Vol.2023-SE-213 No.7
2023/3/9

機械学習を用いて模範コードを提示する初学者向けプログラミング学習システムの構築

北庄司 亮[1,a)]　松下 誠[1,b)]　肥後 芳樹[1,c)]

概要：プログラミングの学習において，単純に回答して結果を採点するだけではなく，学習者が自身の回答に対してフィードバックを受けることは重要である．しかし，プログラミングを始めたばかりの初学者は知識や経験が不足しているために，自分自身でコードをフィードバックすることが困難である．また，あるプログラムの問題に対する解答は複数考えられるため，他者が初心者の解答に合わせて適切にフィードバックすることも容易ではない．本研究では，既存研究を利用して入力したコードを改善すべきか判定した上で，改善が必要な場合は入力したソースコードに構造が類似した模範解答となるソースコードと，その要約を推薦するシステムを構築した．また被験者にシステムを利用してもらい，定量的な評価ならびに被験者へのアンケートを実施することによってシステムの評価を行った．

キーワード：プログラミング，機械学習，学習システム

1. はじめに

コンピュータはスマートフォンやパソコンなどの身近なものに内蔵されており，生活を豊かにするために重要な要素である．そのためコンピュータを効果的に利用することを目的としてプログラミング学習に対する需要が高まっている [12]．これらの内容を受け，文部科学省は小学校からプログラミング教育を必修化しており [13]，プログラミング学習者は増加している．また，プログラミング学習者の増加にともなって学習を支援する機会も増加する．

一般に，何かを学習する際には，インプット，アウトプット，フィードバックを受けることが重要である [9]．しかし，プログラムはアルゴリズムや設計の問題などが理由で，模範解答をひとつに絞れるとは限らないため，フィードバック先のソースコードが複数存在することもありうる．そのため適切にフィードバックすることは容易でない．

本研究では学習モデルを用いて入力したソースコードの構造に類似した模範解答コードを提示し，プログラミングの学習支援を行うシステムを構築する．このシステムは入力をソースコードとし，そのソースコードに改善の余地がある場合には入力したソースコードの構造と類似する模範解答コード，入力したソースコードの要約，模範解答コードの要約を提示するシステムである．

このシステムの評価を実験とアンケートにより行った．まず Project CodeNet のデータセットを基に問題とそれに対する模範解答コード群を用意し，被験者が問題を解いたあと，提案するシステムを用いて改善した場合と改善しない場合での改善状況を計測する．この計測結果によりシステムの有無による改善度の違いを評価した．その後被験者には改善しやすさに関するアンケート調査を行った．

以下，2 節では本研究の背景として，プログラミングコンテストとソフトウェアの評価指標，そしてシステムに用いた既存研究であるソースコードを自動評価する手法，ソースコードをベクトル化するための code2vec, ソースコードを要約するための code2seq について説明する．3 節では提案するシステムの設計について説明し，4 節ではその実装について説明する．5 節では提案するシステムを実験とアンケートによって評価を行った内容について説明し．最後に 6 節ではまとめについて述べる．

2. 背景

本節では研究の背景として，プログラミングコンテストとその採点に利用するオンラインジャッジシステム，データセットとして利用した Project CodeNet[*1]，ソフトウェアの品質指標，システムに用いた既存研究について説明

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Suita, Toyonaka 565–0871, Japan
[a)] ryo-kita@ist.osaka-u.ac.jp
[b)] matusita@ist.osaka-u.ac.jp
[c)] higo@ist.osaka-u.ac.jp

[*1] https://github.com/IBM/Project_CodeNet

する．

2.1 オンラインジャッジシステム

オンラインジャッジシステムとは，オンラインで演習問題に関するソースコードを自動採点するシステムである．プログラマが書いたプログラムをテストすることで自動的に評価し，正誤に関して判定できる．これによりプログラミングの能力を評価するためのオンラインテストやコンテストを開催することができる．採点基準は以下のようなものがある．

- コンパイルできるか
- 出力が適切か
- メモリ使用量
- 実行時間

このような基準に対して満たさない場合はその箇所を，すべて満たす場合は正解であることを示す．具体的なオンラインジャッジシステムとしては AIZU Online Judge[1], AtCoder[2]，Topcoder[3] などがある．

