Vol.2021-HCI-193 No.3
Vol.2021-EC-60 No.3
2021/6/1

プログラム理解を支援するための可視化手法の自動選択

伊藤 誓[1,a)]　福里 司[1,b)]　五十嵐 健夫[1,c)]

概要：プログラミング・アルゴリズム学習支援を目的とした可視化システムが，これまでに多数提案されている．しかし，これらのツールではいずれも，ソースコード提示されるオブジェクトのマッピングをユーザが定義しなければならない点や，適用可能なプログラムの種類が限られてしまう課題が存在し，手軽かつ柔軟な可視化を実行することは難しい．そこで本研究では，これらの問題を解決するためのデータ駆動型システムを提案する．このシステムでは，ソースコードの内容を説明するための可視化方法（ラベル）を深層学習モデルを用いて決定する問題に帰着し，任意のソースコードに対する可視化方法の自動選択とハイライト表示を実現する．更に，ユーザテストの結果を通して，本研究がプログラムの理解支援につながる可能性を示す．

1. はじめに

コンピュータシステムの高度化やビックデータ解析の重要性の拡大に伴い，プログラミング技術が注目を集めている．プログラミング技術において，与えられたソースコードとその実行結果からプログラムの一連の動作を把握する力が求められるものの，プログラミング初学者にとっては非常に難しい．このような背景の下，ソースコード中の変数の値や配列の値の移り変わりを可視化するシステム [1], [2], [3], [4], [5] や，ソースコード内のアルゴリズム（例：ソートアルゴリズムや経路探索アルゴリズム）の特徴を把握するための可視化システム [6], [7], [8], [9], [10], [11], [12], [13], [14] が考案されている．しかしこれらの技術では，ユーザが可視化方法の種類（例：ヒストグラムやグラフ構造，配列）を設定しなければならないことが問題として挙げられる．更に，これらは教育目的に用いられるアルゴリズムに限定されている場合が多く，任意のソースコードに適用することが難しい．このような背景から，プログラムやアルゴリズムを可視化システムが実用化されているケースが非常に少ない．

そこで本研究では，任意のプログラムに適用可能なプログラムの可視化ツールを実現するための第一歩として，データ駆動型アプローチを検討する，具体的には，ソースコードを要約するための機械学習モデルを用いて，入力ソースコードの「可視化方法の選択」及び「メインの動作を表す位置の特定」を行う手法を提案する．本システムでは，オンラインのプログラミング練習サイト [15], [16] で公開されているプログラミング課題に提出されたソースコードをデータベースとして用いている．更に我々は，本システムの有用性を評価するために二種類のユーザテストを実施した．一つ目は，可視化方法に対する被験者の印象を調査するものである．二つ目は，可視化の有無を比較することで，可視化機能がプログラミング学習支援に有用であるかを評価するものである．

[1] 東京大学
The University of Tokyo
[a)] chikai.ito@is.s.u-tokyo.ac.jp
[b)] tsukasafukusato@is.s.u-tokyo.ac.jp
[c)] takeo@acm.org

2. 関連研究

本章では，我々の提案するシステムに影響を与えた先行研究を時系列順に紹介する．

2.1 プログラミング可視化ツール

SeeC [2] は，Heinsen Egan と McDonald によって，2013年に提案されたプログラミング可視化ツールである．このシステムは，初心者向けの C プログラムのデバッグシステムに基づき，メモリ状態のグラフィカルな可視化表示機能や自然言語による説明文の作成機能，エラー検出や実行トレース機能を有している．Omnicode [3] は、Python Tutor に基づく python 対応のライブプログラミング環境である．このシステムでは，プログラムを継続的に実行することで，実行ステップごとにソースコード中の変数の値の移り変わりを可視化することができる．更に，ユーザは表示する情報を制限するフィルタ処理や，より詳細な情報を得るためのズームを行うことができる．その一方で，Omnicode は複数の可視化方法には対応していない点や，プログラム

