修士学位論文

題目

ソースコードのグラフ表現を利用した
深層学習による開発者のコーディング能力の評価手法

指導教員

井上 克郎 教授

報告者

松井 智寛

令和 4 年 2 月 2 日

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 ソフトウェア工学講座

令和 3 年度 修士学位論文

ソースコードのグラフ表現を利用した深層学習による開発者のコーディング能力の評価手法

松井 智寛

内容梗概

ソフトウェア開発者のコーディング能力を知ることができると，企業は必要なコーディング能力を持つ開発者を採用し，開発プロジェクトに配置することができる．しかし，このような場面でコーディング能力の評価を誤り，採用の決定を間違えてしまうと大きなコストがかかってしまう．

そこで，ソフトウェア開発者のコーディング能力を客観的に評価するため，ソースコードを利用した機械学習・深層学習による評価が行われている．ランダムフォレストというモデルを利用した機械学習による手法では上級者または初級者で利用されることが多い予約語のソースコード中の出現回数とソースコード行数といったメトリクスを収集し，モデルに入力している．また，LSTM や CNN というモデルを利用した深層学習による手法では，ソースコードを単語毎に分割して自然言語のように扱い，モデルに入力している．しかし，これらの手法でソースコードの特徴的な構造に関する情報や変数などの意味的な情報をほとんど利用していない．

そこで本研究では，まずソースコードを構造に関する情報や変数の意味的な情報を表すことができるグラフへ変換し，グラフを学習することができる RGCN という深層学習モデルを利用してコーディング能力の評価を行う手法を提案する．

評価実験のデータセットには既存研究で用いられたプログラミングコンテストの Codeforces のデータセットを利用する．コンテストの参加者は成績によって順位がつけられ，レーティングが変動する．このレーティングに従って開発者を上級者・中級者・初級者に分類して学習・評価を行った．その結果，accuracy は 0.871，F 値は上級者で 0.891，中級者で 0.881，初級者で 0.835 となり，いずれも既存手法に比べてよい値となった．

主な用語

コーディング能力
深層学習
プログラミングコンテスト

目 次

1　まえがき

ソフトウェア開発において，企業は必要なコーディング能力を持つ開発者を採用し，開発プロジェクトに配置することが必要になっている．

Facebook や Google，Amazon といった技術系の企業では採用の場面でコーディングの課題を実施することで応募者の能力を測っている [11]．しかし，こういった場面で人によるソースコードの評価が行われると非常に大きなコストがかかってしまう．そのため，コストがかからない方法で開発者の能力を評価する必要がある．

開発経験が長いほど能力が高いといった評価が行われることがある [17]．しかし，開発者の経験と専門知識に関する調査 [2] では一貫した結果が得られておらず，開発経験の長さを利用して能力を予測すると採用の決定を間違えてしまう可能性があり，採用決定を間違えてしまった場合のコストはその従業員の年俸の 10 倍になるとも言われている [3]．

開発者の専門性はソースコードの質に現れるとされている [2]．そのため，ソースコードから開発者の能力を予測できると考えられる．

これまでにソースコード自体から開発者の能力を予測する方法として，機械学習や深層学習を利用したモデル [9, 21, 22, 23, 24] や，Zipf の法則のパラメータ推定を利用したもの [15] がある．しかし，機械学習・深層学習を利用するものはソースコードの特徴的な構造に関する情報や変数などの意味的な情報を利用していない．また，パラメータ推定を利用するものは推定に必要なソースコードの量が多いといった欠点がある．

そこで，本研究ではまずソースコードの構造に関する情報や変数の意味的な情報をグラフとして表現する．そして，グラフを学習することができる RGCN[16] を利用してソースコードから開発者の能力を予測する手法を提案する．

手法の評価には既存研究 [21, 22, 23, 24] で用いられたプログラミングコンテストのデータセットを利用する．コンテストの参加者は成績によって順位がつけられ，レーティングが変動する．問題を解いて実装する能力，すなわちコーディング能力が高い開発者が良い順位を取り，レーティングが高くなると考えられるため，レーティングが高い開発者から上級者・中級者・初級者と分類を行った．これを利用し，3 値分類による学習・評価を既存手法と提案手法で行い，結果を比較したところ，提案手法は既存手法に比べて精度が高い予測を行えることが確認できた．

以下，2 章では，本研究の背景として，プログラミングコンテストと機械学習・深層学習を用いた能力の評価手法，そしてソースコードのグラフ表現について説明する．3 章では提案手法について説明する．4 章では評価実験の内容と結果について説明し，考察を行う．最後に 5 章ではまとめと今後の課題について述べる．

2　背景

この章では本研究の背景として，評価対象のデータセットとして利用するプログラムコンテストとその採点に利用されるオンラインジャッジシステム，そして既存の機械学習・深層学習を用いた評価手法，ソースコードのグラフ表現について説明する．プログラミングコンテストについてはさまざまなものが存在するが，本研究では問題の要求を満たすプログラムを時間内に作成することを競うコンテストを扱う．

以下ではまず，オンラインジャッジシステムについて説明する．

2.1　オンラインジャッジシステム

プログラミングコンテストにおいて，その採点に利用されるオンラインジャッジシステムについて説明する．オンラインジャッジシステムにはさまざまな問題が収録されており，その利用者は問題を選択し，回答を提出する．そして，システムは利用者に提出された回答の採点結果を通知する．採点の基準の一例を以下に示す．

- コンパイルできるかどうか
- 不正なメモリアクセスがないかどうか
- 制限時間内にプログラムが終了するか

このような基準の内，満たさない基準があればそのことを通知し，すべての基準を満たせば正解であることを通知する．

こういったオンラインジャッジシステムの一例として，国内では AIZU ONLINE JUDGE[1]，国外では Topcoder といったものが存在する．

2.2　プログラミングコンテスト

プログラミングコンテストはプログラミングの能力や技術を競い合うコンテストのことである．オンラインジャッジシステムが採用されており，同じ問題に対して，同じ時間内に複数の参加者がソースコードを記述し，提出する．コンテストが終了すると，正解問題数や回答時間に応じて参加者の順位が決定し，レーティング [6, 14] が変動する．プログラミングコンテストには Codeforces[2]のように個人でプログラムを作成して回答するもの以外にも，ACM ICPC[3]のように複数の参加者がチームを組んで回答するものも存在する．本研究では，個人で参加するプログラミングコンテストである Codeforces を対象とする．