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を競うコンテストのことである. 参加者は，与えられた問題を解決するためのプログラムを記述し，その結果を評価するためにオンラインジャッジシステムに提出する．その後，正解した問題数などの様々な基準で参加者の順位を決定する．大学，プログラマ，企業など，様々な団体によって開催されており，参加者も個人で参加するものからチームで参加するものなど様々である．コンテストはオンラインで開催されることが多く，世界中の人々が参加できる．競技内容にも様々な種類があり，例えば以下のようなものがある．

- アルゴリズム，コンピュータサイエンス，数学に関する問題を解く早さを競う
- ソフトウェア，ネットワークのセキュリティ上の問題点の発見を競う
- ある処理を実行するソースコードの小ささを競う

本研究では，個人で参加し，アルゴリズム，コンピュータサイエンス，数学に関する問題を解く早さを競うコンテストを扱う．

2.3 Project CodeNet

Project CodeNet[14] は，ImageNet[7] が Computer Vision の分野に与えた影響を，AI for Code の分野にも与えることを目的としており，オンラインジャッジシステムである AIZU Online Judge と AtCoder[2] から収集された 4000 を超える問題とそれに対応して様々な言語で解答されたソースコードやメタデータがまとめられている．対象となる言語には Java，Python3，C, C++, Ruby などがあり，このデータセットはソースコードに関連する機械学習の訓練に対して使用されている [8]．データセットのメタデータは以下のようなものがある．

- 問題番号
- 参加者の id
- 解答日時
- プログラミング言語
- コンパイル情報
- 実行時間
- 使用メモリ量
- コードのサイズ
- プログラムの正誤

本研究では，プログラム言語，実行時間，プログラムの正誤に関する情報を利用し，模範解答コード群を作成する．

2.4 ソフトウェアの品質指標

ソフトウェアの品質を高く保つことは重要な要素である．品質には可読性，一貫性，拡張性，再利用性などの様々な要素があり，高品質なコードを作成することは高品質なソフトウェアの開発に役立つ．以下では評価に利用した指標について述べる．

2.4.1 Cyclomatic Complexity

Cyclomatic Complexity[16]（以下，CC）はプログラムの構造の複雑さに関する指標であり，この指標が高いほどプログラムは複雑で，バグが発生しやすくなることから保守性が低くなりやすい．制御フローグラフを解析し，そのグラフの辺の数に基づいて計算され,CC はグラフのエッジ数 E, グラフのノード数 N, 連結成分の個数 P を用いて次式で表される．

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

制御フローグラフは図 1 のように，プログラムの中のすべてのフロー制御構造を表したグラフで，ノードは逐次コード，辺は制御フロー（if 文，for 文，while 文など）を用いる．

```
public class Main {
  public static void main(String[] args) {
    if(args.length > 0) System.out.print(args[0]);
  }
}
```

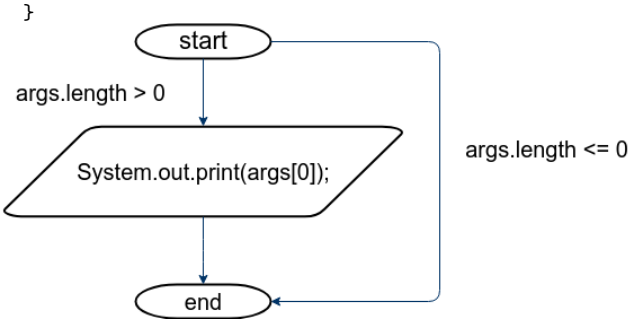

図 1 制御フローグラフの例

2.4.2 Lines of Code

Lines of Code（以下，LOC）はソフトウェア開発の規模や複雑度を評価するために使用される指標であり，プログラムの行数をコメントや空行も含めて数えたものである．LOC はプログラミング言語や開発環境に依存するため，この指標だけでソフトウェアの質を決定することができない．そのため，他の指標と組み合わせて使用する．

2.5 システムに利用した既存研究

2.5.1 品質の自動評価

松井ら [11] は堤ら [17] によって作成されたプログラミングコンテストのデータセット[*2]を用いて，3 値分類によるソースコードの良さを判定するモデルを作成している．判定するまでの流れ（図 2）で示すように，大きく分けて 3 段階になっている．