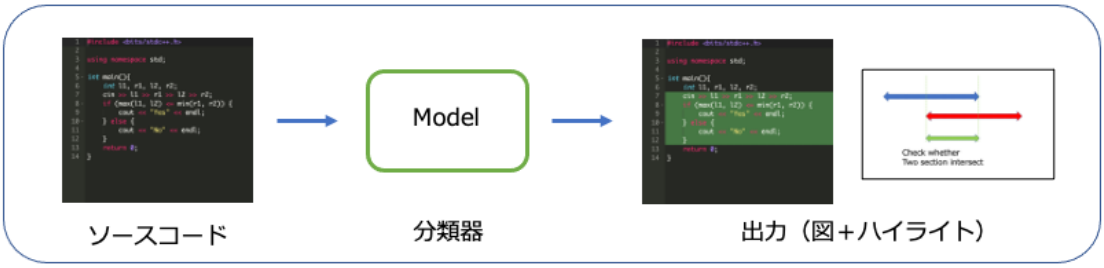

図 1 本システムの概要.

内の全ての変数を可視化することで，挙動を把握しづらくなってしまう点が課題として挙げられる．OPT+Graph [1] は C 言語向けのグラフ構造体の可視化ツールである．このシステムでは，グラフの可視化方法（隣接行列）に加え，ノードとエッジを利用した可視化を行うものである．グラフオブジェクトは、ソースコードが事前に定義された記述方法（例：配列）を含んでいるかどうかを検出している．Willow[4] は，プログラム可視化ツールとアルゴリズム可視化ツールの中間として紹介されている．このシステムは，ステップ実行などに代表されるプログラミング可視化ツールと同様のシステムを搭載している．更に，プログラム内の各オブジェクトに対する可視化方法（例：ヒストグラムやグラフ）の設定や，リストや木構造などのデータ構造の検出も可能である．

2.2 アルゴリズム可視化ツール

TANGO [12] は，アルゴリズムの実行結果に対するアニメーションを生成するシステムである．ユーザは任意のプログラムにおける中層的な操作やイベントを設定やそれらの関係性を定義することができる．TANGO で使用されている “path-transition model” は，抽象的なデータ型を基に基づいてアニメーションの簡単な生成が可能になっている．Javenga [9] は，経路探索アルゴリズムやネットワークフロー等のグラフアルゴリズムの可視化機能を有する WEB ベースのソフトウェアである．このツールでは，テキストエディタの代わりにグラフエディタを用いることで，有向（or 無向）のグラフを作成することができる．また，入力グラフに対して事前に用意されているアルゴリズムを選択することで，アルゴリズムの実行結果をステップごと，またはアニメーション形式に可視化することができる．DAVE [14] は，二分探索をはじめ配列を扱うアルゴリズムを対象とした WEB ベースの可視化環境である．ユーザは表示されたアルゴリズムを一つ選択し，入力データの追加及び疑似コードの修正作業によって，配列内の要素がどのように変化するのかを可視化できる．このツールは、実行中のコード行や重要なイベントをハイライト表示するため、初学者のアルゴリズム理解に有用である．Lytle ら [17] は，効率的な経路探索アルゴリズムを開発するためのシステムを考案した．このシステムの目的は，アルゴリズム理解を支援するのではなく，アルゴリズム自体の作成を支援することである．また，実装したアルゴリズムを基に，周囲の環境に合わせてロボットが動く様子を API を通じて確認することができる．この研究の評価実験は，ツールを使って作成したアルゴリズムの評価によって行われた．

表 1 プログラムの基本動作とデータセット内の出現数.

ID	動作内容	出現数
1	入力値の桁和を計算する	5
2	二つの配列について処理を行う	4
3	配列内の要素をソーティングした後，配列の数値演算をする	4
4	配列内の要素のうち，どの値をいくつ持つのかを記録する	9
5	配列内の要素を用いて演算処理を行う	16
6	配列内の要素のうち，特定の条件を満たす要素数を数える	21
7	累乗や組み合わせ・階乗などを計算する	11
8	条件を満たす添字のペアを探す	8
9	グラフ問題を解く	10
10	二つの数値区間が交差するかどうかを判定する	3
11	入力値の約数を列挙する	5
12	部分集合を列挙し，全探索する	6
13	入力値の桁数を取得する	4

