Vol.2021-SE-207 No.27
2021/3/2

プログラム自動生成に向けたソースコード検索器の性能評価

沖野 健太郎[1,a)]　松尾 春紀[1,b)]　山本 大貴[1,c)]　亀井 靖高[1,d)]　鵜林 尚靖[1,e)]

概要：近年の IT 社会の発展によって IT 人材の不足が深刻になり，ソフトウェア開発の自動化が求められるようになった．実装の自動化において効果的なプログラム自動生成には工夫が必要であると考えられ，その工夫の一つとしてプログラム自動生成の過程でソースコード検索器を使用している先行研究がある．その研究では，競技プログラミングコンテストの問題文から解答ソースコードを自動生成するタスクにおいて，ソースコード検索器を用いることで解答ソースコードの生成件数が 92 件のうち 35 件から 40 件に増加したと報告されている．この手法におけるソースコード検索では，プログラム自動生成の部品となる類似ソースコードの検索性能が自動生成の精度に影響すると考えられる．本研究では，プログラム自動生成を目的としたソースコード検索器に着目した．複数のソースコード検索器を作成して競技プログラミングコンテストの問題に対して検索を行い，プログラム自動生成の観点から検証を行うことで，プログラム自動生成の精度向上への貢献を目指す．調査の結果，プログラム自動生成において TF-IDF 値を用いた検索器より深層学習モデルを用いた検索器が適していること，学習データにおける類似度の分布の影響は大きくないことを示した．

キーワード：自動プログラミング，コード検索，深層学習

1. はじめに

近年の IT 社会の発展により，今後 IT 人材の不足が深刻化すると予想されている．経済産業省によると 2015 年時点では IT 人材が約 17 万人不足していたとされ，2030 年には不足が約 79 万人に拡大すると述べられている[*1]．ソフトウェア開発の自動化を行うことは，今後の IT 社会の発展には重要であると考えられる．

ソフトウェア開発の自動化の中でもソフトウェアの実装自体を自動化することは難易度が高く，自動化の対象が限定的なものやソフトウェアの実装支援を目的とした研究が行われている．そのため，効果的なプログラム自動生成を行うには工夫が必要であると考えられる．

倉林ら [1] の研究では，ソフトウェア開発における実装の自動化への第一歩として，競技プログラミングコンテストサイトである AtCoder の問題文から解答ソースコードの自動生成を行っている．この研究の特徴としては，プログラム自動生成の過程において，クエリ問題文に類似する過去の問題の解答ソースコードをソースコード検索器を用いて検索している点が挙げられる．検索された類似問題をプログラム自動生成の雛形とすることで，ソースコード検索器を用いない場合と比較して自動生成できる件数が 92 件のうち 35 件から 40 件に増加したと報告されている．この手法におけるソースコード検索では，プログラム自動生成の部品となる類似ソースコードの検索性能が自動生成の精度に影響すると考えられる．

本研究では，プログラム自動生成におけるソースコード検索器に着目した．複数のソースコード検索器を用いて調査を行い，プログラム自動生成の観点からその優劣を評価する．プログラム自動生成に適したソースコード検索器の条件について実験的に調査を行うことで，ソースコード検索器を使用したプログラム自動生成の精度の向上に貢献することを目指す．

本稿では，2 節で研究の背景と目的について述べる．3 節では調査における実験の設計について述べる．4 節では提案した調査課題について実験を行い，その結果について考察する．5 節では妥当性に対する脅威について述べ，6 節でまとめを行う．

[1] 九州大学
Kyushu University
[a)] okino@posl.ait.kyushu-u.ac.jp
[b)] matsuo@posl.ait.kyushu-u.ac.jp
[c)] h.yamamoto@posl.ait.kyushu-u.ac.jp
[d)] kamei@ait.kyushu-u.ac.jp
[e)] ubayashi@ait.kyushu-u.ac.jp
[*1] https://www.meti.go.jp/

2. 背景と目的

2.1 ソースコード検索器を用いたプログラム自動生成

倉林ら [1] の研究では，ソフトウェアの実装の自動化への第一歩として，競技プログラミングコンテストサイトである AtCoder の問題文から解答ソースコードの自動生成を行っている．プログラム自動生成自体は遺伝的アルゴリズムを用いて行われ，問題の入出力例を満たすようなプログラムを合成している．この遺伝的アルゴリズムのベースとなる初期生成個体の選定において，ソースコード検索器が利用されている．

ソースコード検索器は自動生成を行いたい問題の問題文をクエリとし，過去に開催されたコンテストの問題で構成されているデータベースから最も類似していると予測される問題の解答ソースコードを検索する．類似問題の解答ソースコードを遺伝的アルゴリズムの初期生成個体とすることで正解となるソースコードまでの合成に必要な手順が少なくなるため，プログラム自動生成を行う上で有利となると考えられる．論文では，ソースコード検索器を用いることでプログラム自動生成に成功した件数が 92 件のうち 35 件から 40 件に増加したと報告されている．