[1] http://judge.u-aizu.ac.jp/onlinejudge/
[2] http://codeforces.com/
[3] https://icpc.baylor.edu/

2.2.1 ルールと流れ

プログラミングコンテストの Codeforces の大まかな流れについて説明する．Codeforces における一般的なコンテストでは，コンテストの開始時間が指定されており，開始時間になるとコンテストの問題が公開され，参加者は問題を解き始める．問題を解く順序や，用いるプログラミング言語は自由であり，得点は解いた問題の難易度や回答にかかった時間，そしてこれまでの提出回数によって変わる．また，参加者は制限時間内であれば，同じ問題に対して何度も回答を提出することが可能であり，回答ソースコードの正誤やコンパイル可能性については提出の可否に影響しない．回答ソースコードは提出された時点でオンラインジャッジシステムによって事前テストの実行が行われ，通過したか否かが参加者に通知される．事前テストは全テストケースを網羅しておらず，事前テストを通過しても回答が正しくない場合もある．コンテスト時間終了後に提出ソースコードに対して最終テストが実施され，参加者の最終的な得点が決定され，それに伴い順位も決定され，レーティングが変動する．

2.2.2 開催規模や参加者

本研究で利用する Codeforces においては，開催規模や参加者は以下の通りになっている．

- 開催頻度：偏りはあるが，大体週に 1 回以上
- 参加人数：偏りが大きいが，毎コンテスト 10000〜30000 人程度
- 国籍：全世界から参加 (使用言語はロシア語・英語)

また，過去 6 か月以内に一度でもコンテストに参加したことのあるユーザーを Codeforces ではアクティブユーザーと定義しているが，Codeforces におけるアクティブユーザー数は 2022 年 1 月 6 日現在，108310 人となっている．

2.2.3 レーティングシステム

Codeforces は参加者の熟練度をレーティング [14] を用いて表している．コンテストが行われるたびに参加者のレーティングが順位によって変動する．レーティングの計算方式はチェスなどの対戦型の競技で用いられるイロレーティング [6] と呼ばれる方式に似たものとなっている．本研究では，既存の研究 [21, 22, 23, 24] と同様に，このレーティングが高い参加者を能力が高い参加者として扱う．

イロレーティングでは A と B のレーティングをそれぞれ r_A，r_B とすると，A が B に勝利する確率が

$P(r_A, r_B) = \frac{1}{1+10^{\frac{r_B - r_A}{400}}}$

となるようにレーティングが調整される．Codeforces では A が B より高い順位にいる確率となる．

Codeforces でのレーティング変更計算のアルゴリズムを Algorithm 1 に示す．(1) の for 文でおおよそのレーティング変動を計算している．関数 $getSeed(r)$ はレーティング r の順位の期待値を返す関数であり，$seed_i \leftarrow getSeed(r_i)$ とすることで，$seed_i$ はレーティング r_i の参加者の順位の期待値となる．そして $seed_i$ と実際の順位 $rank_i$ の幾何平均を m_i とし，$getSeed(R_i) = m_i$ となるようなレーティング R_i の探索を行う．その後，探索によって得られた R_i と r_i の平均を取った後，全体のレーティングの変動が 0 になるように処理を行ったものを参加者 i の暫定的なレーティングとする．しかしこのままでは上位のレーティングがインフレする恐れがあるため，(2) 以降でレーティングの変動の調整を行っている．こうして最終的なレーティングが決まる．

2.3 機械学習・深層学習を用いた能力の評価手法

ソースコードそのものを用いて機械学習や深層学習によって能力を評価する手法には，ソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法 [21, 22, 23, 24] やソースコードを単語毎に分割して自然言語のように扱った深層学習による手法 [9] が存在する．

槇原 [21, 22] はプログラミングコンテストに提出されたソースコードを対象とした学習を行った．プログラミングコンテストの上級者と初級者の特定の予約語の利用頻度やメトリクスの値の違いは堤 [20] によって確認されている．そこで槇原はプログラミングコンテストのレーティングをもとに参加者を上級者と初級者に分類し，ソースコードを堤により確認された上級者または初級者で利用頻度が高い予約語の利用頻度とメトリクスの値から成るベクトルに変換して決定木や SVM で学習を行うことで，開発者の能力を予測することができるモデルを作成した．

また，この評価手法に改良を加えたものも存在する [23, 24]．ここでは槇原の研究では利用されていなかった上級者・初級者以外の参加者が中級者として評価の対象に加えられ，さらに利用するメトリクスが増やされた．また，学習アルゴリズムもランダムフォレストに変更することでモデルの改良が行われた．利用された予約語とメトリクスはそれぞれ表 1 と表 2 に示している．

Javeed ら [9] は GitHub に公開されている Java プロジェクトを対象とした学習を行った．ここではまず 14,785 のプロジェクトが収集され，コンパイル可能であった 6,244 のプロジェクトが機密性，信頼性，複雑性，行数，保守性，重複といった観点から expert と novice に分類される．そしてソースコードを単語毎に分割し，単語の埋め込みを深層学習のライブラ

Algorithm 1 Codeforces におけるレーティング変更計算

Data: U：コンテストの全参加者

Data: r_i：参加者 i のコンテスト前のレーティング

Data: $rank_i$：参加者 i のコンテストでの順位

Result: r'_i：参加者 i の変化後のレーティング

function $getSeed(r)$

  return $\Sigma_{i \in U} P(r_i, r) + 1$;

function $getRatingToRank(r)$

  $left \leftarrow 1$;

  $right \leftarrow 8000$;

  while $right - left > 1$ **do**

    $mid \leftarrow \frac{left+right}{2}$;

    if $getSeed(mid) < r$ **then**

      $right \leftarrow mid$;

    else

      $left \leftarrow mid$;

  return $left$;

$sum \leftarrow 0$; // (1)

for $i \in U$ **do**

  $seed_i \leftarrow getSeed(r_i)$;

  $m_i \leftarrow \sqrt{seed_i * rank_i}$;

  $R_i \leftarrow getRatingToRank(m_i)$;

  $d_i \leftarrow \frac{R_i - r_i}{2}$;

