Vol.2019-IFAT-134 No.5
Vol.2019-DC-112 No.5
2019/3/22

ソースコードの自動コメント付与を指向した構文木とキーワードの対応付け手法

畠山 和久[1,a)]　藤井 敦[2,b)]

概要：
現代は情報化社会であり，システムやプログラム，ソースコードというものが社会に必要不可欠になっている．プログラムの言語にはコメントという機能が存在し，コメントを元にソースコードを読むことで，コメントがない場合に比べて理解がしやすくなる．しかし，コメントが付いていても，ソースコードの理解には段階があり，読者のレベルに合ったものでないとあまり理解の助けにならない場合がある．
本研究は，大局的な意味を表すコメント生成として，単語の形式でキーワードをソースコードと対応づける手法を検討した．
課題に対してのソースコード群とアルゴリズムに関連する単語としてキーワードを対応させる教師あり学習を提案した．
ソースコードの素性として構文木の情報を含むことで，ソースコードとキーワードに一定の対応付けをすることができ，ソースコードをキーワードに振り分ける分類器の精度が上がることが示された．
今後の課題として，分類器の性能向上と，ソースコードに対して与えられたキーワードがどの部分を示しているかを明確にすることが挙げられる．

Associating source program codes with algorithm-related keywords

1. 序論

現代は情報化社会であり，ソフトウェアシステムが必要不可欠である．ソフトウェアシステムを記述するプログラムを理解するプログラマーが現代社会の持続可能性を高めていて，将来小学校でもプログラミング教育が導入されるように，システムやプログラム，ソースコードというものが一般的になっている．プログラムの言語にはコメントという機能が存在し，ソースコードの説明を記述する等に使用される．プログラム初学者にとってはコメントを元にソースコードを読むことで，変数や式の字句情報だけでなく，それらの示す値の意味などを把握でき，コメントなしに読むのに比べ，理解がしやすくなる．

ソースコードにコメントが付与されている場合は，まず全体像を把握し，ソースコードでの処理内容とコメントを照らし合わせるように理解することができるが，コメントが付与されていない場合には処理内容を少しずつ追いながら変数の変化を確かめるなど，全体像が見えないまま読み進めることになり，理解するためのコストが増えてしまう．

また，コメントが付いていても，読者のレベルに合ったものでないとあまり理解の助けにならない場合がある．
‘‘ for (int i = 0;i ¡ N; i++){...}’’
に対して
‘‘ i を 0 で初期化し，N 未満である間以下を実行した後，i を 1 増やす’’
のようなコメントはプログラミング初学者にとっては有用なコメントであるが，プログラム言語の文法を理解している読者にとっては自明なコメントであり，このコメントがあることによる可読性の向上はあまり期待されない．このようなコメントは自然言語分野でいう直訳のようなコメントである．つまり，ソースコードが記述する処理の内容ではなく，処理をそのまま自然言語に置き換えたようなコメントである．

[1] 東京工業大学工学部情報工学科
Tokyo Institute of technology
[2] 東京工業大学情報理工学院
Tokyo Institute of technology
[a)] hatakeyama.k.aa@m.titech.ac.jp
[b)] fujii@cs.titech.ac.jp

ⓒ 2019 Information Processing Society of Japan

本研究ではソースコード全体に対して，プログラム言語の文法は理解しているが，どのような課題に対しての処理なのかが理解できないというレベルの読者にとって有用なコメントを，キーワードという形で単語を提示する形での自動生成を検討する．

2. 関連研究

本研究の関連研究として，自然言語処理の自動コメント生成分野とソフトウェア解析の分野の研究がある．

コメント生成の分野において，Sridhara らの研究 [1] は，Java の Method を対象に，関数名や一行の処理に対して返り値の有無などの特定と，ソースコードを単語等に分割し，処理内容に対しての主語や目的語を特定する処理を行なった後，各構文に対してのルールベースでそれらの単語を繋ぎ合わせ，ソースコードを的確に自然言語で表現するコメントを生成する研究を行った．その結果

print(sendRequest())

に対して

/*Send request and get response, print response*/

というコメントを生成できるようにした．また，小田らの研究 [2] は，Python を対象に，一行ごとのソースコードに対して自然言語での 1 文形式のコメントを付けたデータセットを準備し，ソースコード，自然言語双方で構文木を生成し，Tree2String 翻訳を用いて統計的な自然言語とソースコードの対応付けの学習を指向し，その際自然言語間での翻訳にはないソースコードとの対応付けとして問題となる非終端記号を素性としたいなどの問題の解決のため，主辞の追加，ルールによる構文木の抽象化， 識別子と定数の抽象化を図った．その結果

if x % 5 == 0:

に対して

#もし x が 5 で割り切れるなら

というコメントを生成できるようにした．

Sridhara らの研究ではコメント生成の対象となるソースコードにおける関数や変数を特定し，それらの情報をコメントに反映することができる．小田らの研究では統計的手法を用いることで， ‘‘% 5 ==0’’ に対して「剰余が 0 と等しい」ではなく，「割り切れる」というような，字句や構文から得られる情報以上の言い回しをコメントに反映することができる．

また，ソフトウェア解析の分野において，渡邊らの研究 [3] は，プログラムの授業におけるプログラミング課題について，学習者の提出した複数の回答をプログラミング言語の予約語や演算記号の出現頻度を数え上げることによって解法の傾向の観点でクラスタリングをし，各クラスタを分析することで，回答者内での実装方針での分類や，回答者の習熟度を低コストで把握した．また，コードクローンと呼ばれるソースコードのコピー&ペーストによって生まれる類似性の高いソースコード対を検出する研究 [4] もある．

渡邊らの研究ではプログラム課題に対する回答となるソースコード群に対し，k 平均法や階層化クラスタリングを行った後，各クラスタについてどのように類似性があって同一クラスタに割り振られたかを個別のソースコードについて確認して分析を行い，同一のプログラミングの授業を受講している学生に対して例えば goto 文を用いて実装したグループとそうでないグループの分離や，‘‘++’’ という記法を習得できたグループと出来ていないグループを分離するなどして，クラスタリング後に各クラスタに手動でラベルを付与することを行なっているが，本研究ではプログラム課題に対応する自然言語で表されたキーワードを定義し，キーワードをラベルとしてソースコードをマルチラベル分類することを指向する点で異なっている．

コードクローンについては，文字列単位やトークン，構文木，依存関係やモジュール構造など様々な粒度での類似性を定義することができる．

文字列単位であれば自然言語処理の手法である Ngram を用いたものや，プログラム言語における単語であるトークンごとに比較することで空白などに依存しない正規化を指向したもの，構文木であれば関数定義などの制御構造単位での検出を考慮したもの，PDG(Program Dependence Graph) をソースコードから取り出し同型部分グラフを検出するもの，関数の依存関係や複雑度メトリクスによって関数に相当するコード断片から特徴ベクトルを取り出し比較するものなど様々な研究がなされている．コードクローンでは個別の事象に対して類似性を発見するのに対し，本研究では構文木の観点ではあるが，3 つ以上の複数のソースコードから本研究で提示するラベルごとに類似性を発見することを意図している点で異なっている．

コメント生成の分野での先行研究では，メソッド名や一行のソースコードなど小さい粒度でのソースコードを入力として，自然言語分野でいわゆる直訳のようなコメント生成がなされているが，本研究ではソースコード全体の大きな粒度に対してのコメント生成を検討する点で異なっている．

ソフトウェア解析の研究にあるような，複数のソースコードの類似性に着目し，コードクローンのように個別の事象に大きく限定されないようにデータセット全体から一般的な構文木の類似性を元に，ソースコード全体に対して処理内容を表すような直訳的でないコメントであるキーワードを対応付ける手法を検討した．

3. 提案手法

本研究で指向する，大きな粒度でソースコードを説明するキーワードとして考えられるものに，実装されるアルゴ

ⓒ 2019 Information Processing Society of Japan

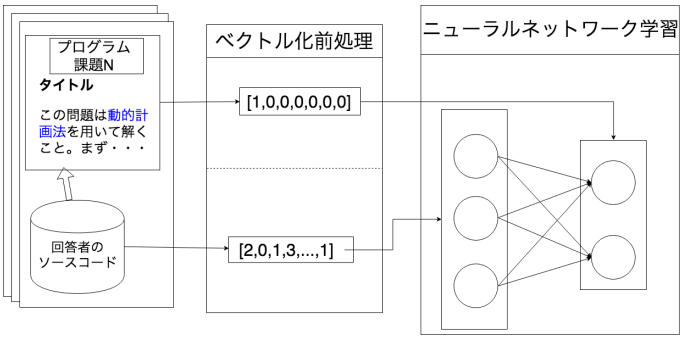


図 1　提案モデル概要

リズムや実装方針を表すソフトウェア工学での専門用語が挙げられる．例えば，動的計画法という単語をコメントとして与えられれば，読者の有する知識や，キーワードからの Web 検索によって，ソースコードで辿るべき配列の動きなどを重点的に読もうとする方針が立ちやすく，直訳的なコメント以上に有意義であると考える．このようなキーワードは例えば競技プログラミングという分野で様々に見つけられる．