2.2 情報検索を用いたソースコード検索

Sourcerer[2] や CodeHow[3] を始めとする従来のソースコード検索手法は情報検索（Information retrieval）に基づいているものが多い．情報検索はデータベースに蓄積されている検索対象となるデータに対するメタデータを検索アルゴリズムによって選択する検索手法である．

Gu ら [4] によると，情報検索に基づいたソースコード検索が抱える問題として，自然言語で記述されたクエリに含まれる高レベルの意図とソースコードに含まれる低レベルの意図との不一致が存在すると述べられている．具体的な理由の一つに，情報検索に基づいた検索では類義語を検索することが困難であるという点が挙げられている．例えば，“read” や “object” という単語が含まれるクエリに対し，ユーザが求めている挙動をするソースコードでも “load” や “instance” などの類義語で構成されたソースコードは検索できない可能性がある．また，クエリ内の無関係なキーワードやノイズを効果的に処理できないという問題点が存在するとも述べられている．自然言語とソースコードの間には意図の乖離があり，情報検索を用いたソースコード検索には限界があると考えられる．

2.3 深層学習を用いたソースコード検索

深層学習を用いたソースコード検索では，従来の情報検索に基づいたソースコード検索で問題となっていた自然言語とソースコード間の意図の乖離の解消を期待できる．自然言語処理分野で深層学習を行う際に広く用いられている Embedding 層を用いた埋め込みの場合，“read” という単語の埋め込みベクトルを [3.0, -1.2, 2.5] とすると，“load” は [3.0, -1.3, 2.4] のように近距離に埋め込まれるため，検索対象のソースコードに類義語が含まれる場合でも検索結果に出現することを期待できる．また，自然言語であるクエリ文字列とソースコードをそれぞれ数値ベクトルという共通した要素に変換することで，コサイン類似度という指標を用いてそれぞれの類似度を比較することが可能となる．この手法は Joint Embedding と呼ばれ，異なる種類のデータ同士の関連度を学習する際に用いられる [5].

プログラム 1 “reverseArray” と予測されるメソッドの例

```
1 String[] f(final String[] array) {
2     final String[] newArray = new String[array
          .length];
3     for (int index = 0; index < array.length;
          index++) {
4         newArray[array.length - index - 1] =
              array[index];
5     }
6     return newArray;
7 }
```

DeepCS[4] は深層学習を用いたソースコード検索の代表的な研究の一つである．GitHub 上の Java プロジェクトに含まれる関数とドキュメンテーション文字列を入力として学習を行い，プログラム技術関連のコミュニティである Stack Overflow に投稿された実際のクエリに対して深層学習を用いたソースコード検索器の検証を行っている．検証結果を従来研究である CodeHow[3] などの情報検索に基づくソースコード検索器による検索結果と比較して評価値が高いことを示し，深層学習を用いたソースコード検索の有用性を評価している．また，クエリに対するベストマッチなソースコードに対する検索精度だけでなく，開発者の役に立つと考えられる類似したソースコードの検索精度についても検証を行っている．

2.4 AST を用いた分散表現の獲得

code2vec[6] は効果的なソースコードの分散表現を獲得するための手法の提案を行っている．この研究では，ソースコードの抽象構文木から AST パスを抽出して深層学習モデルへ入力することで，ソースコードの構造的な情報を効果的に獲得できると述べられている．

具体的な例として，配列を逆順に並べ替える Java メソッド（プログラム 1）からメソッド名を予測するタスクが紹介されている．このソースコードには “reverse” という単語や類義語は含まれていないにもかかわらず，“reverseArray” というメソッド名の予測値が最大化される．AST パスを用いることにより，プログラム内で逆順の配列を作成する構造の特徴を学習できていると考えられる．ソースコード

の構造情報を用いない深層学習モデルの場合，このような予測を行うことは難しい．

このメソッド名予測タスクを含むいくつかの検証結果をASTパスを利用していない従来の深層学習モデルで行った場合の結果と比較し，ASTパスを用いたソースコードの分散表現の獲得の有用性を評価している．また，論文中ではASTパスを用いた分散表現の獲得手法は，ソースコード検索器へも応用可能であると述べられている．

code2seq[7] はcode2vecを発展させたもので，code2vecと同様に効果的なソースコードの分散表現を獲得する手法を提案している．code2vecが既存の各メソッド名に対する確率をベクトルとして出力していたのに対し，code2seqはデコーダを用いたシーケンスを出力するようになっている．そのため，code2vecは既存のメソッド名しか予測できないという問題点があったのに対し，code2seqでは単語を組み合わせて新たなメソッド名を作り出すことができるようになっている．いくつかのタスクに対してcode2vecを含む従来手法との検証結果の比較を行い，code2seqを用いたソースコードの分散表現の獲得の有用性を評価している．

2.5 本研究の目的

ソースコード検索器を用いたプログラム自動生成手法では，ソースコード検索器自体の性能がプログラム自動生成の精度に影響すると考えられる．本研究では，先行研究で行われていたプログラム自動生成の一要素であるソースコード検索器に着目する．プログラム自動生成に適したソースコード検索器について調査することで，プログラム自動生成の精度向上のための知見を得る．

本研究ではプログラム自動生成に適したソースコード検索器について，2つの調査課題の調査を行う．調査内容は以下の通りである．

調査課題 1: 各ソースコード検索器はどの程度の性能を持つか.