3. 提案システム

本システムの概要を図 1 に示す．我々のシステムでは，ユーザが実装した C++のソースコードを入力とし，(1) メインの動作に適した可視化方法の自動選択と，(2) ソースコード中の最も関連する領域のハイライト提示を行う．

3.1 データセット

我々は，二つの競技プログラミングの練習サイト（AtCoder [15] と Aizu Online Judge (AOJ) [16]）に着目し，機械学習モデル用のデータセットを構築する．競技プログラミングとは，問題に対して正しい解を出力するプログラムを作成するマインドスポーツの一種である．プログラムの正当性については，いくつかのテストケースに対して正しい出力が得られるかどうかで判定される．

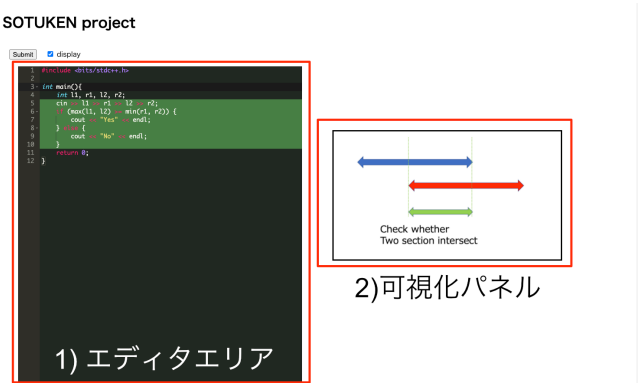

図 2 提案システムのスクリーンショット.

本研究では，頻出問題である十三種類のプログラムの基本動作に着目し，「ソースコード」と「基本動作に対する可視化方法 (ラベル)」のペアデータを作成することを検討する（動作の種類とデータセット中の出現数を表 1 に示す）．具体的な手順として，はじめに十三種類の動作に分類可能な問題，計 97 問 (AtCoder から 82 問，AOJ から 15 問) に対してラベルを手作業で付与する．但し，複数の動作を含むプログラムの場合，複数のラベルを付与するものとする．次に，各問題に対して提出されたソースコード計 46,305 個（一問当たり最大 500 個分）を収集する．但し，ソースコードの長さが極端に短いものや長いものは除外し，学習用とテスト用にランダムに分割した（今回は 68 問を学習用データ，29 問を評価実験用に用いる）．

3.2 ユーザインターフェース

本システムは，1) C++のソースコードを記述するためのテキストエディタと，2) 可視化パネルの二つで構築されている（図 2）．テキストエディタは，行番号と自動インデントをサポートする Javascript のライブラリ Ace を用いた．ユーザが “submit” ボタンを押すことで，テキストエディタに書いたソースコードに対応する可視化方法を図形式で表示することができる．更に，図を表示するタイミングで，図に対応するソースコードの一部をハイライト表示する（緑色）．

3.2.1 機械学習モデル

我々は，ソースコードに対する既存の機械学習モデルのうち，分散表現を得ることに特化した Code2Vec [18] を基に可視化方法のラベル ID を推定する学習器を構築する．

具体的な方法としては，第 3.1 節で用意したソースコードに astminer [19] を適用することで，Code2Vec モデル用のデータ形式を生成する．但し，変換後のデータは固有 ID が付与されてしまい，そのまま学習に用いることが難しい．そこで，我々は固有 ID を元の文字列情報に復元する後処理を加えた．Code2Vec の学習に用いたパラメータ値は表 2 に示す．

表 2 Code2Vec モデルに用いたパラメータ値.

パラメータ	使用した値	初期値 [18]
MAX_CONTEXTS	160	200
MAX_TOKEN_VOCAB_SIZE	8000000	1301136
MAX_TARGET_VOCAB_SIZE	1000	261245
MAX_PATH_VOCAB_SIZE	300000	911417
NUM_TRAIN_EPOCHS	15	20