まずデータセットとなるソースコード群を上級者コード群・中級者コード群・初級者コード群で分割する．このとき，上級者・中級者・初級者の定義は解答者のレートを基準として 4 つに分割し，最もレーティングが高いソースコード群を上級者，最もレーティングが低いソースコード群を初級者，残りの 2 つを中級者コード群とする．

次に分割したソースコードに対して表 1 に示した予約語の利用頻度やメトリクスの値などのソースコード特徴量（表 2）を取得し，機械学習の説明変数とする．目的語はソースコードの良さであり，上級者，中級者，初級者それぞれのコード群をコード群を学習する．

最後に，学習したモデルにソースコードを入力することで，ソースコードの品質を評価することができる．

表 1 特徴量に使用した予約語

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			

2.5.2 code2vec

code2vec[6] はソースコードのメソッドに対する分散表現を作るニューラルモデルであり，Uri らは Allamanis ら [4] が作成した Java のデータセットを基に作成した訓練済みモデル[*3]や，モデルを利用するためのソースコード[*4]をオンライン上に公開している．

[*2] https://sites.google.com/site/miningprogcodeforces/
[*3] https://code2vec.s3.amazonaws.com/model/java-large-released-model.tar.gz
[*4] https://github.com/tech-srl/code2vec

表 2 特徴量に使用したメトリクス

メトリクス	説明
avg complexity	各関数の CC の平均値
max complexity	各関数の CC 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

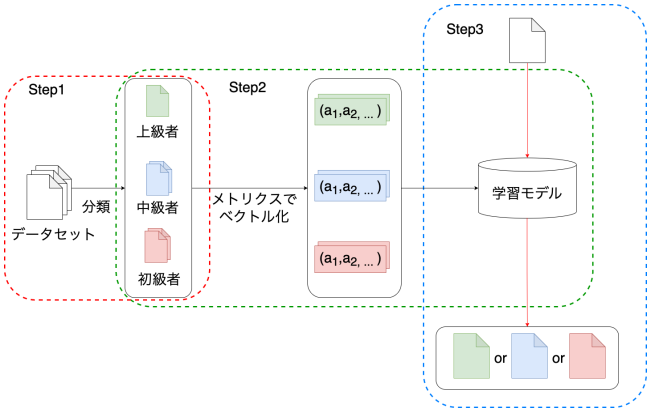

図 2 品質の自動評価手法

このモデルの学習までの手順を説明する．まず図 3 のようにソースコードのメソッドを抽象構文木（以下，AST）に変換する．AST はソースコードをコンパイル際にできる構文情報を表現した中間表現であるため，AST を利用することでソースコードの構造に着目した分散表現を作成する．

次に AST から Bag of Path-Contexts という分散表現を作成する過程を図 4 に示す．AST の葉から葉までのパスを表現したものを AST-Path とし，（始点, AST パス, 終点）を Path-Context とする．Path-Context は一つの AST に対して複数存在するため，それらをまとめて Bag of Path-Contexts という．

最後に Bag of Path-Contexts をベクトルに変換する．まず Path-Context の始点，終点，AST パスを図 5 のように

```
public class Main {
  public static void main(String[] args) {
    if(args.length > 0) System.out.print(args[0]);
  }
}
```