調査にあたり，関数名を用いた検索器，ASTパスを用いた検索器，TF-IDF値を用いた検索器の3種類のソースコード検索器を作成した．本調査では，各ソースコード検索器に対して学習と検索を行い，それぞれの検索器の性能を調査する．この調査では，プログラム自動生成におけるソースコード検索器として有効な検索器の構成を明らかにする．

調査課題 2: 類似度の分布は検索精度にどのような影響を与えるか.

学習データの出力として期待される値（類似度）の計算式には閾値が含まれるため，この値によって学習データの類似度の分布が変化する．この調査課題では，類似度の分布を変化させてソースコード検索器の学習と検索を行い，ソースコード検索器の性能を調査する．この調査では，類似度の分布がソースコード検索器の検索精度にどのような影響を与えるのかを明らかにする．

プログラム 2 ABC035 の A 問題の解答ソースコード

```
W, H = map(int, input().split())
if 3 * W == 4 * H:
    print('4:3')
else:
    print('16:9')
```

プログラム 3 ABC152 の A 問題の解答ソースコード

```
N, M = map(int, input().split())
if N == M:
    print('Yes')
else:
    print('No')
```

3. 実験の設計

3.1 データセット

本研究では，競技プログラミングコンテストサイトであるAtCoderのデータを用いて調査を行う．過去に開催された初級者向けコンテストであるAtCoder Beginner Contest(ABC)のA問題183問を調査対象とした．各問題に対する解答ソースコードはPythonを用いて作成し，AtCoder上の全てのテストケースに通過することを確認した．各データは自然言語（英語）で記述された問題文と対応する解答ソースコードによって構成される．これらのデータによって構成されるデータセットのことを以降ABCデータと呼ぶ．

ABCデータの特徴として，問題文は類似していない場合でも解答ソースコードの構造が類似している組み合わせが存在することが挙げられる．例として，ABC035のA問題（プログラム2）とABC152のA問題（プログラム3）を示す．ABC035はテレビ画面の縦の長さHと横の長さWからアスペクト比を求める問題，ABC152はテストケースの数Nと通ったテストケースの数MからAC（正答）になるかを判定する問題となっており，問題文の観点からは両者は類似していないと考えられる．一方で解答ソースコードは，両者とも入力した2つの数値の比較結果により出力を行うというソースコードであり，構造的な観点からは類似していると言える．

3.2 実験の全体像

実験の全体像を**図 1** に示す．深層学習を用いた各ソースコード検索器に対する1回の実験は以下の流れで実施される．なお，図中の問題文A-コードBのように英字部分が異なる組み合わせのデータは，元々のデータとは異なる組み合わせであることを示している．

(1) 分割. ABCデータ183件からシード値を用いてランダムに10件のデータを取り出し，そのデータ中の

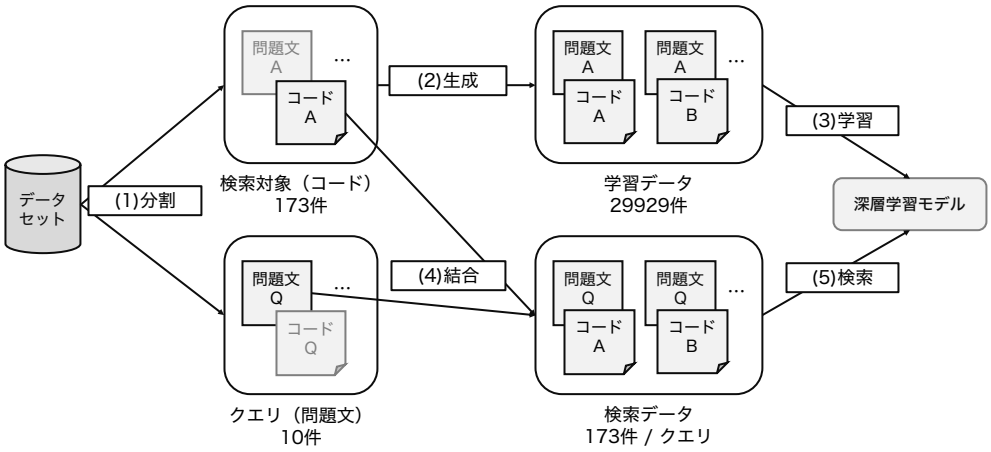

図 1 実験の全体像

問題文をクエリ用データとする．残った 173 件のデータのソースコードを検索対象データとする．

(2) 生成. 検索対象の 173 件のデータに対し，問題文とソースコードが異なる組み合わせのデータを生成する．元々の組み合わせのデータと生成された異なる組み合わせのデータを合わせて 29929($= 173^2$) 件のデータを各深層学習モデルの学習データとする．このとき，各問題文とソースコードの組み合わせに対し，後述する TED を用いて類似度を計算する．