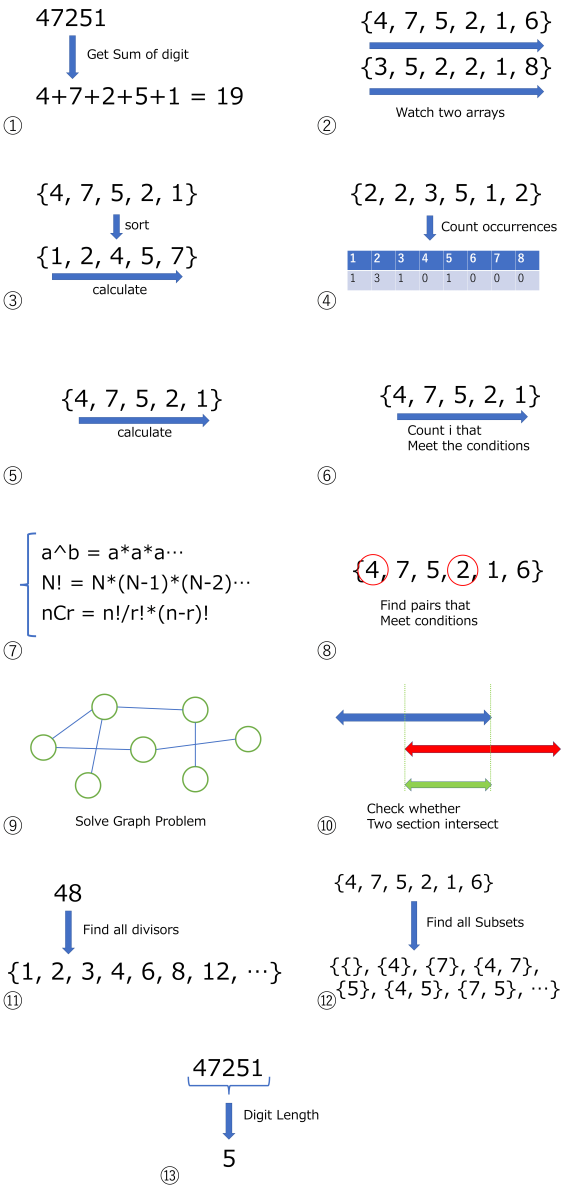

図 3 各ラベルに対するテンプレート図．プログラムの基本動作に対応するイラストと自然言語による説明で構成されている．

しかしながら，本研究の学習モデルによるラベル推定精度は約 45%程度となり，数値的には不十分な結果となった．将来的にはデータセットの拡張等が求められる．

3.2.2 可視化手法

本研究では，推定されたラベル ID を基に（事前に用意した）十三種類のテンプレート図（図 3) の中から一つを選択し，画面に直接表示する方法を用いた．但し，ソース

表 3　テストデータ内のソースコード数.

ID	選別前 (58 個)	選別後 (31 個)
1	4	4
2	2	0
3	2	1
4	1	0
5	9	4
6	10	7
7	6	5
8	8	3
9	6	3
10	2	0
11	4	2
12	3	1
13	3	1

コード内の変数とテンプレート図が一致しない可能性があるため，将来的には既存の可視化ツールとの組み合わせることを予定している．

ハイライト機能については，機械学習モデルによって推定された Attention Weight が最大となるパスを取得する．得られたパスを基にソースコードの開始行と終了行を検出し，ハイライト箇所を決定する．

4. ユーザテスト

提案システムの有用性を確認するために，ユーザテストを実施した．但し，本研究の目的である「ユーザのソースコード読解の助けになるか」については，事前に用意したラベル（テンプレート図）や機械学習モデルの推定精度に大きく依存する．そこで我々は，推定されたラベルが正解データと一致するソースコードを用いて，1) 可視化方法に関する評価実験と，2) 可視化の有無に関する評価実験を行った．

4.1 可視化方法における評価

出力結果（図とハイライト）が，「ソースコードの内容を表すのに適切か」を評価するために，1 年以上のソースコード読解の経験を有する学部生三名を対象としたユーザテストを実施した．但し，C++のソースコード読解の経験については「1 ヶ月から 3 ヶ月程度」が一名，「1 年以上」が二名である．

4.1.1 実験環境

このユーザテストは，全ての工程をオンライン上実施した．具体的には，被験者に Google Form を用いて作成したアンケートを送付し，自由なタイミングで回答してもらうものである．但し，ソースコードと出力結果（図とハイライト位置）を同時に見れるように，今回はプログラムの出力結果をスクリーンショットで記録し，Google Form に画像として貼り付けている．