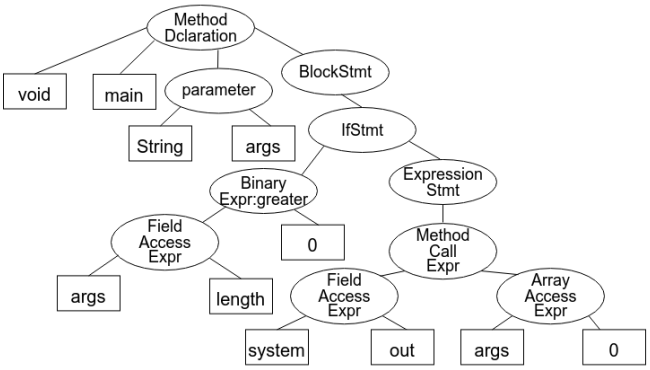

図 3 AST の例

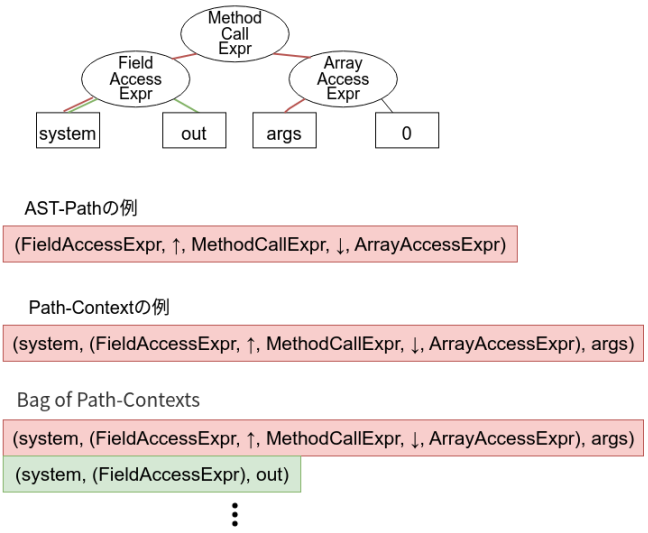

図 4 Bag of Path-Contexts までの流れ

One-Hot ベクトルに変換した後，埋め込みを行うことで次元を固定長に変換する．さらに作成した Path-Context の埋め込みベクトルを活性化関数 tanh と線形変換する行列を用いて一つのベクトルに結合する．

モデルは Bag of Path-Contexts を基に学習するため，ソースコードの構造に着目したベクトルとなる．著者の Alon らは code2vec を用いて以下のようなことができると述べている．

- メソッド名の提案
- 機能の類似性を基にしたコード検索
- プログラムが I/O を実行するか予測
- プログラムの依存関係を予測
- マルウェアの検知

研究ではソースコードをベクトル化し，類似したコードの探索を行うことに利用している．

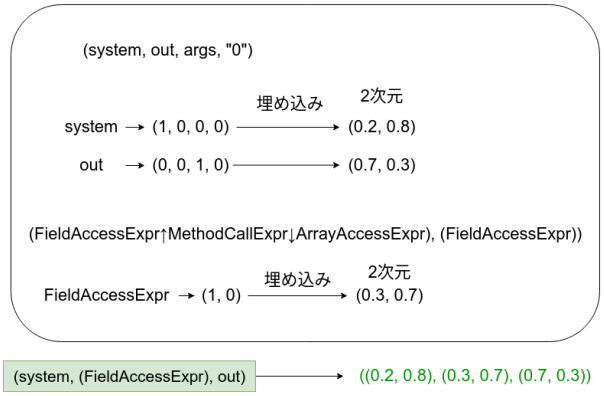

図 5 Path-Context のベクトル化

2.5.3 code2seq

code2seq[5] は code2vec を改良し，メソッド名の提案に特化させるために設計されたニューラルモデルであり，Uri らは code2vec のモデルを作成したときと同様のデータセットを用いて作成したモデル[*5]と，モデルを利用するためのソースコード[*6]をオンライン上に公開している．code2seq でも 2.5.2 項と同様に AST から AST-path を作成するが，code2vec とは異なり code2seq では学習の精度を上げるために葉をサブトークンに分割する．

具体例としては $linear\_search$ を $linear, search$ のように変化させることで固有の言葉を減少させる．また AST-Path をベクトル化する際に bi-directional LSTM[15] という入力の時系列に依存した学習に有効な学習方法を利用している．これはメソッド名を単語の羅列として出力できるように前後の関係を学習させるためである．さらに Attention 機構 [10] を用いることにより，入力のどの単語が重要かを重み付けしている．これらの工夫により code2seq はメソッド名を連続した単語の羅列で提案できる．この手法を利用して研究ではソースコードの要約を行った．

このようにして作成したモデルも 2.5.2 項で説明した code2vec と同様に bag of Path-Contexts を基にベクトル化しており，ソースコードの構造に依存した要約となる．

3. システムの設計

本節では，提案するシステムの設計について説明する．システムの全体像を図 6 に示す．

このシステムでは，入力するソースコードと，改善の基となる模範解答コード群が事前に準備されていることを前提とする．まずソースコードが入力されると，そのソースコードに改善の余地があるかを判定する．これは改善の余地が特段必要ない場合に終了させることを目的としており，この部分には 2.5.1 節で述べた既存研究を用いる．仮