(3) 学習. 学習データを用いて各深層学習モデルの学習を行う．深層学習モデルの入力は問題文とソースコード，出力ラベルには (2) で計算した類似度を用いる．ただし，前処理の方法や入力形式は各ソースコード検索器によって異なる．

(4) 結合. それぞれのクエリ用データの問題文に対し，検索対象データのソースコード 173 件との組み合わせデータを作成する．これを検索データとする．

(5) 検索. 検索データを深層学習モデルに入力し，問題文とソースコードの類似度を推測する．推測された類似度を降順に並べ替えたものを検索結果とする．

TED. TED(Tree Edit Distance，構文木編集距離)[8] は，2 つの木構造で表されるデータ同士の編集距離を表す指標である．この値が小さいほどソースコード同士の構造が類似していることを示す．本研究では，問題文とソースコード間の類似度 S を TED を用いた以下の式により計算を行う．

$$S(n,m) = Max(1 - TED(p_n, p_m)/T_{const}, 0) \qquad (1)$$

式中の p_n は問題文に対する正しいソースコード，p_m は問題文と組になってるソースコードを示す．例えば，問題文 A-ソースコード B の組み合わせのデータの場合は，p_n はソースコード A，p_m はソースコード B となる．T_{const} は閾値であり，TED がこの値を超えたものは類似度が 0 になる．この式により，問題文とソースコード間が類似している組み合わせは 1 に近い出力が得られ，そうでない組み合わせは出力が 0 に近くなるように学習される．

3.3 調査する検索モデル

本調査では 3 つの検索器を用いて調査を行う．先に述べる 2 つが深層学習を用いた検索器，最後に述べる 1 つが情報検索に基づいた検索器となっている．

3.3.1 関数名を用いた検索器

関数名を用いた検索器の深層学習モデル部分の概略図を**図 2** に示す．このモデルは，自然言語の入力を後述する ALBERT[9] を用いてベクトル化したものと，ソースコードに使用されている関数とその引数を LSTM などを用いてベクトル化したもの同士のコサイン類似度を出力する．コサイン類似度とは 2 つのベクトル A，B がどの程度同じ方向を向いているかを表す指標で，以下の式 2 により算出される．

$$cosine\_similarity(A, B) = \frac{A \cdot B}{|A||B|} \qquad (2)$$

コサイン類似度は [-1, 1] の範囲で表され，1 に近いほどベクトル A,B の類似度が高く，-1 に近いほど類似度が低い．深層学習では出力が式（1）で計算した [0, 1] の範囲を取るラベルの値に近くなるように学習を行うため，sigmoid 関数を用いて [0, 1] に正規化を行っている．

検索時は，クエリ文字列と検索対象のソースコードを入力して類似度を出力し，類似度が 1 に近いものから降順に並べ替えたものを検索順位とする．この検索モデルのことを以降 F2V(Function to Vector) 検索モデルと呼ぶ．

ALBERT. 本研究では，ソースコード検索器の自然言語入力部分に ALBERT の学習済みモデルを使用している[*2]．ALBERT は BERT[10] から派生した自然言語処理に用いられる深層学習モデルの一つで，その特徴の一つ