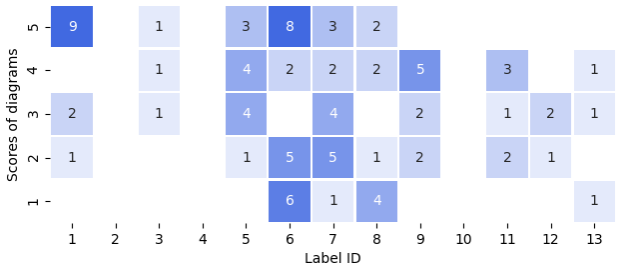

図 4　可視化方法に対する各ラベルのスコア分布.

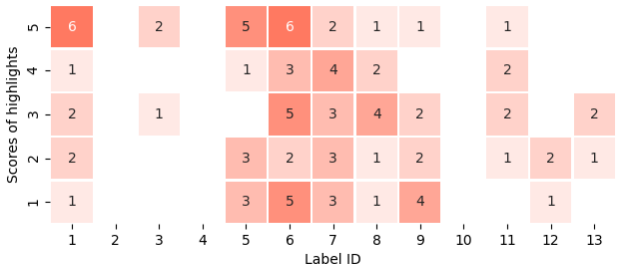

図 5　ハイライト表示に対する各ラベルのスコア分布.

4.1.2 実験手順

第 3.1 節にて用意したテストデータ（29 問）に対するソースコード 58 個（各問題につき 2 つ）のうち，ソースコードの長さが 50 行以下かつ，推定された可視化方法のラベルが正解データと一致する 31 個を用いた（ラベルごとに分類した結果を表 3 に示す）．次に，被験者は各ソースコードに対する可視化方法の結果とハイライト箇所を確認し，「図とハイライトが，プログラムの基本動作を表しているか」を五段階リッカート尺度（1:妥当ではない，5:妥当である）によって回答した．更に，実験全体を対する感想を記述してもらった．

4.1.3 実験結果

可視化方法（ラベル）ごとにスコアを集計し，ヒートマップ形式に表したものを図 4，5 に示す．この結果から，可視化方法（表示される図）が 3 点以上となった一方で，ハイライト位置については全体的に低い結果（2 点以下）となったことが分かる．また，ラベルの種類によって，被験者の回答に差があることが確認できる．

被験者の意見として，表示された図については「図がソースコードの一部にのみ対応し，コードコード全体と対応していない」「図に余計な情報が含まれている」といった意見が挙げられた．その一方，ハイライトについては，一名の被験者から「ハイライトのルールが一貫していないため，違和感を感じる」といった回答があった．例としては，for-loop ブロックをハイライトする際に，外側の変数宣言をハイライトに含める場合と含めない場合が挙げられる．

4.2 可視化の有無における有用性評価

二つ目の実験として,「可視化自体がソースコード読解に役に立つのか」を評価するために，提案システムによる可視化結果（図やハイライトの表示）がある場合とない場合の比較実験を行った．今回は，インターネット上で募集した五名のプログラマを被験者として集めた．なお被験者のソースコード読解の経験は全員「1 年以上」，C++のソースコード読解については「1 ヶ月以下」が二名,「1 ヶ月から 3 ヶ月程度」が一名,「1 年以上」が二名であった．

4.2.1 実験環境

実験 1 と同様，Google Form を用いて作成したアンケートを被験者に送付し，自由なタイミングで回答してもらった．但し，本実験の目的は「提案システムの出力の有無」によるソースコード読解の理解度の違いの検証であるため，出力結果が ON の状態と OFF の状態の二種類のスクリーンショットを画像として貼り付けている．