[*5] https://s3.amazonaws.com/code2seq/model/java-large/java-large-model.tar.gz
[*6] https://github.com/tech-srl/code2seq

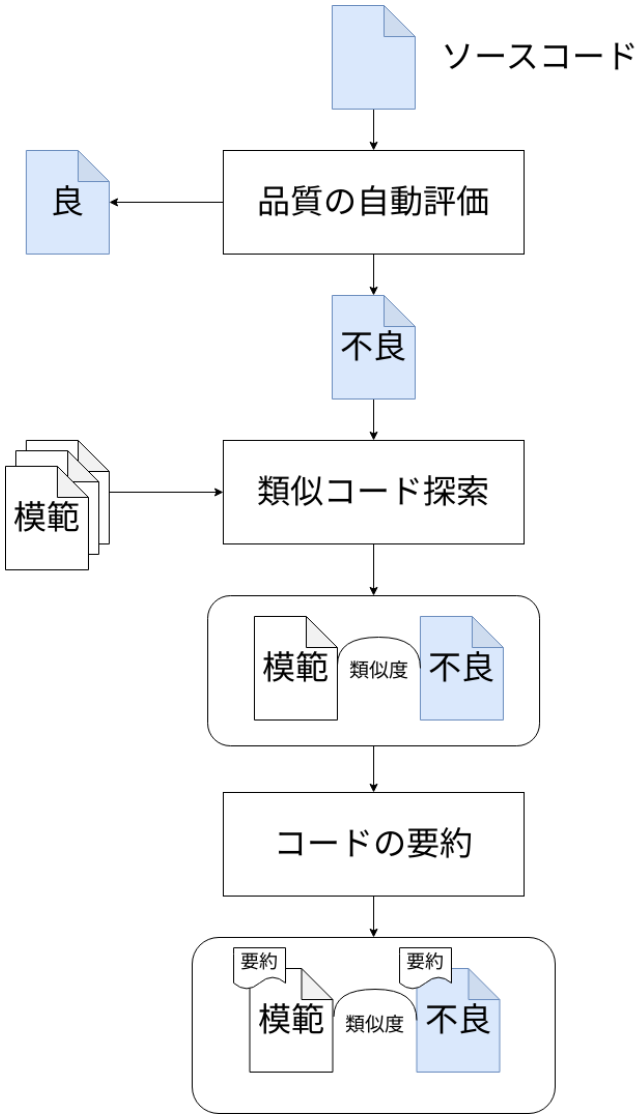

図 6 プログラミング学習システム

に良いソースコードであれば改善の余地がないとして終了し，それ以外の判定が出力された場合は改善の余地があるとして改善に適切な模範解答コードを探索する．

次に，入力したソースコードの情報を基に適切な模範解答コードを決定するため，入力と模範解答コード群に含まれるソースコードをそれぞれ分散表現にした上で，その類似度を計算する．これはソースコードを分散表現にすることで適切な模範解答コードを計算により求めることを目的としており，分散表現を作成する手法として 2.5.2 節で述べた既存研究を用いる．また類似度としてコサイン類似度を用いた．

類似コード検索の結果，与えられた入力コードに類似している模範解答コードに含まれるソースコードを得た後，入力コードと模範解答コードとの差分をわかりやすく示すために，ソースコードの要約を行う．この要約は，単に模範解答コード群に含まれているソースコードそれ自体を提示するだけでなく，入力コードならびにそのソースコードを要約することで，ソースコードをどのように変更するのか，といった模範解答コードに対する理解を促すことを目的としており，要約する手法は 2.5.2 節で述べた既存研究を用いる．

最後に，上記の結果として，以下 4 種類の情報を出力する．

- 入力したソースコードの要約情報
- 模範解答コードのファイル名
- 模範解答コードの要約情報
- 入力コードと模範解答コードの類似度

4. システムの実装

本節では，3 節で述べた内容の実装方法について説明する．

4.1 模範解答コード群と付随する情報の作成手順

模範解答コードは入力と独立しているため，あらかじめソースコードの構造に基づいたベクトル化やコードスニペットの予測によって要約することができる．模範解答コード群は 2.3 節で述べた Project CodeNet のデータセットを用いて作成した．ここでは模範解答コードとする条件とその要約方法，ベクトル化の実装について述べる．