[*2] https://tfhub.dev/tensorflow/albert_en_base/2

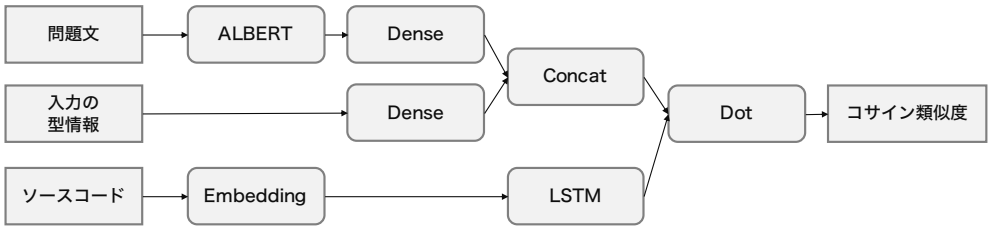

図 2 F2V 検索モデルの概略図

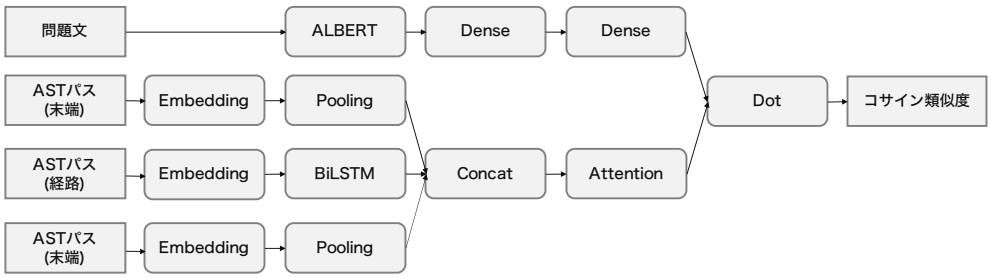

図 3 C2V 検索モデルの概略図

に Transformer[11] 構造を採用していることが挙げられる．Transformer は従来の RNN(Recurrent Neural Network) ベースの自然言語処理に比べて学習が高速で高精度であることが報告されている．

3.3.2 AST パスを用いた検索器

AST パスを用いた検索器の深層学習モデル部分の概略図を**図 3** に示す．このモデルは，自然言語の入力を ALBERT を用いてベクトル化したものと，後述する AST パス化を行ったソースコードの入力に対して LSTM などを用いてベクトル化したもの同士のコサイン類似度を出力する．

AST パスは末端部分 2 箇所と経路部分に分解して入力を行っている．それぞれの入力に対して埋め込みを行ったものを再び結合し，Attention を用いてそれぞれの AST パスに重みづけを行っている．AST パスに対して重みづけを行うことでソースコードの特徴を表すベクトルを効率的に学習できる．

検索時は，F2V 検索モデルと同様に，類似度が 1 に近いものから降順に並べ替えたものを検索順位とする．この検索モデルのことを以降 C2V(Code to Vector) 検索モデルと呼ぶ．

AST パス． 本研究では，C2V 検索モデルにソースコードを入力するための前処理として，AST パス化を行う．AST パス化とは code2vec や code2seq で用いられている手法であり，ソースコードを抽象構文木（Abstract Syntax Tree, AST）に変換し，ある末端ノードから別の末端ノードまでのパスの集合を取り出す手法である．AST パスを用いることで，従来の手法と比べてソースコードの構造的な意味を深層学習モデルにより多く学習させることができる．

図 4 に AST パス化の例を示す．まず，“print(1+2)” というプログラムに対し，Python 標準ライブラリである ast パッケージを用いて AST 化を行う（実際の出力は辞書型のリストになる）．この AST に対して末端ノードを 2 つ選んでそのパスを抽出する処理を全ての末端ノードの組み合わせに対して行う．この際，末端ノードの部分は “print” と “NameLoad” のように名称部分と動作部分に分けられて抽出される．この例の AST の末端ノードは 3 個であるため，この AST からは AST パスを ${}_3C_2 = 3$ 通り抽出できる．この AST パスを末端の名称部分と経路部分に分けてエンコードを行い，C2V 検索モデルへの入力とする．ただし，AST パスの数は末端ノード数の 2 乗に比例するため，ソースコードによっては膨大な数になる．そのため，学習に用いる AST パスの上限を 256 通りに定め，上限を超えるソースコードについては AST パスの集合からランダムに 256 通りを抽出したものとした．

3.3.3 TF-IDF 値を用いた検索器

深層学習を用いた 2 つの検索モデルとの比較として，情報検索の 1 つである TF-IDF(Term Frequency-Inverse Document Frequency) 値を用いたソースコード検索器を使用する．TF-IDF 値は，単語 t の文書 d における出現頻度（TF）と全文書に対する文書頻度の逆数（IDF）を掛け合わせた値で，この数値が高い単語はその文書の特徴を表していると言える．本調査では，検索対象のソースコードをトークンに分解し，それぞれの単語トークンに対して TF-IDF 値を計算した．検索時には，クエリ文字列に含まれるそれぞれの単語における TF-IDF 値の和を計算し，その値を降順に並べ替えたものを検索順位とする．TF-IDF 値は以下の式で求められる．N は全文書数，$n_{t,d}$ は文書 d における単語 t の出現回数，$df(t)$ は単語 t が出現する文書数を表す．

$$tf(t,d) \cdot idf(t) = \frac{n_{t,d}}{\sum_{s \in d} n_{s,d}} \cdot (\log \frac{N}{df(t)} + 1) \qquad (3)$$