競技プログラミングとは提示された課題に対して各回答者がソースコードを提出して正解率と早さを競うもので，コンテスト終了後に見られる課題の解説文書では上記のようなキーワードを用いて解説される．この課題に対して提出されたソースコード群には共通性があると仮定し，ソースコード群とキーワードをニューラルネットワークを用いて対応付けを学習することで未知のソースコードに対してもそれを解説するようなキーワードを出力するモデルを提案する．

また，ソースコードの共通性として，字句情報だけではなく，ソースコードが有する構文木の構造が大きな役割を果たしていると考え，素性として構文木の情報を与えることを提案する．

以上を踏まえ，**図 1** で示すようなソースコード，解説文双方からそれぞれの素性をデータから対応付けるニューラルネットワークを学習するモデルを構成した．

3.1　データ収集

本研究のデータセットとなるソースコードは競技プログラミングサービスの AtCoder[6] より収集した．このサービスではユーザーにプログラム課題を提示し，各ユーザーが課題に対する回答となるプログラムを提出する．コンテスト終了後には日本語で解説文書を提示する．

AtCoder では様々なプログラミング言語で提出され，課題をどれだけ満たしているかも多様なソースコードを閲覧できるが，本研究では C++14(G.C.C 5.4.1) で提出され，0 点でないソースコード，つまり最低でも部分点を獲得したソースコードに限定して収集した．

AtCoder では色々なコンテストが開催されるが，今回収集した AtCoderRegularContest (ARC) では 1 コンテストにつき 4 問出題される．本研究では 40 コンテスト 160 問題に対して 32844 ソースコードを収集した．

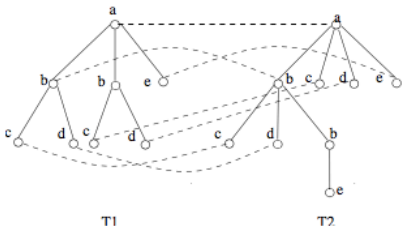


Figure 1: Tree Examples

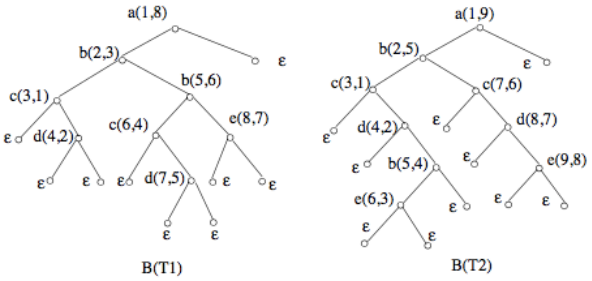


Figure 2: Normalized Binary Tree Representation

図 2　Rui らによる二分木表現

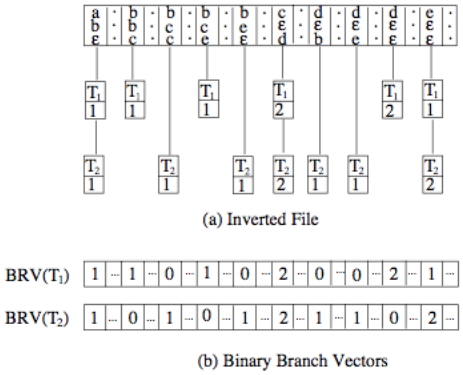


Figure 3: Binary Branch Vector Representation

図 3　Rui らによる BRV

3.2　ソースコード前処理

本項と次項ではニューラルネットワークの素性としてのベクトル化前処理について述べる．

ソースコードには抽象構文木で表される木構造が存在する．ソースコード全体の処理内容を説明することを目的とする本研究では，木構造の素性を含むために，Rui らの 2 つの木構造の類似性を図る研究 [5] を参考にベクトル化した．Rui らは一般の木構造データを二分木表現し，各ノードごとに (自ノード，左の子，右の子) の三つ組を数え上げ BinaryBranchVector(BRV) を作成し，二つの木構造に対して BRV の L1 距離が木編集距離の下限となることを証明した．