4.2.2 実験内容

可視化自体の精度による影響を軽減するために，実験 1 にて高いスコアを得たソースコード 20 個（内容を表す適切な可視化結果が得られるソースコード）を用いた．ラベルごとのソースコード数の分布を表 4 に示す．それぞれの理解度を評価するために，我々は被験者に要約課題を与えた．具体的な手順としては，まず被験者を二つのグループに分ける．次に，一つ目のグループには 10 個のソースコードと可視化結果を同時に見た上で要約を行ってもらい，残り 10 個のソースコードは，可視化結果なしで要約を行ってもらう．二つ目のグループには，一つ目のグループとは逆の順序のタスク（前半 10 問はソースコードのみ，後半 10 問は可視化有）を設けた．但し，実験開始前にそれぞれの回答に時間制限（3 分以内）と文字制限（10 文字〜20 文字以内）を設定し，要約の例を複数紹介することで，被験者の負担を軽減した．また，被験者の C++に関する基礎知識の差による影響を軽減させるために，インターネットによる検索を行うことや，わからない問題をスキップすることを明示的に許可した．

更に，上記の 20 個のソースコードに対する要約が終わった後，被験者には以下の三つの設問（自由記述式）に回答してもらった．

(1) 可視化機能によって，コード読解に要する時間が短くなったと思いますか？
(2) 可視化機能によって，コードをより正確に読解できるようになったと思いますか？
(3) 可視化の有無を比較した際，他に感じたことや思ったこと（例：提示される情報や，タスクの内容について良いと思ったことや不満だったことなども含む）があれば記述してください．

表 4 実験 2 で用いたソースコードのラベルごとの分布.

ID	選別前 (31 個)	選別後 (20 個)
1	4	4
2	0	0
3	1	1
4	0	0
5	4	2
6	7	3
7	5	4
8	3	1
9	3	1
10	0	0
11	2	1
12	1	0
13	1	1

4.2.3 実験結果

三つの質問に対する被験者の回答を紹介する．一つ目の質問については，五名の参加者のうち「要約に要する時間が減ったと感じる」と回答したのは二名,「少し減った」と回答したのは一名,「変わらなかった」と回答したのは二名となった．肯定的意見としては,「ハイライトがあることで，ソースコードの中のどこを読むべきかすぐ判断できた」等があった，その一方で,「ソースコードの内容を可視化結果（図やハイライト）が正しく表現していない場合があり，その場合は読む時間が長くなる」という否定的意見も得られた．また一部の参加者からは「表示された図とプログラムの関係性が理解得できなかった」「前半の実験での慣れによって，後半の実験の時間が短くなった気がする」といった意見も得られた．二つ目の質問については，読解の正確性について肯定的な意見を持つ被験者は一名のみであった（否定的な意見は三名，中立な意見が一名）．主な意見としては,「ソースコード自体を読めば，内容を把握できるため正確性には影響しない」「ソースコードに対する図の表示が一致していない場合 (or 理解しにくい場合)，要約との整合性を取るべきか分からず悩んだ」などが挙げられる．三つ目の質問の回答として，以下のような肯定的な意見（一部）を載せる．

- 可視化機能があることで，ソースコードの詳細部の確認が不要になると思う．
- COVID-19 の状況下，画面共有によるコードレビュー等を行うシナリオに現実味を感じる．

また，実験方法の改善点に関する指摘も見られた．

- 提示されるソースコード自体が，（本来の内容に大して）読みづらいと感じた．
- 提示されるソースコードが，似ている機能や関数を有するものが時々登場していた．

5. 議論

5.1 可視化方法について

可視化方法について，被験者からは「出力された図が，ソースコードと部分的にしか一致していない」という意見が多く得られた．これは，各ソースコードに事前に用意した十三種類のラベルの中から一つを割り振る方法を用いていたため，ソースコードとラベルが部分的に一致していないことが原因として考えられる．また，表示される図の種類（ラベル）によって被験者の回答に差があることも確認できた．これはデータセット内の各ラベルのソースコード数に偏りがあることや，事前に用意したテンプレート図自体の品質に影響していることが考えられる．今後の課題として，「図の種類を増やす」「テンプレート図の再設計」といったデータセットの改善や，図以外の表示方法（例：自然言語による文章の生成）を組み合わせることが挙げられる．

5.2 ハイライト箇所について

ハイライト位置について，被験者から「ハイライトされる箇所の一貫性がないため，違和感を感じる」という意見が得られた．これは，学習器による推定位置を直接ハイライトしたことが原因として考えられる．今後の課題として，推定された位置情報を直接用いるのではなく，ルールベース等の手法と組み合わせることが挙げられる．