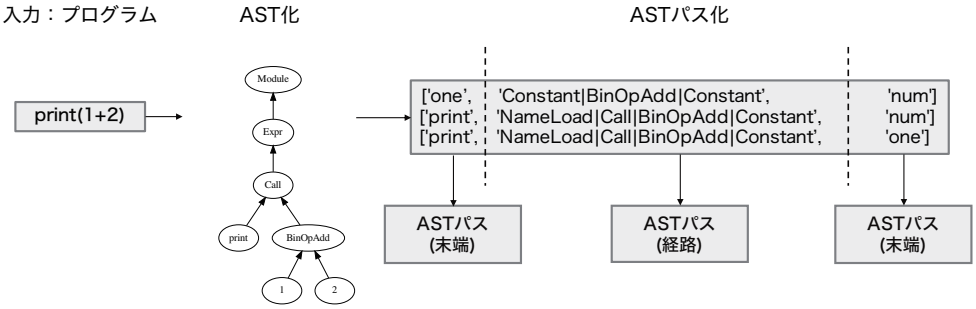

図 4 AST パス化の例

3.4 評価指標

各ソースコード検索モデルに対する検索精度を客観的に評価するため，以下の評価指標を用いて検索精度の評価を行う．なお，今回の調査における出力として最も期待されるソースコードとは，検索対象のソースコードのうち正解ソースコードとの類似度が最大（TED が最小）であるソースコードとする．その理由は，遺伝的アルゴリズムを用いてプログラム自動生成を行う際，TED が小さいほど正解ソースコードを合成するまでの手順が少なく，合成確率が高くなると考えられるからである．なお，正解ソースコードとの類似度が同じソースコードが複数存在する場合は，最も検索順位が高いソースコードを利用する．

MRR. MRR(Mean Reciprocal Rank) は，クエリ集合 Q に属する各クエリ q に対して, 出力として最も期待されるソースコードの順位の逆数の平均をとったものである．MRR は以下の式によって計算される．$Rank_q$ はクエリ q に対する出力として最も期待されるソースコードの順位を表す．MRR の値が大きいほど，検索器が出力として最も期待されるソースコードを上位に予測できていると言える．

$$MRR = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{Rank_q} \tag{4}$$

SuccessRate. SuccessRate は，クエリ集合 Q に属する各クエリ q に対して, 出力として最も期待されるソースコードが上位 k 位に含まれている割合を測定する．SuccessRate の値は以下の式によって計算される．δ は条件式が真のときは 1，偽のときは 0 を返す関数であり，この場合 $Rank_q$ が上位 k 位に入っている場合は 1 になる．SuccessRate の値が大きいほど，検索器が出力として最も期待されるソースコードを上位に予測できていると言える．ただし，k=1 などの小さい k の値を用いる場合は誤差が大きく出やすいため，今回の調査では k=10 を用いて評価を行う．以降 SuccessRate@10 のことを SR@10 と呼ぶ．

$$SuccessRate@k = \frac{1}{|Q|}\sum_{q=1}^{Q}\delta(Rank_q \leq k) \tag{5}$$

予測値 1 位の平均 TED. 上記 2 つの指標は出力として最も期待されるソースコードの順位を用いた指標であったが，実際にプログラム自動生成に用いられるのは（目視などを行わない場合）検索器によって 1 位であると予測されたソースコードとなる．よって，予測値 1 位のソースコードの類似度が高い（TED が小さい）かどうかは重要だと考えられる．予測値 1 位の平均 TED が小さいほど，その検索器はプログラム自動生成に適していると言える．以降予測値 1 位の平均 TED のことを MTED と呼ぶ．

4. 結果と考察

本節では，2 つの調査課題について調査内容と実験のアプローチ，その実験結果について述べ，考察を行う．

4.1 調査課題 1： 各ソースコード検索器はどの程度の性能を持つか.

調査内容. 各ソースコード検索器の学習と検索を行い，各評価指標の値を用いてソースコード検索器の検索精度を調査する．ソースコード検索器の構成の違いが検索精度にどの程度の影響を与えるかを明らかにする．

アプローチ. F2V 検索モデル，C2V 検索モデルのそれぞれについて，ABC データを用いて図 1 の要領で学習と予測を行う．その際に得られた検索結果をもとに，それぞれ MRR, SR@10，MTED の値による評価を行う．なお，学習データの類似度を計算する際の式 (1) 中の T_{const} の値は，目視においての類似度の分布の偏りが少ないと考えられた 48 とした．TF-IDF については，学習を行う代わりに検索対象のソースコードに含まれる各トークンの TF-IDF 値を計算する．その後，クエリ用データの問題文に含まれる単語の TF-IDF 値の和を降順に並べたものを検索結果とし，各検索モデルと同様に評価指標の値による評価を行う．クエリが 10 件と少なくデータが偏る可能性があるため，各ソースコード検索器による学習と検索はクエリ用データを変更してそれぞれ 3 回ずつ行い，得られた評価指標の値の平均値を求める．

結果と考察. 調査課題 1 の結果は**表 1** の通りである．乱数期待値は，ランダムに検索順位を決定した場合における

表 1 各検索器における評価指標の値

検索モデル	MRR	SR@10	MTED
F2V	0.080	0.200	21.70
C2V	0.087	0.133	22.40
TF-IDF	0.104	0.167	26.63
乱数期待値	0.033	0.058	29.35