  $sum \leftarrow sum + d_i$;

for $i \in U$ **do** // レーティング変動の合計を 0 にする

  $d_i \leftarrow d_i - \left(\frac{sum}{|U|} + 1\right)$;

// (2) 上位のレーティングのインフレを抑える処理

$topU \leftarrow U$ の上位 $min(|U|, 4\sqrt{|U|})$ 人;

$sum \leftarrow \Sigma_{i \in topU} d_i$

$inc \leftarrow min(max(-\frac{sum}{|topU|}, 10), 0)$;

for $i \in U$ **do** // 最終的なレーティングの決定

  $d_i \leftarrow d_i + inc$;

  $r'_i + d_i$;

表 1: [23, 24] で利用された予約語

asm	break	case	catch
class	continue	decltype	do
else	enum	extern	for
friend	goto	if	namespace
operator	private	public	return
struct	switch	template	try
typedef	typeid	typename	using
while			

表 2: [23, 24] で利用されたメトリクス

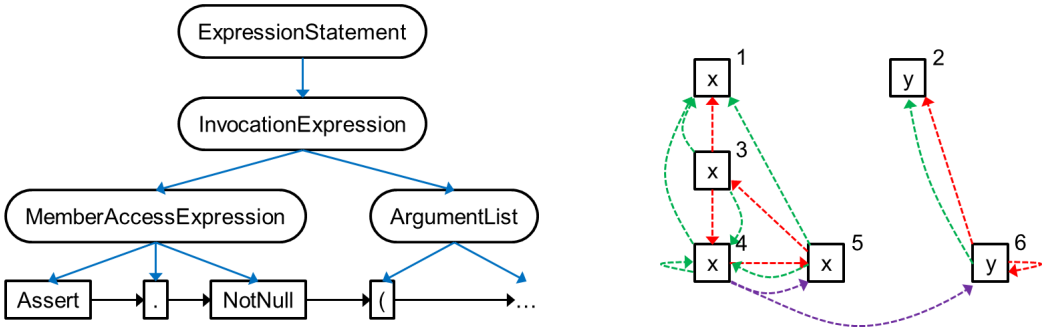
メトリクス	説明
avg_complexity	各関数の循環的複雑度の平均値
max_complexity	各関数の循環的複雑度の最大値
avg_depth	各関数のネスト深さの平均値
max_depth	各関数のネスト深さの最大値
methods_per_class	クラス当たりのメソッド数
n_classes	クラス数
n_func	関数の数
n_lines	物理行数
n_statements	セミコロンで区切られた論理行数
percent_branch_statements	全体の論理行数に占める分岐文 (if，else，for，while，goto，break，continue，switch，case，default，return 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
avg_statements_per_method	各メソッドの論理行数の平均値
statements_at_block_level_0	深さ 0 の論理行数
statements_at_block_level_1	深さ 1 の論理行数
statements_at_block_level_2	深さ 2 の論理行数
⋮	⋮
statements_at_block_level_8	深さ 8 の論理行数
statements_at_block_level_9	深さ 9 以上の論理行数

リである Keras[4]に実装されている Embedding 層を利用して行い，LSTM[7] や CNN[8]，またはその両方を利用した学習を行うことで，開発者の能力を予測することができるモデルを作成した．

しかし，これらのモデルによる予測はソースコードから予測を行うにもかかわらず，利用する情報はソースコードにおいて特徴的な構造に関する情報や変数の意味的な情報が大きく損なわれている．そのため，このような情報を利用することができればより精度の高い予測を行うことができると考えられる．

2.4 ソースコードのグラフ表現

Allamanis ら [1] はソースコードをグラフとして表現する手法を提案した．ノードにはソースコードを構文木に変換した際の文法とトークンを利用する．そして構文木から得られる親子関係を表す Child エッジとトークンの順序関係を表す NextToken エッジを接続する．この例を図 1a に示している．変数が最後に利用されうる場所へ LastUse エッジ，最後に書き込まれうる場所へ LastWrite エッジ，代入文の左辺から右辺へ ComputedFrom エッジ，最後に出現した場所へ LastLexicalUse エッジを接続する．LastUse エッジ，LastWrite エッジ，ComputedFrom エッジの例は図 1b に示している．他にもメソッド中の return トークンに関する ReturnsTo エッジや仮引数に関する FormalArgName エッジ，if 文に関する GuardedBy，GuardedByNegation エッジといったものが定義されている．そして最後にそれらのエッジの両端を逆にしたものを別のラベルとして加える．



(a) メソッド呼び出しにおける Child エッジ (青) と NextToken エッジ (黒) の例

(b) $(x^1, y^2) = \mathrm{Foo}(); \mathrm{while}(x^3 > 0) x^4 = x^5 + y^6;$ というプログラム中の LastUse エッジ (赤)，LastWrite エッジ (緑)，ComputedFrom エッジ (紫) の例

図 1: グラフ中のエッジの例

このグラフ表現により，従来使われることが多かった自然言語の手法では扱うことができなかったソースコードの特徴的な構文的構造や意味的構造を扱うことができるようになった．

[4] https://keras.io/ja/

そしてこのグラフ表現とグラフを対象とした深層学習のグラフニューラルネットワークの 1 つである GGNN[12] を利用して，変数の利用箇所の中から 1 つを空けたソースコードを提示した際にどの変数が使われるべきかを予測する VARMISUSE タスクにおいて従来の手法よりも優れた結果が得られることがわかった．このタスクの精度を高めるにはソースコードの構文に関する情報だけでなく，変数などの意味的な情報も必要である．このタスクで優れた結果が得られたことから，このグラフ表現と深層学習を用いるとそれらの情報を上手く学習できることがわかる．

また，このソースコードのグラフ表現は変数の予測以外にも，ソースコード中の脆弱性の特定 [19] やバグの検知・修正 [4] といった分野でも優れた結果が得られている．

3　提案手法

3.1　目的

2.3 節で示したように，既存の予約語の利用頻度とメトリクスを利用した機械学習による手法 [21, 22, 23, 24] やソースコードを単語ごとに分割して自然言語のように扱った深層学習による手法 [9] はソースコードの構造に関する情報や変数の意味的な情報がほとんど利用されていない．

そのため，私は 2.4 節で示したようなソースコードのグラフ表現とグラフを対象とした深層学習を利用することで開発者の能力を評価する手法を提案する．そこで，以降の節では，この手法におけるグラフ表現について説明した後，コーディング能力の評価の手順について説明する．