本研究ではこの BRV をソースコードの素性として組み込むことを提案する．clang コンパイラにおける抽象構文木を出力するオプション'ast-dump' を用いて得られる，ソースコードの抽象構文木に対して非終端記号をノードのラベルに対応させるように**図 2**，**図 3** の変換を行い，ソースコードをベクトルに写像する．

提案手法である BRV の有効性を確かめるため，ベースラインとしてプログラム言語の予約語や，演算記号の一部についての出現数を計測し特徴ベクトルとするモデルを設定した．

ソースコードでは define 文や typedef 文による簡略化，特定の処理の関数化が行われているが，字句単位でのものは書き換え，関数や for 文などの木構造については呼び出し文で定義文の木構造を接続するように前処理される．

3.3 キーワード抽出

本研究ではソースコードに付与するコメントはキーワードという形で単語を出力する．扱うキーワードは AtCoder における各プログラミング課題の解説文書から頻出する単語を人手で選出し，定義する．各ソースコードに対して対応するキーワードのベクトルは，それぞれの次元についてキーワードが解説文書に存在するかどうかの 2 値を取り，キーワードの種類数の次元に何もキーワードがないという 1 次元を足した次元数を持つベクトルになる．

3.4 ニューラルネットワークを用いた学習

ソースコード，解説文書の両方について上記の前処理を行ったものを，隠れ層 1 層を持つニューラルネットワークで学習する．入力となるソースコードから，前述したキーワードについてのベクトルを推定するマルチラベル分類を行う．入力層は各実験でのソースコードがベクトル化された時の次元数分のノード数を持ち，出力層は各ノードがラベルの出現確率に対応するノードを持つ．いずれの実験でも隠れ層は 512 ノードを持つ．入力層から隠れ層への活性化関数は ReLU 関数，出力層への活性化関数は sigmoid 関数を用いた．損失関数として Binary-CrossEntropy を用いた．

4. 評価実験

本研究では，ソースコード全体に対し，対応するキーワードを特定することで，可読性を高めるコメントとして付与できるようにすることを目的としている．キーワードの対応付けが行えることが可能で，ソースコードの素性の選び方によって精度よく対応するキーワードを特定できるかを検証するために以下の実験を行なった．

表 1 ラベルごとの問題数
Table 1 number of questions of each label

DP	ソート	二分探索	全探索	累積和	セグメント木
34	24	11	17	13	5

表 2 各モデルについての比較

モデル	入力次元数	説明
Tok	120	ソースコードから予約語等を数える．
Rec	-	常に Recall=1
Pro	195232	Tok モデルに構文木の素性を加える．$120 + 58^3 = 195232$ 次元

4.1 コメント

コメントとして出力される候補として，今回定義したキーワードは以下の 6 つである．

- 動的計画法
- ソート
- 二分探索
- 全探索
- 累積和
- セグメントツリー

これらを一部表記揺れを考慮しつつ，各プログラム課題の解説文書について存在するかしないかの 2 値での Bag Of Words でベクトル化する．ラベルごとの分布は**表 1** の通りである．1 つの問題に対して，複数のキーワードが対応することもある点に注意されたい．

4.2 ソースコード

ソースコードの素性の取り方として，提案手法である構文木を素性とするものの有効性を確かめるために，以下の 3 つのモデルを比較する．

- 主にソースコードの字句情報を素性とする，プログラム言語の予約語や演算記号の一部を数える Token モデル．特徴ベクトルでは，位置情報としてグローバル関数での予約語や演算記号には 3 倍の重み付けで数えている．
- 入力によらず全ての次元が 1 のベクトルを出力する，全てのキーワードについて再現率 1 を達成する Recall モデル．
- 上記の Token モデルにソースコードの抽象構文木の素性を追加した提案手法である Proposed モデル．main 関数の関数宣言である非終端記号を根とする抽象構文木を対象にする．main 関数に頻出な非終端記号 58 種類に対して Rui らの手法を適用し，58^3 次元が追加される．

ベースラインとして Token モデル (以下 Tok モデル) と Recall モデル (以下 Rec モデル) を用意し，Proposed モデル (以下 Pro モデル) とラベルごとの適合率，再現率，F 値の観点で比較する．