表 2 T_{const} の値ごとの評価値

T_{const} の値	MRR	SR@10	MTED
32	0.156	0.200	23.47
48	0.087	0.133	22.40
64	0.028	0.067	24.23

それぞれの評価指標の期待値で，各評価指標の値を評価する上での参考値として表している．

TF-IDF 値による検索を F2V 検索モデルと比較すると，MRR は 0.104 と約 30%増加，SR@10 は 0.167 と約 16%減少，MTED は 26.63 と 4.93 増加した．C2V 検索モデルを F2V 検索モデルと比較すると，MRR は 0.087 と約 9%増加，SR@10 は 0.133 と約 33%減少，MTED は 22.40 と 0.70 増加した．

いずれの検索器においても乱数期待値よりも優れた値となっており，一定の検索精度があると言える．TF-IDF 値による検索においては，MRR や SR@10 の値が高くなっている一方で，MTED の値は F2V 検索モデルや C2V 検索モデルのほうが優れていることがわかる．この結果から，TF-IDF 値による検索は出力として最も期待されるソースコードに対しては一定の検索性能があるが，深層学習を用いた検索器と比較して TED が小さいものを 1 位として検索する性能は低いことがわかる．

TF-IDF 値による検索での MTED の値が低い理由としては，F2V 検索モデルと C2V 検索モデルが TED の値を用いて学習しているのに対し，TF-IDF 値を用いたソースコード検索は TED を利用していないことが要因だと考えられる．それにより，ソースコードの構造的な類似度を検索する性能が低くなった可能性がある．

F2V 検索モデルと C2V 検索モデルの MTED の値にあまり差が出なかった理由としては，ABC データはプログラム 2 やプログラム 3 のように類似した構造のものが多く，AST の特徴を十分に捉えられなかったことが考えられる．

> TF-TDF 値による検索は出力が最も期待されるソースコードに対する一定の検索性能があるが，深層学習を用いた検索のほうが予測値 1 位の平均 TED が 4.0 以上小さく，プログラム自動生成のためのソースコード検索器として適している．また，F2V 検索モデルと C2V 検索モデルの間には大きな差は見られなかった．

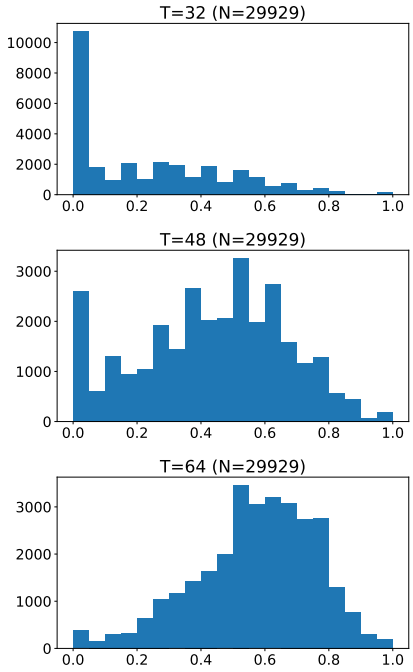

図 5 T_{const} の値に対する類似度の分布
（上から $T_{const} = 32, 48, 64$）

4.2 調査課題 2： 類似度の分布は検索精度にどのような影響を与えるか．

調査内容． C2V 検索モデルに対し，学習用データの類似度の分布を変えて学習を行い，検索精度を評価する．学習用データの類似度の分布が検索精度にどのような影響を与えるのかを明らかにする．

アプローチ． ABC データから学習用データを生成する際に，式 (1) 中の T_{const} の値を増減させて類似度の計算を行う．各 T_{const} の値に対する類似度の分布を**図 5** に示す．その学習データを用いて調査課題 1 と同様に C2V 検索モデルの学習を行い，検索結果をもとに MRR, SuccessRate@10, 予測 1 位の平均 TED の値を算出し，評価を行う．

結果と考察． 調査課題 2 の結果は**表 2** の通りである．

$T_{const} = 32$ の場合を $T_{const} = 48$ の場合と比較すると，MRR が 0.156 と約 79%の増加，SR@10 は 0.200 と約 50%の増加，MTED は 23.47 と 1.07 増加した．$T_{const} = 64$ の場合を $T_{const} = 48$ の場合と比較すると，MRR が 0.028 と約 68%の減少，SR@10 は 0.067 と約 50%の減少，MTED は 24.23 と 1.76 増加した．

この結果から，T_{const} の値が小さいほど MRR や SR@10 の値が小さくなっており，一方で MTED の値はあまり大きな差はないことがわかる．よって，類似度の分布は出力が最も期待されるソースコードを検索する性能への影響が大きい一方で，TED が小さいものを 1 位として検索する性能への影響はあまり大きくはないことがわかる．

MRR や SR@10 に差がでた理由としては，類似度の分布が偏ることで全体的な出力が 0 や 1 に近づくように学習されるようになったためであると考えられる．それにより，