5.3 有用性評価で得られた被験者の意見について

有用性の評価実験で得られた二つの否定的な意見について，それぞれ議論する．一つ目は，ソースコード自体の読みにくさに関する意見である．今回の実験では，競技プログラミングの練習サイト上に提出されるソースコードを直接用いているが，これらは可読性を意識せずに記述されている可能性がある（例：変数名の省略やキャストなどが頻繁に用いられている）．この問題については，ソースコードの収集方法を見直すことで解決できると考えられる．

二つ目は，提示されたソースコード自体が類似している点である．これは，データベース構築の際に「複数の問題に共通して登場する機能」に対するラベルを手作業で付与したものの，その機能が過度に具体的であったことが影響していると考えられる．今後の課題としては，ラベルの種類（例：抽象度の高いラベル）の再検討が挙げられる．

6. 今後の課題

6.1 データセットについて

本研究のデータセットの構築方法について，いくつかの改善すべき点がある．一つ目は，データ数が少ないことが挙げられる．今回収集したソースコード 46,305 個のうち 29,258 個を学習に用いている．これは Code2Vec で用いた 12,636,998 個（10,072 件の GihtHub レポジトリ）と比較し，データセット数が不十分であることが予想される．二つ目は，(５章でも述べたように) データセットの作成方法である．今回は，競技プログラミングの問題ごとに一つのラベルを設定したため，同じ問題に対して提出されたソースコード全てに，同じラベルが付与されている．しかし，同じ問題向けのソースコードであっても，細部の実装方法は人によって大きく異なるため，ラベル自体が適切でない可能性がある．また，ソースコード内に問題とは関係ない部分が含まれているなど，ラベル自体がソースコードの一部分にしか対応していないケースも多数存在する．これらはいずれも，(1) 正しい出力ができればソースコードの品質自体は気にしないプログラマが多い点や，(2) 提出されたソースコードが学習モデル用の最小単位として扱うには大きすぎるといった「競技プログラミングのソースコード」自体の品質に起因すると考えられる．

6.2 ユーザテストについて

今回の被験者数は計８人であったため，定量的な評価（例：有意差検定）を行うことが困難であった．今後の課題として，被験者数を増やし，本システムのプログラミング学習に対する有用性を定量的に示すことが挙げられる．また，被験者に実際にソースコードを記述してもらうことで，システムに関する感想（例：使いやすさ）を回答してもらうアンケート形式の被験者実験も実施する予定である．

6.3 アニメーション技術などによる発展

本システムは，(1) 可視化手法の種類の選択や (2) 位置の指定といった複雑な設定を行わずに，ソースコードの内容を示す図の可視化を実現した．しかしその反面，可視化方法が静止画像（テンプレート図）によるものに限定されており，実行結果の各ステップ（数値の変化）を可視化することは難しい．そこでアニメーション形式の可視化機能を本システムに組み込むことで，更なるプログラミング学習支援が可能になると考えられる．具体的には，配列オブジェクトの使用を検知したたびに自動可視化する機能などが考えられる．

7. むすび

本研究では，任意のソースコードに対し，「可視化方法の自動選択」と「メインの挙動を示す箇所を推定」を行うプログラミング支援システムを提案した．具体的には，競技プログラミングの練習サイト上の問題とソースコードを基に，十三種類の可視化方法（ラベル）を推定する機械学習モデルを構築した．また，二種類の主観評価実験を実施することで，提案システムの精度及び有用性を示した．今後の課題としては，データセットの拡張による機械学習モデルの精度向上や，アニメーションなどのより複雑な可視化機能を追加することが挙げられる．

参考文献

[1] Dien, H. E. and Asnar, Y. D. W.: OPT+Graph: Detection of Graph Data Structure on Program Visualization Tool to Support Learning, *Proceedings of 5th International Conference on Data and Software Engineering (ICoDSE '18)*, IEEE, pp. 1–6 (online), DOI: 10.1109/ICODSE.2018.8705794 (2018).