3.2　ソースコードのグラフ表現

Allamanis らが [1] 定義したグラフを利用した深層学習は 2.4 節で説明したように，ソースコードの特徴的な構造に関する情報や変数の意味的な情報を上手く学習できることが分かっている．また，本研究でのコーディング能力の評価対象はプログラミングコンテストの問題に対する回答のような比較的単純なものを想定している．しかし Allamanis らが定義したグラフはオープンソースのプロジェクトのような複雑なものを対象としており，エッジの種類が多く複雑なものとなっている．そこで，本研究では Allamanis らが定義したグラフの内，コーディング能力の評価に十分だと考えられたものを利用する．まず以下のものをノードとして利用する．

- ソースコード中のトークン
- 構文木から得られる文法を表す節

ノード間を接続するエッジは以下のものを利用する．

- 構文木の親子関係を表す Child エッジ
- トークンの順序関係を表す NextToken エッジ
- 同一変数の利用情報を表す LastLexicalUse エッジ

こうして得られたノードの中には”assignExpression”のように複数の単語の組合せとなっているものがある．このような単語の特徴をグラフニューラルネットワークで効果的に学習するため，”assign”や”Expression”のようなノード名を分割したノードを追加する．そして分割前のノードから分割後の各ノードには UsesSubtoken エッジを接続するようにする．この

ノードの分割および UsesSubtoken エッジは Microsoft が公開している VARMISUSE タスクの実装 [13] を利用した．

3.3　開発者のコーディング能力の評価の手順

開発者のコーディング能力の評価の手順を図 2 に示す．この手順は大きくソースコードからグラフへ変換する部分と深層学習を利用してグラフから能力を評価する部分に分けられる．以降の節でそれぞれの部分について説明する．

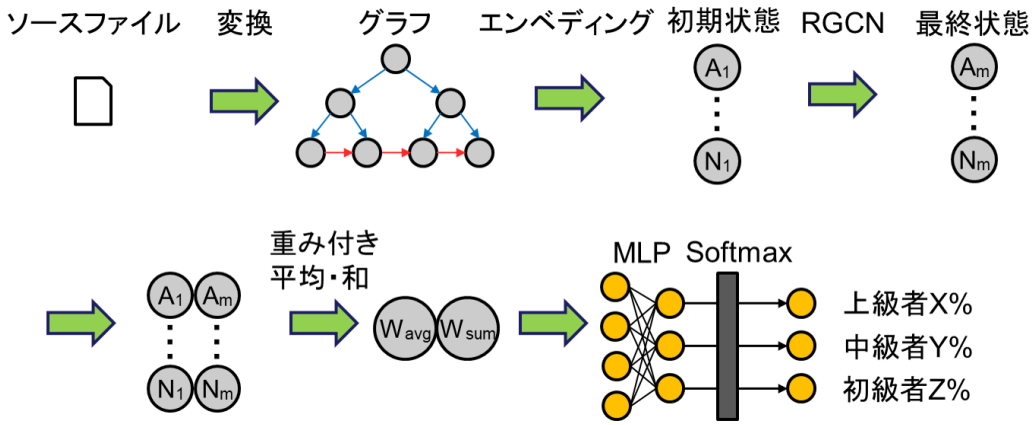


図 2: コーディング能力の評価の手順

3.3.1　ソースコードのグラフへの変換

ANTLR[5]と Understand[6]という 2 つのツールを利用して，以下の手順でソースコードからグラフへの変換を行った．

1. ANTLR でソースコードを構文木に変換する
2. 構文木中の各ノードをグラフのノードとする
3. 親ノードから子ノードに接続されるエッジを Child エッジとする
4. 前のトークンから後ろのトークン NextToken エッジを接続する
5. 冗長なノードを削除する
6. Understand を利用して LastLexicalUse エッジを追加する

[5] https://www.antlr.org/
[6] https://www.scitools.com/

7. ノード名を分割して新しいノードと UsesSubtoken エッジを追加する

冗長なノードの削除について説明する．

図 3 にソースコード中の代入文 n=1 を構文木に変換したものを示している．代入文を表すノード”assignmentExpression”から左辺，=，右辺を表すノードに分かれている．この例では代入文の右辺に 1 という定数が入っているが，代入文の右辺には定数以外にも変数や関数など様々なものが入る可能性がある．そういったものの中から右辺が定数であることを確定するまでの処理が赤い点線で囲まれた部分となっている．左辺も同様に変数であることを確定するまでの処理が赤い点線で囲まれた部分で行われている．これらの処理によって構文木が深くなっている．グラフニューラルネットワークではグラフの各ノードはエッジが隣接するノードから情報を集めて状態を更新していくため，この構文木のまま学習してしまうと例えば右辺の 1 というトークンが代入文の中にあるという情報を得るまでに非常に長い時間がかかってしまう．そのため，赤い点線で囲っている部分を冗長なノードとして削除することで効率よく学習させる．

そこで，以下の 2 つの条件を同時に満たすノードを冗長なノードと定義し，削除する．この際，対象の親ノードから接続される Child エッジはその子ノードへ接続する．

- 子ノードが文法を表すノードであるもの
- 子ノードが 1 つしかないもの

この条件に従い図 3 の冗長なノードを削除したものを図 4 に示している．構文木の深さは改善されており，学習の際にノードは効率よく情報を集めることができるようになる．また，4.1 節で説明するデータセットに対して冗長なノードの削除を行った結果，分割したノードの追加を行う前に存在するノードの約 73%を削減することができた．

次に新しいノードの追加と UsesSubtoken エッジについて説明する．キャメルケースまたはスネークケースに従ってノード名を分割して得られた単語のノードを新たに作成する．そして分割する前のノードから分割後の各ノードへ UsesSubtoken エッジを追加する．ただし，分割前のノードが予約語である場合，これらの処理は行わない．

こうしてソースコードをグラフに変換する様子を図 5 に示す．ただしここでは式以外を表す文法ノードと分割後のノード，そして UsesSubtoken エッジを省略している．

3.3.2 グラフを対象とした深層学習

この節では提案手法の内，グラフを対象に深層学習を行い，上級者・中級者・初級者の評価を行う部分について説明する．この部分はさらに大きく 3 つの段階に分けることができる．まず，入力されたグラフのノードをベクトルに変換し，初期状態を得る．次に，ノード