ⓒ 2019 Information Processing Society of Japan

実験は 40 コンテストについてを 10 コンテストずつに分割し 4 分割交差検証で行なった．

5. 結果

結果は**表 3** の通りである．

ただし表の平均はラベルごとではなく個別のソースコードごとの混同行列を足し合わせるマイクロ平均である．

ラベルごとに Tok モデルと Pro モデルでの適合率，再現率，F 値を比較すると，どのラベルについても Pro モデルの方が値が上回っていて，提案手法に有効性が認められる．例えばラベル DP について Tok モデルでの F-measure 0.1948 と Pro モデルでの F-measure 0.3251 を比較することで上記の結果を確認する．Pro モデルと Rec モデルを F 値の観点で比較するとラベルごとに大小関係が異なっていて，ラベルによって十分にソースコードからキーワードを容易に当てられるものとそうでないものがあるのが認められる．

5.1 考察

結果から，本研究である，粒度の大きなソースコードの処理内容分類については，Tok モデルと Rec モデルを F 値の観点で比較して Tok モデルの方が値が小さいことから字句情報だけでは精度よく分類できないことが認められる．

複数行のソースコードから素性を得る際に，例えば Tok モデルでは for 文が並列に並べられる場合と for 文の statement に for 文が記述される二重ループの場合をその素性の取り方ゆえ同一視することになるが，処理内容を踏まえると両者は別物であり，構文木を素性に含むことを提案する Pro モデルでは両者を別物と扱い，処理内容を区別することができる．その結果提案手法にあるように構文木の素性を追加することで分類器の精度を上げることができたと考える．

Tok モデルにおいて適合率の高い DP とセグメントツリーに対して特徴ベクトルを調べたところ，DP については”const”，セグメントツリーにおいては”void”と”<=”が比較的出現頻度が高く，手がかりとなっていたと考えられる．

DP については配列の更新式においてオーバーフローを防ぐための剰余が問題に設定されている場合に関しての定数宣言に const が用いられることが多く，セグメントツリーに関してはツリーの更新に用いる関数が”void”を返すことや，区間の特定の際に”<=”を用いるためこのような特定の予約語が比較的多く出現していたと考えられる．これらが注目され，DP とセグメントツリーの二つのラベルにおいて少なくとも適合率をあげる手がかりとして Tok モデルが比較的働いたと考えられる．

構文木を用いても F 値があまり上がらなかった全探索と累積和については，これらのラベルは本研究で選出した他ののラベルや，本研究では選出しなかったキーワードと共に現れることで構文木として対応するラベル箇所が認識されづらく，性能が上がりにくかったと考えられる．

数値の観点で見ると一つのテストデータ集合については**表 4** のようになる．このテストデータ集合は全部で 7769 のソースコードからなっている．Tok モデルと Pro モデルの Precision,Recall について, ソースコードの数の情報を示している．例えば DP ラベルについて Tok モデルの Precision はシステムが 777 ソースコードについて DP と分類し，そのうち実際に DP のラベルを持つものは 217 ソースコードであったということであり，Recall はテストデータ集合のうちの 1208 の DP のラベルを持つソースコードのうち，217 のソースコードをシステムが挙げられたということを示している．Tok モデルと Pro モデルについての各ラベルの F 値はこのテストデータ集合内で計算した F 値である．

Rec モデルについては割合のみを示している．Precision が全 7769 ソースコードのうち各ラベルがどれだけ含まれているかの割合を示していて，F-measure がこのテストデータ集合における Precision と再現率 (=1) を用いて計算される F 値を示している．

表 3 と比べて F 値等の数値に違いがあることからもわかるようにテストデータ集合によってばらつきが大きいが，テストデータ集合のうちの 1 つでは DP について全部で 1208 の DP ラベルを持つソースコードがある中，Tok モデルのベースラインと比べ 27 個多く正解を見つけられた．

また今回の実験ではソースコード全体についてキーワードを与えているが，全探索と累積和の結果についての考察で述べたように，1 つのソースコードについて複数のキーワード要素を含む場合があるので，それぞれのキーワードが示す手続き部分のみをハイライトする仕組みがあれば有用である．

6. 結論

6.1 まとめ

プログラムのソースコードを読む際には理解の助けになる読者のレベルに合わせたコメントが重要であるため，本研究では，プログラム言語の文法は理解しているが，どのような処理内容であるかはわからないというレベルの読者に対して有用なコメントを自動的に付与することを目的とし，既存研究であるような，一行や，ユーザー定義関数の関数名などに対する自動コメント生成に比べ，粒度の大きいソースコードに対して，プログラムの実装方針などを表すキーワードという形でコメントを対応付ける手法を検討した．