[2] Egan, M. H. and McDonald, C.: Program Visualization and Explanation for Novice C Programmers, *Proceedings of the Sixteenth Australasian Computing Education Conference (ACE '14)*, Vol. 148, AUS, Australian Computer Society, Inc., p. 51–57 (online), available from ⟨https://dl.acm.org/doi/10.5555/2667490.2667496⟩ (2014).

[3] Kang, H. and Guo, P. J.: Omnicode: A Novice-oriented Live Programming Environment with Always-on Runtime Value Visualizations, *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, New York, USA, ACM, pp. 737–745 (online), DOI: 10.1145/3126594.3126632 (2017).

[4] Moraes, P. and Teixeira, L.: Willow: A Tool for Interactive Programming Visualization to Help in the Data Structures and Algorithms Teaching-Learning Process, *Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES '19)*, New York, USA, ACM, pp. 553–558 (online), DOI: 10.1145/3350768.3351303 (2019).

[5] Oka, A., Masuhara, H. and Aotani, T.: Live, Synchronized, and Mental Map Preserving Visualization for Data Structure Programming, *Proceedings of the 2018 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '18)*, New York, USA, ACM, pp. 72–87 (online), DOI: 10.1145/3276954.3276962 (2018).

[6] Park, J.: Algorithm-Visualizer, `https://algorithm-visualizer.org/`.

[7] Šarić, S.: Algoviz: A Project Focusing on Visualzing A⋆ Pathfinding Algorithm, `https://algoviz.njanjo.com/` (2021).

[8] OpenDSA-Project: OpenDSA, `https://opendsa-server.cs.vt.edu/`.

[9] Baloukas, T.: JAVENGA: JAva-based Visualization Environment for Network and Graph Algorithms, *Computer Applications in Engineering Education*, Vol. 20, No. 2, pp. 255–268 (online), DOI: 10.1002/cae.20392 (2012).

[10] Becker, K. and Beacham, M.: A Tool for Teaching Advanced Data Structures to Computer Science Students: An Overview of the BDP System, *Proceedings of the Second Annual CCSC on Computing in Small Colleges Northwestern Conference*, Evansville, USA, Consortium for Computing Sciences in Colleges, p. 65–71 (online), available from ⟨https://dl.acm.org/doi/10.5555/369274.369319⟩ (2000).

[11] Smith, C., Strauss, J. and Maher, P.: Data Structure Visualization: The Design and Implementation of an Animation Tool, *Proceedings of the 48th Annual Southeast Regional Conference (SE '10)*, New York, USA, ACM, pp. 72:1–72:10 (online), DOI: 10.1145/1900008.1900105 (2010).

[12] Stasko, J. T.: Tango: A framework and system for algorithm animation, *Computer*, Vol. 23, No. 9, pp. 27–39 (online), DOI: 10.1109/2.58216 (1990).

[13] Chen, T. and Sobh, T.: A Tool for Data Structure Visualization and User-defined Algorithm Animation, *Proceedings of 31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. (Cat. No.01CH37193)*, Vol. 1, IEEE, pp. 1–27 (online), DOI: 10.1109/FIE.2001.963845 (2001).

[14] Vrachnos, E. and Jimoyiannis, A.: Design and Evaluation of a Web-based Dynamic Algorithm Visualization Environment for Novices, *Procedia Computer Science*, Vol. 27, pp. 229–239 (online), DOI: 10.1016/j.procs.2014.02.026 (2014).

[15] AtCoder-Inc.: AtCoder, `https://atcoder.jp/` (2021).

[16] The University Of Aizu: Aizu Online Judge, `http://judge.u-aizu.ac.jp/onlinejudge/index.jsp` (2021).

[17] Lytle, N., Floryan, M. and Barnes, T.: Effects of a Pathfinding Program Visualization on Algorithm Development, *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19)*, New York, USA, ACM, p. 225–231 (online), DOI: 10.1145/3287324.3287391 (2019).

[18] Alon, U., Zilberstein, M., Levy, O. and Yahav, E.: Code2Vec: Learning Distributed Representations of Code, *Proceedings of the ACM on Programming Languages*, Vol. 3, New York, USA, ACM, pp. 40:1–40:29 (online), DOI: 10.1145/3290353 (2019).

[19] JetBrains-Research: Astminer, `https://github.com/JetBrains-Research/astminer` (2021).