図 3: 代入文 n=1 を ANTLR で構文木に変換したもの

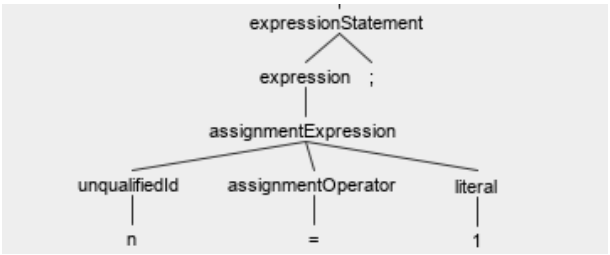


図 4: 代入文 n=1 の構文木から冗長なノードを削除したもの

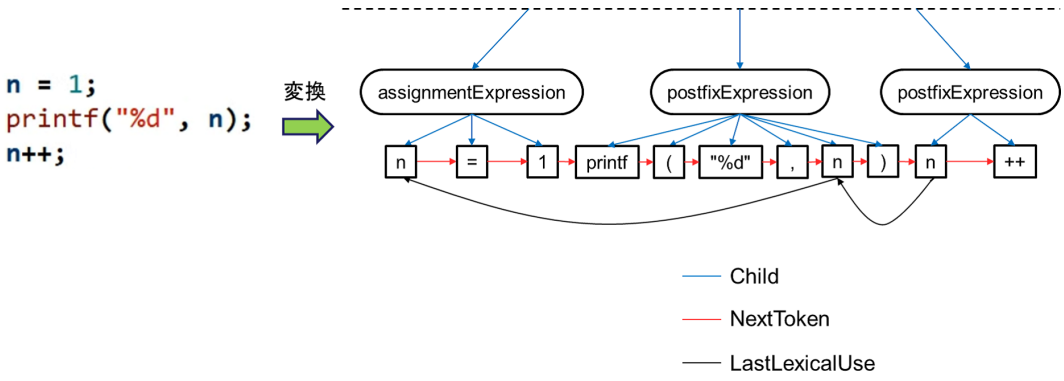


図 5: ソースコードと変換後のグラフ

の状態とエッジからグラフニューラルネットワークによりノードの状態を更新していく．最後に，ノードの状態から上級者・中級者・初級者の確率を出すことで能力の評価を行う．

3 つの段階それぞれについて，以下で詳細を述べる．

エンベディング

グラフニューラルネットワークを利用するためには，まずノードをベクトルへ変換する必要がある．本手法ではソースコード中のトークンと構文木から得られる文法を表す節，そしてそれらを分割したものをノードとして扱っている．そこで，ノードを表す文字列に対して Character-level CNN[18] を適用することでノードの初期状態を取得する．

グラフニューラルネットワーク

グラフニューラルネットワークはグラフ構造を学習し各ノードの状態を更新していくことで，ノードの予測やエッジの予測，グラフレベルの予測などに利用することができる．本手法はソースコードをグラフに変換し，そのグラフに対してグラフレベルの予測を行う．予測の際，各ノードはエッジが接続している隣接ノードから情報を集約し，集めた情報をもとにノードの状態を更新する．この工程は繰り返し行われる．この集約・更新の方法の違いにより，様々な種類のグラフニューラルネットワークが存在する．本手法では Microsoft が公開している複数のグラフニューラルネットワーク [13] を検証し，最も精度が高かった RGCN[16] を採用した．RGCN は画像に対する深層学習に使われる畳み込みという技術をグラフニューラルネットワークに応用した GCN[5, 10] をさらにラベル付きのエッジを持つグラフに適用できるように拡張したものとなっている．

そして本手法では RGCN によりノードの情報を集約する際，終端のノードが始端のノードの情報を集めるだけでなく，始端のノードも終端のノードの情報を集めることができるよ

うにする．これは VARMISUSE タスク [1] でも採用されており，情報の伝達をより速くし，モデルの表現力を向上させることがわかっている．

能力の評価

RGCN によって得られたノードの最終状態から能力の評価を行う．

まず，各ノードの初期状態と最終状態のベクトルを連結させる．そして全ノードの重み付き和と重み付き平均を算出し，連結させる．この連結したベクトルを出力数 3 の MLP に入力し，得られる出力をされに Softmax 層に入力すると，3 つの値の合計値が 100%となるように上級者・中級者・初級者である確率が出力される．ここで例えば上級者の割合が最も高くなった場合，このソースコードを書いた開発者は上級者であると判定する．

この実装は Microsoft が公開しているグラフの回帰タスクの実装[7]を本手法で扱う 3 値分類に拡張したものとなっている．

[7] https://github.com/microsoft/tf2-gnn

4　手法の評価

この章では実験を行い提案手法を評価する．そこで，まず実験に利用したデータセットと比較対象のベースラインについて説明する．そして提案手法とベースライン手法で開発者のコーディング能力の評価を行い，結果の比較と考察を行う．

4.1　データセット

ここでは評価実験に用いたデータセットについて述べる．このデータセットは既存のソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法 [21, 22, 23, 24] で用いられているものであり，堤 [20] によって作成された．このデータセットはオンライン上で公開されている[8]．

データセットは，参加者が問題への回答として提出したソースコードと，言語やタイムスタンプ等の提出履歴情報データベースの 2 種類からなる．提出履歴情報データベースの内容は表 3 で，データセットの統計情報は表 4 で示している．また，本データは 2016/5/19〜2016/11/15 の期間に収集されており，ソースコードのファイル数は 1,644,636 である．プログラミングコンテストにおける提出は 1 つのソースファイルにまとめられるため，これは提出数と一致している．参加者数は，2016/5/19〜2016/11/15 の期間に 1 度以上 Codeforces のコンテストに参加したユーザーの総数である．また，各コンテストには複数の問題が含まれるため，コンテストの数と比較して問題数が多くなっている．

