Vol.2021-CE-162 No.13
Vol.2021-CLE-35 No.13
2021/12/4

プログラミング演習の軌跡：学生のコーディング過程理解のための教師支援

谷口 雄太[1,a)]　峰松 翼[2,b)]　大久保 文哉[2,c)]　島田 敬士[2,d)]

概要：プログラミング演習中の学生を支援するためには、学生の学習行動を把握することが重要であるが、学生が書いているソースコードを教師が常に確認し続けることは不可能であり、プログラミング教育における大きな困難の 1 つとなっている。教師支援のための代表的なアプローチの 1 つとしては、ダッシュボードシステムが挙げられるが、これまでのダッシュボードシステムでは、表面的な情報しか提供できておらず、コーディングのプロセスにまで踏み込んで情報を提供するものはほとんど存在しない。本研究は、学生のコーディングプロセスに関して有益な情報を教師に提供することを目的として、高頻度のソースコードスナップショットデータに基づく 2 つの教師向けのフィードバックツールを提案する。プログラミング演習担当教員による評価実験を行った結果、これらのツールは、教師が学生の学習状況をよりよく理解するために有用な情報を提供することが示された。

Programming Activity Trajectories: Supporting Teachers in Understanding Student Coding Processes

1. はじめに

プログラミングは重要なスキルの 1 つとして位置付けられるようになってきており，多くの大学では理系の学生だけでなく，文系の学生にもプログラミング演習授業が提供されている．プログラミング学習を成功させる上で学習意欲の高さは重要な要因とされているが [1], [2]，多くの学生はプログラムを書くのに悪戦苦闘しており [3]，適切に支援を行わなければ学習者の意欲を低下させてしまい，最終的には脱落に繋がる恐れもある．

しかし，実際に支援の必要な学生を特定することは容易ではない．困難に遭遇した時に誰かに助けを求めることは難しいメタ認知スキルの 1 つとして知られており [4]，支援の必要な学生が教師に助けを求めることは必ずしも期待できない．そのため教師は学生の学習状況に積極的に注意を払う必要がある．しかし，プログラミング演習科目の担当教員に対して行った我々の調査からは，提出されたソースコードについてはある程度把握できているが，提出までの間に書かれたソースコードや演習中にエラーが発生して困っている学生については，十分に把握しきれていないことが示唆されている．そのため，最終的な成果物だけでなく，学生のプログラミングの過程についても容易に把握できるようにするための教師支援が重要である．

クラスの学習状況を教師に提示する代表的な方法の 1 つはダッシュボードである [5], [6]．ダッシュボードではいくつかの統計情報や指標を可視化することで、状況の概要を容易に把握することができるようになっている．しかし、従来のシステムではソースコードの長さや発生したエラーの数など、演習の過程の表面的な情報しか提示できておらず，実際に学生が書いたコードの内容にまで着目した研究は少ない。学習者のソースコードが変化する過程には、プログラム構築や演習問題への取り組み方といった戦略が反映されていると考えられ，これらを把握できれば個別の対応のみならず，授業全体の運営を最適化できる可能性がある．

そこで本稿では，学習者が書いたソースコードの一連の高頻度スナップショットデータを用いて，教師に有益なフィードバックを提供をする手法を提案する．ソースコー

[1] 九州大学 情報基盤研究開発センター
[2] 九州大学 システム情報科学研究院
[a)] yuta.taniguchi.y.t@gmail.com
[b)] minematsu@ait.kyushu-u.ac.jp
[c)] fokubo@ait.kyushu-u.ac.jp
[d)] atsushi@ait.kyushu-u.ac.jp

ドの状態変化を直感的な方法で可視化することで，クラスの演習状態について容易に把握できるようにする．またプログラミング演習科目担当教員による評価実験を行い，授業の振り返りにおける提案手法の有効性を確認する．

2. 提案手法

本稿ではコーディング過程の可視化により，学生の演習活動の視覚的な理解を可能にするツールを開発する．教師は状況に応じて個人ごとの対応とクラス全体の対応の双方を行う必要があることから，異なる粒度で情報提供を行えるよう 2 種類のツールを提案する．1 つは「ソースコードフローツール」と呼ばれ，各学習者の演習過程をソースコードのレベルで容易に確認できることを目標とする．もう 1 つのツールは「クラスタフローツール」と呼ばれ，ソースコードよりは粗いものの，クラス全体の演習活動を包括的に確認することができることを目標とする．