その際ソフトウェア解析の既存研究にあるようなソフトウェアの類似性に着目し，構文木の素性を用いることを提

ⓒ 2019 Information Processing Society of Japan

表 3 各モデルについてラベルごとの適合率，再現率，F 値を示す表

モデル	評価指標	DP	ソート	二分探索	全探索	累積和	セグメント木	平均
Tok	Precision	0.2980	0.1884	0.0839	0.1241	0.1083	0.4450	0.2090
	Recall	0.1489	0.0746	0.0181	0.0542	0.0372	0.0044	0.0836
	F-measure	0.1948	0.0974	0.0209	0.0704	0.0320	0.0793	0.1178
Rec	Precision	0.1713	0.1327	0.0560	0.0900	0.0826	0.0371	0.0919
	Recall	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	F-measure	0.2903	0.2325	0.1053	0.1607	0.1477	0.0712	0.1681
Pro	Precision	0.3848	0.4507	0.3111	0.1715	0.1184	0.7454	0.3433
	Recall	0.3149	0.2789	0.2374	0.0717	0.0291	0.3242	0.2049
	F-measure	0.3251	0.3340	0.2260	0.0831	0.0458	0.4373	0.2517

表 4 一つのテストデータ集合についてソースコードの数による
適合率，再現率とそれに対する F 値を示す表

モデル	評価指標	DP	ソート	二分探索	全探索	累積和	セグメント木
Tok	Precision	217/777	92/368	2/55	82/367	57/183	7/19
	Recall	217/1208	92/1136	2/616	82/549	57/1195	7/397
	F-measure	0.216	0.122	0.005	0.179	0.082	0.033
Rec	Precision	0.155	0.146	0.079	0.070	0.0153	0.051
	Recall	1.000	1.000	1.000	1.000	1.000	1.000
	F-measure	0.269	0.255	0.146	0.132	0.266	0.097
Pro	Precision	244/517	444/1019	13/202	36/301	59/208	86/109
	Recall	244/1208	444/1136	13/616	36/549	59/1195	86/397
	F-measure	0.282	0.412	0.031	0.084	0.084	0.339

案した．

提案するモデルは競技プログラミングの課題に対する解説文書から頻出するキーワードを人手で抽出し，回答群となるソースコードと対応付けを学習することで，未知のソースコードに対してコメント付与することを指向し，学習には隠れ層 1 層を持つニューラルネットワークを用いて，入力がラベルごとにクラスに属するかどうかを判別するマルチラベル分類を行なった．その際，ソースコードの素性として，字句情報となるプログラム言語の予約語や演算記号を数えるだけでなく，構文木の素性を導入することで，上記の分類器の性能が上昇することが認められた．また，キーワードによって構文木を素性とする分類器が精度よく働くものとそうでないキーワードがあることが認められた．

6.2 今後の課題

残された課題として，字句情報や構文木からの異なる方式での素性，もしくは別の観点からソースコードの素性を得ることによる分類器の性能向上と，実用のために，ソースコードに対して付与されたコメントが対応する適切な部分をハイライトする必要があると考える．

参考文献

[1] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock and K. Vijay-Shanker *Towards Automatically Generating Summary Comments for Java Methods* ASE '10 Proceedings of the IEEE/ACM international conference on Automated software engineering Pages 43-52 , 2010

[2] 小田 悠介, 札場 寛之, ニュービッグ グラム, サクティ サクリアニ, 戸田 智基, 中村 哲, "ソースコード構文木からの統計的自動コメント生成" 研究報告自然言語処理（NL）Vol.2014-NL-219, No.12, 1 - 9, 2014-12-09

[3] 渡邊 裕貴, "プログラミング入門教育におけるソースコードの自動分類" 電気通信大学平成 27 年度卒業論文 2016 年 2 月

[4] 神谷 年洋, 肥後 芳樹, 吉田 則裕, "コードクローン検出技術の展開" 日本ソフトウェア科学会 , 28 巻 3 号 pp.29-42 2011 年

[5] Rui Yang, Panos Kalnis, Anthony K. H. Tung *Similarity Evaluation on Tree-structured Data*, SIGMOD '05 Proceedings of the 2005 ACM SIGMOD international conference on Management of data Pages 754-765 , 2005

[6] AtCoder, https://atcoder.jp/?lang=ja

ⓒ 2019 Information Processing Society of Japan