4.1.1 模範解答コード群

Project CodeNet に含まれている解答コードとメタデータを基に，模範解答コード群を作成する．ここで模範解答コードは，Project CodeNet に含まれていた解答コードのうち，問題に正解しており，ソースコードのメトリクスや実行時間が表 3 をすべて満たすものとした．この指標を用いた理由は，LOC と CC が低いものを指標にすることで，模範解答コードのソフトウェア品質を高く保ち，実行時間が短いソースコードを選択することで，模範解答コードをより良いソースコードとするためである．

表 3 模範解答コードの条件

メトリクス	閾値
LOC	全体の 50%以下
CC	全体の 50%以下
実行時間	全体の 50%以下

4.1.2 模範解答コードの要約とベクトル化

模範解答コードのベクトル化と要約の様子を図 7 に示す．2.5.2 節で述べた，オンラインで公開されている code2vec の訓練済みモデルを用いて模範解答コードを分散表現に変換し，得られたベクトルをファイルに出力した．ここで用いるベクトルの次元は 100 次元であり，main 関数の分散表現をソースコードの分散表現としている．

同様に 2.5.3 節で述べた，オンラインで公開されている code2seq の訓練済みモデルを用いて模範解答コードを要約

し，模範解答コードの要約情報をまとめるファイルに出力した．このとき，ソースコードの要約として main 関数を要約したものを利用する．

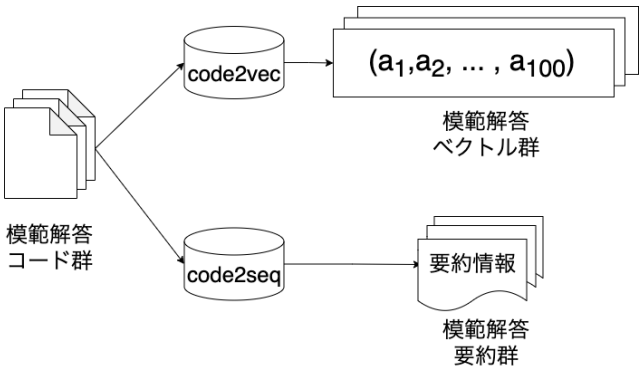

図 7 ソースコードのベクトル化と要約

4.2 入力コードの判定

入力コードの判定は 2.5.1 節で述べた既存手法を用いた．具体的には，Project CodeNet の解答コードを用いてモデルを作成した上で，ソースコードの評価を図 2 と同様に行った．その結果，仮に入力したソースコードが’良’と判定された場合はシステムを終了させ，それ以外の場合は改善が必要であると判断して，以降の手続きを行う．

4.3 入力コードの要約とベクトル化

改善が必要と判断された入力コードを対象として，要約やベクトル化を行う．この時，code2vec, code2seq のモデルはそれぞれ 4.1.2 節と同様のものを用いて，入力したソースコードのベクトル化と要約情報を得る．このときのベクトルも 100 次元であり，要約情報はソースコードの main 関数を要約したものである．

4.4 入力コードと類似する模範解答コードの選択

入力したソースコードのベクトルと各模範解答コードとのコサイン類似度を計算し，類似度の高いもの 3 つを模範解答コードとして出力する．このときコサイン類似度はベクトルの大きさ $|V|$ を用いて以下のように定義した．この類似度は 1 から-1 までの値を取り，ベクトル間の角度に依存するという特徴があり，機械学習の分野でよく用いられている．

$$cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}|\cdot|\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\cdot\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

4.5 出力

実際の入力と出力結果を図 8，図 9 にそれぞれ示す．図 9 より入力された Input.java の要約が main, 最も類似したソースコードが s000127.java であり．類似度が 0.16 であることなどがわかる．

Input.java

```
public class Main {
  public static void main(String[] args) {
    if(args.length > 0) System.out.print(args[0]);
  }
}
...
```

図 8 入力コード

```
Input.java
predict: main

s000127.java
predict: main
similarity: 0.16

s000182.java
predict: echo
similarity: 0.12

s000102.java
predict: main, echo
similarity: 0.08
```