[8] https://sites.google.com/site/miningprogcodeforces/

表 3: データセットの内訳

テーブル名	カラム名	キー	説明	データ型
Participant	user_name	PK	Codeforces の参加者名	String
	rating		参加者の現在のレーティング	Integer
	max_rating		2016/11/15 までの最大到達レーティング	Integer
Participant-Submission	user_name	PK FK	*Participant* テーブルにおける *user_name*	String
	files		データセット内における *user_name* の提出数	Integer
File	file_name	PK	データセット上でのソースファイル名	String
	submission_id		Codeforces における提出 ID	Integer
	lang		ソースファイルのプログラミング言語	String
	user_name	FK	ソースファイルの提出者	String
	verdict		提出の正誤	String
	timestamp		提出時刻	Date/Time
	competition_id	FK	*Competition* テーブルにおける *competition_id*	Integer
	problem_id		この提出に対応する *problem_id*	String
	url		Codeforces におけるこのソースファイルの URL	String
	during_competition		この提出がコンテスト時間内に提出されたかどうか	Boolean
Competition	competition_id	PK	コンテスト ID	Integer
	name		コンテスト名	String
	start_time		コンテスト開始時刻	Date/Time
	duration_time		コンテスト時間	Integer
	participants		コンテスト参加者数	Integer

次ページへ続く

前ページからの続き

テーブル名	カラム名	キー	説明	データ型
Problem	problem_id	PK	問題 ID	String
	competition_id	FK	この問題が掲載されたコンテストの *competition_id*	Integer
	prob_index		Index of the problem in the competition	String
	points		コンテストにおけるこの問題の得点	Integer
Acceptance	problem_id	PK FK	対応する問題の *problem_id*	String
	submission_in_sample		データセット上におけるこの問題に対する提出数	Integer
	solved_in_sample		データセット上におけるこの問題に対する正答数	Integer
	submission		この問題に対する全提出数	Integer
	solved		この問題に対する全正答数	Integer
	acceptance_rate		*solved/submissions*	Double
	lastmodied		データの最終更新日	Date/Time
Problem-Submission-Statistics	problem_id	PK	対応する *problem_id*	String
	filesize_max		この問題に対する提出ファイルサイズの最大値	Integer
	filesize_min		この問題に対する提出ファイルサイズの最小値	Integer
	filesize_mean		この問題に対する提出ファイルサイズの平均値	Double
	filesize_median		この問題に対する提出ファイルサイズの中央値	Integer
	filesize_variance		この問題に対する提出ファイルサイズの分散	Double
	filesize_max_competition		*filesize_max* のうちコンテスト中に提出されたもの	Integer

次ページへ続く

前ページからの続き

テーブル名	カラム名	キー	説明	データ型
	filesize_min_competition		*filesize_min* のうちコンテスト中に提出されたもの	Integer
	filesize_mean_competition		*filesize_mean* のうちコンテスト中に提出されたもの	Double
	filesize_median_competition		*filesize_median* のうちコンテスト中に提出されたもの	Integer
	filesize_variance_competition		*filesize_variance* のうちコンテスト中に提出されたもの	Double
Submission-Distance	file_name	PK	対応するソースファイル名	String
	proble_id		この提出に対応する *problem_id*	String
	next_file		*file_name* の次の提出	String
	submission_index		同じ問題に対してこの提出が何番目の提出か	Integer
	levenshtein_distance		*file_name* と *next_file* とのトークンベースの編集距離	Integer
	add_node		*file_name* から *next_file* にかけての追加ノード数	Integer
	delete_node		*file_name* から *next_file* にかけての削除ノード数	Integer
	update_node		*file_name* から *next_file* にかけての更新ノード数	Integer
	move_node		*file_name* から *next_file* にかけての移動ノード数	Integer
	node_sum		追加， 削除， 更新， 移動ノード数の合計	Integer

以上

表 4: 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

4.1.1 ソースコードデータ

ソースコードは，プログラミングコンテストの参加者が問題に対する回答として提出したものであり，コンパイル環境が用意されている任意の言語を提出することができる．また，回答ソースコードが正答であるか，コンパイル可能かどうかは提出の可否に影響しないため，文法的に不完全なソースコードが含まれる場合もある．

4.1.2 提出履歴データ

提出履歴情報には，参加者のレーティング情報や，どの参加者がどの問題にいつ提出したか，提出が正解であったか等の情報が含まれている．データベースの構成は表 3 に示す通りであり，以下では本研究で利用したテーブルである，Participant と File の詳細について述べる．

- **Participant** このテーブルには，2016/5/19〜2016/11/15 の期間中 1 度以上 Codeforces で開催されたプログラミングコンテストに参加したユーザーの情報を含む．各参加者情報には表 3 に示す通り 3 種類の項目が含まれる．Codeforces は一意のユーザー ID を提供しておらず，本データセットにおいてはユーザー名を ID としてある．また，レーティングはコンテストごと変化するが，本データに含まれるレーティングは 2016/11/15 時点のものである．
- **File** 2016/5/19〜2016/11/15 の期間中に Codeforces に対して提出されたソースコードの情報を収集したテーブルである．このテーブルには，ソースコードデータの総数と同じ 1,644,636 のデータを含む．各提出履歴に与えられた一意の提出 ID と提出対象である問題の ID をもとに対応するソースコードの URL を構築することができ，url カラムに格納されている．

4.1.3 統計情報

本データセットにおける，ソースコードの言語別提出数の割合を図 6 に示す．提出されたソースコードのうち，90%は C++によって記述されている．

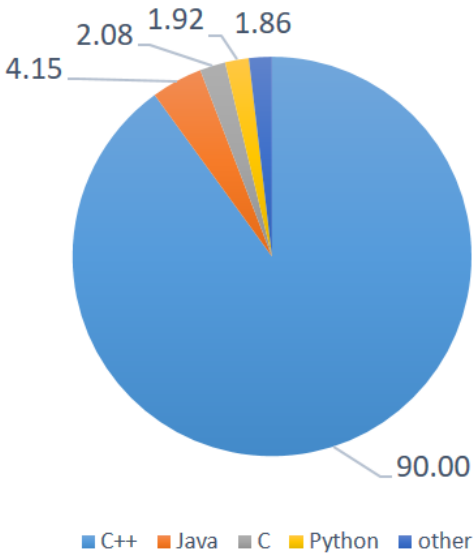


図 6: 提出ソースコードの言語分布

4.2 ベースライン

実験には既存のソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法とソースコードを単語毎に分割して自然言語のように扱った深層学習による手法 [9] を比較に利用する．

ランダムフォレスト

メトリクスを利用する機械学習による手法は 2.3 節で 2 つ説明したが，ここでは槇原の手法 [21, 22] に改良を加えたもの [23, 24] を利用する．