出力が最も期待されるソースコードと他のソースコードの出力の差が $T_{const} = 32$ の場合は大きく，$T_{const} = 64$ の場合は小さくなった結果，出力が最も期待されるソースコードを検索する性能が変化した可能性がある．MTED に大きな差が出なかった理由としては，TED が小さいソースコードの出力が大きくなるように学習することは変わらず，今回の類似度の分布の偏りは学習の大きな障害になるほどの偏りではなかったためであると考えられる．

類似度の分布は出力が最も期待されるソースコードを検索する場合には一定の影響があるものの，プログラム自動生成に重要な予測値 1 位の平均 TED への影響はあまり大きくない．

5. 妥当性に対する脅威

5.1 内的妥当性

本研究ではクエリ用データが 10 件と少ないため複数回実験を行って評価指標の平均値を出したが，実験回数自体も各 3 回と十分であるとは言えないため，各評価指標の値に偏りがある可能性がある．

また，各深層学習モデルの学習を行う際は損失関数の最小値が 10epoc の間更新されなくなるまで学習を行ったため学習不足ではないと考えられるが，過学習についての検証は行っておらず，学習に偏りがある可能性がある．

5.2 外的妥当性

本研究では AtCoder のデータに対してのみ検証を行っており，他のデータに対する一般性は保証されていない．

また，AtCoder の各問題に対する解答ソースコードは自作を行ったが，解答ソースコードと言えるものは 1 通りではなく複数存在するため，これも一般性は保証されない．

6. おわりに

本研究では，プログラム自動生成に適したソースコード検索器について調査を行った．今回の結果から，(1) プログラム自動生成を目的としたソースコード検索において，深層学習を用いたソースコード検索器は TF-IDF 値を用いたソースコード検索器よりも優れていること，(2) 学習データの類似度の分布によってソースコード検索器の性能に一定の影響があるものの，プログラム自動生成への影響はあまり大きくないことを示せた．

本研究の結果から，プログラム自動生成に適したソースコード検索器の精度向上には検索器自体の構造の影響が大きいと考えられる．よって，深層学習モデルの中間層の構造とハイパーパラメータについての検討や深層学習を用いない検索方法について検討を行うことで，よりプログラム自動生成に優れたソースコード検索器を作成できる可能性があると考えられる．また，ABC データ以外のデータセットに対しても検討を行うことで調査結果を一般化することも重要であると考えられる．

謝辞 本研究の一部は JSPS 科研費 JP18H04097・JP18H03222，および，JSPS・国際共同研究事業の助成を受けた．

参考文献

[1] 倉林利行，吉村 優，切貫弘之，丹野治門，富田裕也，松本淳之介，まつ本真佑，肥後芳樹，楠本真二：深層学習と遺伝的アルゴリズムを用いたプログラム自動生成，ソフトウェアエンジニアリングシンポジウム 2020 論文集，pp. 143–152 (2020).

[2] Linstead, E., Bajracharya, S., Ngo, T., Rigor, P., Lopes, C. and Baldi, P.: Sourcerer: Mining and Searching Internet-Scale Software Repositories, *Data Min. Knowl. Discov.*, Vol. 18, No. 2, pp. 300–336 (2009).

[3] Lv, F., Zhang, H., Lou, J., Wang, S., Zhang, D. and Zhao, J.: CodeHow: Effective Code Search Based on API Understanding and Extended Boolean Model (E), *In Proc. of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 260–270 (2015).

[4] Gu, X., Zhang, H. and Kim, S.: Deep Code Search, *In Proc. of the 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 933–944 (2018).

[5] Xu, R., Xiong, C., Chen, W. and Corso, J. J.: Jointly Modeling Deep Video and Compositional Text to Bridge Vision and Language in a Unified Framework, *In Proc. of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 2346–2352 (2015).

[6] Alon, U., Zilberstein, M., Levy, O. and Yahav, E.: code2vec: Learning Distributed Representations of Code, *In Proc. of the ACM Program. Lang.*, Vol. 3, No. POPL (2019).

[7] Alon, U., Brody, S., Levy, O. and Yahav, E.: code2seq: Generating Sequences from Structured Representations of Code, *In Proc. of the International Conference on Learning Representations* (2019).

[8] Bille, P.: A survey on tree edit distance and related problems, *Theoretical Computer Science*, Vol. 337, No. 1, pp. 217 – 239 (2005).

[9] Zhenzhong, L., Mingda, C., Sebastian, G., Kevin, G., Piyush, S. and Radu, S.: ALBERT: A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS., *In Proc. of the International Conference on Learning Representations* (2020).

[10] Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, *In Proc. of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Vol. 1, pp. 4171–4186 (2019).

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u. and Polosukhin, I.: Attention is All you Need, *In Proc. of the Advances in Neural Information Processing Systems*, Vol. 30, pp. 5998–6008 (2017).