図 9 システムの出力

5. 評価実験

本節では，提案システムの評価について説明する．システムを被験者に利用してもらう実験を行い，定量的評価とアンケート評価の 2 種類で評価を行った．

5.1 概要

実験は図 10 のように行った．まず最初に，Project CodeNet に含まれている表 4 で示した 2 種類の問題を用意し，被験者に実験の概要と注意事項を説明する．被験者は表 4 の問題に対して Java を用いて各問題について 1 つづつ，合計 2 つの解答を作成する．その後，片方の解答は正解とされた解答群を基に自身で模範解答を決定して改善し，もう片方の解答はシステムによって提示された模範解答コードを基に改善する．その後，システムに関するアンケートへ回答する．アンケート調査には Google が運営している Google Forms[*7]を用いた．

5.1.1 定量的評価

システム利用の有無により，以下の指標がどのように変化したか確認することでシステムを評価する．

- CC
- LOC

[*7] https://docs.google.com/forms/

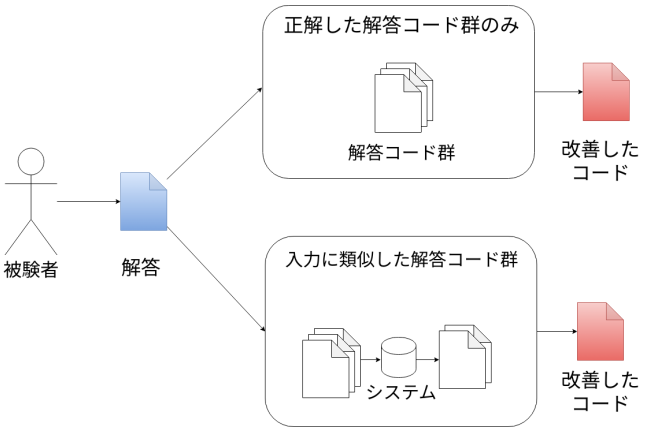

図 10 実験の流れ

表 4 利用した問題

問題番号	問題名	問題内容
1	玉	2 つの箱があり，1 10 までの数字がランダムに書かれた玉を先頭から順にすべて 2 つの箱のどちらかに入れたとき，数字が箱の中で降順となる場合は存在するか.
2	平面上の図形	8 × 8 マスにある数字列を読み込んで，置かれている図形の種類を出力しろ.

- 改善時間

5.1.2 アンケート評価

アンケートによって，被験者の妥当性とシステムの利便性について確認する．被験者は以下の内容に対する理解度を，表 5 を基準に 5 段階で回答する．

- Java
- プログラミングコンテスト
- Visual Studio Code

これは被験者に関する妥当性を確認するものであり，Java は解答する言語が Java のため，プログラミングコンテストは問題がプログラミングコンテストのため，Visual Studio Code は解答する環境が Visual Studio Code のためである．この理解度を 1 と回答した被験者はシステム以外の要因で結果が変わることを考慮し，今回の実験の対象外とする．また，問題それ自身と手動で改善を行った場合，ならびに本システムを用いて改善を行った場合の 3 点について，表 6 に示す 5 段階の難易度の指標を回答する．これにより問題の妥当性とシステムの利便性について確認する．

5.2 結果と考察

この節では被験者 2 名によって行った実験の結果と．結果に対する考察に関して説明する．

被験者が解答したソースコードを，システムを用いずに正解した解答群のみで改善した場合とシステムを用いて改善した場合，それぞれに対して，定量的評価とアンケート

表 5 理解度の指標

level	内容
1	理解していない
2	あまり理解していない
3	どちらともいえない
4	理解している
5	普段から使用している

表 6 難易度の指標

level	内容
1	簡単だった
2	やや簡単だった
3	普通
4	やや難しかった
5	難しかった

調査を行った結果を表 7 に示す.