LSTM

ソースコードを自然言語のように扱った深層学習による手法では，始めにソースコードを単語ごとに分割し，それぞれの単語をベクトルに変換する．そしてソースコードを単語列のベクトルとして深層学習モデルに入力して評価を行う．既存研究では CCN や LSTM そしてその両方を組み合わせたモデルで実験を行っていたが，本研究のベースラインにはその実験で最も精度が高かった LSTM によるモデルを利用する．また，2 値分類による評価が行われていたが，本研究では上級者・中級者・初級者の 3 値分類による評価を行うため，出力層を変更する．そこで GitHub 上で公開されている実装[9]を確認したところ，以下の 2 つの問題点が存在した．

[9] https://github.com/Akhtar-Munir/Developer-level-detect

- 単語の分割をスペースとタブ，改行，コンマの有無で判断している
- コメントアウトされている部分も単語をベクトルに変換して学習する

まず単語の分割が不適切であることについて説明する．単語の分割を上で示したように行うと，例えば printf(”a b”); のようなソースコードの内の 1 行は printf(”a という部分と b”); という部分の 2 つを単語として分けてしまう．しかし，ソースコードの最小単位であるトークンで分割する場合は printf，(，”a b”，)，; という部分に分かれるため，この分割方法は適切であるとは言えない．そのため，実験を行う際にはソースコードをトークンで分割して学習を行うように変更する．

次に，コメントアウトされている部分の学習が不適切であることについて説明する．まず，この手法では単語をベクトルへ変換した後，全ての単語を深層学習へ入力するのではなく，ハイパーパラメータで指定された 251 個目以降の単語は入力しない．また，ソースコードの始めにはコピーライト等がコメントとして記述されることもある．実際に本研究で利用するデータセット内に図 7 や図 8 に示すようなソースコードも存在した．このようなソースコードをコメントを含めて学習してしまうと，ソースコードの処理を行う部分に焦点を当てた学習ができなくなる可能性があり，コメントアウトの部分を含めた学習は適切であるとは言えない．そのため，実験を行う際にはコメントアウトされている部分を無視するように変更する．

```
#include <bits/stdc++.h>
/*
    5ak0 5ak0 5ak0  5ak0 5ak0 5ak0  5ak0     5ak0  5ak0 5ak0 5ak0
    5ak0            5ak0      5ak0  5ak0   5ak0    5ak0       5ak0
    5ak0 5ak0 5ak0  5ak0 5ak0 5ak0  5ak0 5ak0      5ak0       5ak0

              5ak0  5ak0      5ak0  5ak0   5ak0    5ak0       5ak0
    5ak0 5ak0 5ak0  5ak0      5ak0  5ak0     5ak0  5ak0 5ak0 5ak0
*/
using namespace std;
```

図 7: 開発者名がコメントで記述されている例

4.3 実験内容

提案手法とベースライン手法で開発者のコーディング能力の評価を行う．

まず，データセットのソースコードを上級者・中級者・初級者のコードに分類する．ここでは既存の手法 [23, 24] と同様に最も提出数の多かった C++のソースコードを提出者のレーティングでソートし，上位 25%を上級者，下位 25%を初級者，残りを中級者のコードとしている．この時，上級者・中級者・初級者のレーティングの分布は表 5 のようになった．

```
// Copyright (C) 2016 Sayutin Dmitry.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License as
// published by the Free Software Foundation; version 3

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program; If not, see <http://www.gnu.org/licenses/>.

```

図 8: コピーライトとライセンスがコメントで記述されている例

表 5: 上級者・中級者・初級者のレーティングの統計情報

		初級者	中級者	上級者
レーティング	平均	1180.22	1459.66	1944.35
	分散	13170.37	10153.27	46307.75
	最小値	-39	1311	1711
	中央値	1211	1434	1902
	最大値	1310	1710	3367
人数		3899	8409	2212
ファイル数		353,346	701,871	352,557

そして提案手法とベースライン手法それぞれでこれらのソースコードに対して学習・評価を行う．提案手法及び深層学習を利用した手法 [9] は学習データを全体の 8 割，評価データを全体の 1 割，テストデータを残りの 1 割して分割したときのテストデータの評価値を出す．機械学習による手法 [23, 24] は学習データを全体の 9 割，テストデータを残りの 1 割として 10 分割交差検証を行い評価値を出す．評価値には accuracy と F 値の 2 つを採用した．提案手法はソースコードの構造や変数の意味情報を表すエッジの有効性を評価するために NextToken エッジの制限や LastLexicalUse エッジの制限，そしてその両方のエッジを制限した学習の評価も行う．

利用するソースコードに対して提案手法のグラフ変換を行った際に得られた構文的な情報と変数の意味的な情報を表す Child エッジ，NextToken エッジ，LastLexicalUse エッジの 1 ファイル当たりの数を表 6 に示す．

表 6: 各エッジの 1 ファイル当たりのエッジの数

エッジの種類	エッジ数
Child エッジ	656
NextToken エッジ	347
LastLexicalUse エッジ	60

4.4 実験結果

4.3 節で説明した実験の結果を表 7 と表 8 に示す．

表 7: 提案手法とベースライン手法の結果

モデル	上級者 F 値	中級者 F 値	初級者 F 値	accuracy
提案手法	0.891	0.881	0.835	0.871
ランダムフォレスト	0.758	0.793	0.699	0.762
LSTM	0.770	0.768	0.697	0.750

表 8: 提案手法のエッジを制限した学習の結果

制限したエッジ	上級者 F 値	中級者 F 値	初級者 F 値	accuracy
NextToken	0.882	0.874	0.827	0.863
LastLexicalUse	0.888	0.876	0.829	0.866
NextToken+LastLexicalUse	0.868	0.860	0.804	0.847

4.5 結果の比較・考察

まず表 7 から，提案手法は上級者の F 値が 0.891，中級者の F 値が 0.881，初級者の F 値が 0.835，accuracy が 0.871 となっており，全ての評価値でベースライン手法を上回っている．そのため，ソースコードの構文的情報と変数の意味的な情報によりコーディング能力を効果的に学習できたことがわかる．

