Cohen. *HStatistical Power Analysis for the Behavioral Sciences, 2nd edition.* Lawrence Erlbaum Associates, 1988.

[7] 白川真澄, 原隆浩, 西尾章治郎. コルモゴロフ複雑性に基づく idf の単語 n-gram への適用. データ工学と情報マネジメントに関するフォーラム (DEIM 2015), 3 2015. 助成：CPS, 開催地：福島, 開催日程：3/2-3/4, 発表日：3/2.

[8] iwnsew/ngweight: N-gram weighting scheme. https://github.com/iwnsew/ngweight. (Accessed on 02/08/2019).

[9] Aizu online judge: Programming challenge. http://judge.u-aizu.ac.jp/onlinejudge/. (Accessed on 02/09/2019).

[10] J.McQueen. *Some methods for classification and analysis of multivariate observations.* University of California Press, 1967.

[11] T. コホネン. 自己組織化マップ. 丸善出版, 2016.