表 7 実験結果

問題番号	1		2	
システム	なし	あり	なし	あり
CC	6	4.5	7	16
LOC	34	36	60	53
改善時間 (min)	13	10	14	13
改善の難易度	1	2	3	2

表 7 から問題番号 1 ではシステムを用いた方の CC が低くなっているが，問題番号 2 ではシステムを用いた場合 CC が高くなっていることがわかる．これはシステムが類似度による模範解答コードを出力するため，入力されたソースコードの構造が影響されたと考えられる．

次に，改善時間の変化からシステムを用いてることで改善時間の短縮ができていることがわかる．これは模範解答を提示することで被験者が確認したソースコードの量が少なくなったためであると考えられる．

最後に，アンケート結果ではシステムを利用することで改善の難易度が下がったと感じる場合と，改善の難易度が上がったと感じる場合があることが分かった．これはシステムの評価基準がソースコードの構造のみに着目しているため，コーディングルールのようなその他の要因によるものであると考えられる．またこの結果は CC の改善とトレードオフになっていると考えられ，CC を小さくすることで改善の難易度が上がると予想される．

今回の内容から，ソースコードの構造に関する類似度を用いて模範解答を推薦すると，より品質の高い構造に気づかず，根本的な修正ができない場合がありえる．そのため，類似度の高さ以外にも新たな指標を用いて模範解答を推薦し，その際 CC を下げつつ改善の難易度が上がらないような工夫を行った方が良いと考えられる．

6. まとめ

本研究では，増加するプログラミング初学者の学習を支援するために，入力したソースコードを基に，入力と構造が類似している模範解答コードを提示するプログラミング学習支援システムを構築した．このシステムの主な機能

はソースコードの自動評価，ソースコードの要約，ソースコードのベクトル化と類似コードの選定である．

提案したシステムを Project CodeNet のデータセットを利用して定量的評価とアンケート評価を行った．その結果，システムを用いたほうが改善時間を短縮できることがわかった．またアンケートの結果から，システムを用いて改善する方が改善自体を難しく感じる場合があり，ソースコードの類似度が高いだけでは必ずしも理解しやすいとは限らないことがわかった．この結果から，プログラミング学習者にとって理解しやすい模範解答を推薦するために，プログラムの構造以外の指標も含めたシステムの改良を行いたい．

参考文献

[1] Aizu online judge: Programming challenge. https://judge.u-aizu.ac.jp/onlinejudge/. (Accessed on 02/07/2023).
[2] Atcoder. https://atcoder.jp. (Accessed on 02/07/2023).
[3] Topcoder. https://www.topcoder.com/. (Accessed on 02/07/2023).
[4] Miltiadis Allamanis, Hao Peng, and Charles Sutton. A convolutional attention network for extreme summarization of source code. In *International Conference on Machine Learning (ICML)*, 2016.
[5] Uri Alon, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. 2019.
[6] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. Code2vec: Learning distributed representations of code. *Proc. ACM Program. Lang.*, Vol. 3, No. POPL, pp. 40:1–40:29, January 2019.
[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. pp. 248–255, 2009.
[8] 福家範浩, 藤原裕士, 吉田則裕, 崔恩瀞, 井上克郎. 深層学習を用いたコードクローン検出器の汎化性能に関する調査. 情報処理学会研究報告, Vol. 2021-SE-207, No. 12, pp. 1 – 7, 2021.
[9] David Kolb. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice Hall, 1984.
[10] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. Vol. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1412–1421, 09 2015.
[11] 松井智寛, 松下誠, 井上克郎. ソースコードのグラフ表現を利用した深層学習によるコーディングの専門性の判定手法. 情報処理学会研究報告, Vol. 2022-SE-210, No. 12, pp. 1 – 8, 2022.
[12] 文部科学省. 教育の情報化の手引き-追補版-(令和 2 年 6 月). 2020. https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/mext_00117.html. (Accessed on 02/07/2023).
[13] 文部科学省. 小学校プログラミング教育の手引 (第 3 版). 2020. https://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1403162.htm. (Accessed on 02/07/2023).
[14] Ruchir et al. Puri. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. 2021.
[15] Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, Vol. 45, No. 11, pp. 2673–2681, 1997.
[16] Martin Shepperd. A critique of cyclomatic complexity as a software metric. *Software Engineering Journal*, Vol. 3, pp. 30 – 36, 04 1988.
[17] 堤祥吾, 吉田則裕, 崔恩瀞, 井上克郎. プログラミングコンテスト参加者を対象とした編集作業の特徴調査. 情報処理学会研究報告, Vol. 2017-SE-197, No. 6, pp. 1 – 8, 2017.