また，表 8 の提案手法から NextToken エッジと LastLexicalUse エッジを制限した学習は表 7 のベースライン手法に比べて各評価値が高く，大きく精度が向上している．このことから，構文木から得られる構文の情報は非常に有用だと考えられる．

そしてこの 2 つのエッジの両方を制限した学習はどちらか片方を制限した学習に比べて全ての評価値が低くなっている．NextToken エッジを制限した学習と LastLexicalUse エッジを制限した学習の評価値は多少の差はあるが，両方を制限した学習に比べて各評価値は同じ程度向上している．表 6 に示したように，1 ファイル数当たりのエッジ数は LastLexicalUse エッジは NextToken エッジの 1/5 以下となっている．このように LastLexicalUse エッジは少ない数で NextToken エッジと同等の精度向上があるため，変数の意味的情報はコーディング能力の評価において非常に有用だと考えられる．

5　まとめ

本研究では，ソースコードの構文的情報や変数の意味的情報をグラフとして表現し，グラフを対象とする深層学習の 1 つである RGCN によりコーディング能力を評価する手法を提案した．そしてプログラミングコンテストのデータセットを利用して提案手法の評価を行ったところ，提案手法は従来のソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法やソースコードを単語毎に分割して自然言語のように扱った深層学習による手法に比べて良い結果が得られた．また，グラフのエッジを制限した学習の評価も行うことでエッジの有用性も確認した．

今後の課題としては汎化性能を確認することが考えられる．本研究ではデータセットにプログラミングコンテストを利用した．そのため，この手法がプログラミングコンテスト以外のドメインにおいて有用かどうかは判明していない．プログラミングコンテストではラベル付けに客観的な指標であるレーティングを利用することができた．しかし，他のドメインでレーティングのような客観的なコーディング能力の指標を得ることは難しいのではないかと考えられる．データセットを作成することができれば汎用性の確認だけでなく，汎用性の高い手法の作成も行うことができるようになる．

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指導及び御助言を賜りました．井上 教授に心より深く感謝いたします．

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，研究の方針から本論文の執筆に至るまで，直接の御指導及び御助言を賜りました．松下 准教授に心より深く感謝いたします．

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，研究活動において適切な御助言を賜りました．神田 助教に心より深く感謝いたします．

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 春名修介 特任教授には，研究活動において適切なご助言を賜りました．春名 特任教授に心より深く感謝いたします．

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 藤原裕士 氏には，深層学習や構文解析ツールに関する貴重な知見を共有していただきました．藤原 氏に心より深く感謝いたします．

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 福家範浩 氏には，深層学習や実験環境の構築に関する貴重な知見を共有していただきました．福家 氏に心より深く感謝いたします．

最後に，その他様々な御指導及び御助言を頂い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深く感謝いたします．

参考文献

[1] Miltiadis Allamanis, Marc Brockschmidt, and Mahmoud Khademi. Learning to represent programs with graphs. In *International Conference on Learning Representations*, 2018.

[2] Sebastian Baltes and Stephan Diehl. Towards a theory of software development expertise. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 187–200. Association for Computing Machinery, 2018.

[3] Martin S. Bressler. Building the winning organization through high-impact hiring. *Journal of Management and Marketing Research*, Vol. 15, 2014.

[4] Elizabeth Dinella, Hanjun Dai, Ziyang Li, Mayur Naik, Le Song, and Ke Wang. Hoppity: Learning graph transformations to detect and fix bugs in programs. In *International Conference on Learning Representations*, 2020.

[5] David Duvenaud, Dougal Maclaurin, Jorge Aguilera-Iparraguirre, Rafael Gómez-Bombarelli, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P. Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems*, Vol. 2, pp. 2224–2232, 2015.

[6] Arpad E Elo. *The Rating of Chessplayers, Past and Present.* Arco Pub., 1978.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

[8] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pp. 56–65. Association for Computational Linguistics, 2018.

[9] Farooq Javeed, Ansar Siddique, Akhtar Munir, Basit Shehzad, and Muhammad I.U. Lali. Discovering software developer's coding expertise through deep learning. *IET Software*, Vol. 14, No. 3, pp. 213–220, 2020.

[10] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

[11] Gayle Laakmann McDowell. *Cracking the coding interview: 189 programming questions and solutions*. CareerCup, 2015.

[12] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemeli. Gated graph sequence neural networks. In *International Conference on Learning Representations*, 2016.

[13] Microsoft. tf-gnn-samples. `https://github.com/microsoft/tf-gnn-samples`, 2019.

[14] Mikhail Mirzayanov. Codeforces rating system. `http://codeforces.com/blog/entry/102`, 2010.

[15] Arghavan Moradi Dakhel, Michel C. Desmarais, and Foutse Khomh. Assessing developer expertise from the statistical distribution of programming syntax patterns. In *Proceedings of the Evaluation and Assessment in Software Engineering*, p. 90–99. Association for Computing Machinery, 2021.

[16] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne vanÂ den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer International Publishing, 2018.

[17] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. Measuring and modeling programming experience. *Empirical Softw. Engg.*, Vol. 19, No. 5, pp. 1299–1334, 2014.

[18] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, Vol. 1, pp. 649–657, 2015.

[19] Yaqin Zhou, Shangqing Liu, J. Siow, Xiaoning Du, and Yang Liu. Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks. In *Advances in Neural Information Processing Systems*, pp. 10197–10207, 2019.

[20] 堤祥吾. プログラミングコンテスト初級者・上級者におけるソースコード特徴量の比較. 大阪大学大学院情報科学研究科修士論文, 2018.

[21] 槇原啓介. ソースコード特徴量を用いた機械学習によるソースコードの良否の判定. 大阪大学基礎工学部情報科学研究科卒業論文, 2019.

[22] 槇原啓介, 松下誠, 井上克郎. ソースコード特徴量を用いた機械学習によるソースコード品質の評価手法. 電子情報通信学会技術研究報告, Vol. 119, No. 113, pp. 105–110, 2019.

[23] 松井智寛. 判定対象の拡大を目的とした 3 値分類によるソースコードの良さの判定手法. 大阪大学基礎工学部情報科学研究科卒業論文, 2020.

[24] 松井智寛, 松下誠, 井上克郎. 判定対象の拡大を目的とした 3 値分類によるソースコード品質の評価手法. 情報処理学会研究報告, Vol. 2020-SE-205, No. 7, pp. 1–8, 2020.