2.1 ソースコードフローツール

ソースコードフローツールはソースコードの分布の可視化，及び選択したソースコードの情報を確認できる属性パネルの 2 つから構成されている．ソースコードの分布はソースコード間の近さ（類似性）に基づき，1 つのソースコードを 1 つの点として散布図の形で表示する．ソースコードの近さは文字列としての編集距離により測る．また，散布図として可視化する際の座標は t-SNE アルゴリズム [7] により定める．

t-SNE は著名な次元削減手法であり，入力された多数の高次元ベクトルデータをそれらの距離関係を可能な限り保ったまま低次元ベクトルに変換することで可視化を行うのによく使われている．ソースコードをベクトル表現する手法も存在する [8], [9], [10], [11] が，それらの多くはソースコードの抽象構文木表現が構築可能なことを仮定しており，本研究が対象とするようなコーディング途中の構文的に正しくないソースコードを扱うことができない．そこで本研究ではソースコードの距離を編集距離により測り，それをできるだけ保存するような 2 次元ベクトル表現を求めるのに t-SNE を利用する．

編集距離を使う理由の 1 つは，コーディング活動の量がソースコード間の近さとして現れる点である．これにより散布図上における点の近さの意味が理解しやすいものになる．また，プログラミングは自由度が高く，同じ内容でも異なる書き方を行うことができる．編集距離であれば書き方の違いを含めて類似性を測ることができ，クラス全体における書き方の傾向を分析することが可能となる．

図 1 はソースコードフローツールの初期状態でのユーザーインターフェイスを示している．図中で星マークは授業時間での最初のコード，旗マークは授業時間内の最後のコードをそれぞれ表している．また緑色はソースコードが

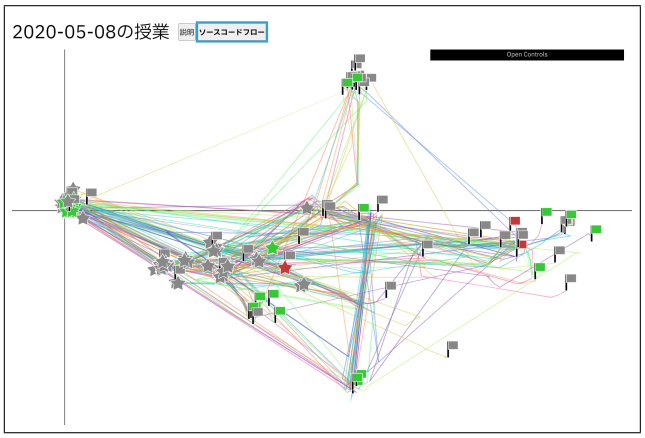

図 1: ソースコードフローツール（初期画面）．星マークは授業時間での最初のコード，旗マークは授業時間内の最後のコードをそれぞれ表している．緑色はソースコードが実行されてエラーが無かったことを表し，赤色はエラーがあったことを示す．

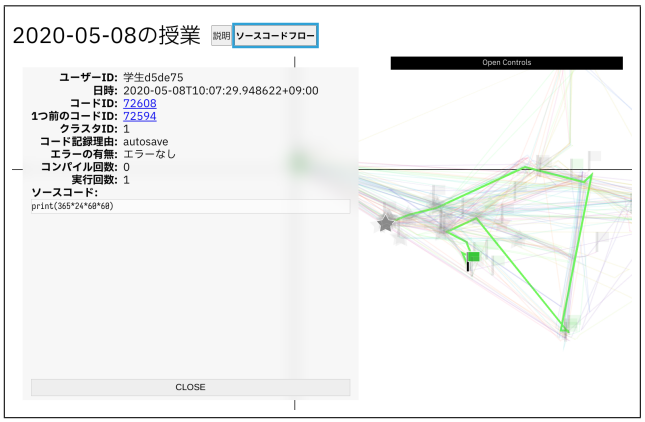

図 2: 選択されたソースコードの情報と，そのソースコードを書いた学生の軌跡．ソースコードが選択されると，選択されたソースコードとその学生の軌跡が強調して表示される．

実行されてエラーが無かったことを，赤色はエラーがあったことを，灰色は実行されなかったことを示す．この図では中間状態のソースコードについてはマークが表示されていないが，ユーザーは必要に応じて表示するように設定することができる．色付きの線は同一学生のソースコードを時系列順に結んだ軌跡を表しており，異なる学生は異なる色で表示されている．

マークをクリックすると対応するソースコードについての情報が図 2 のように左側に表示される．また同時に，ソースコードの所有者である学生の活動が目立って表示されるよう，他の学生の活動は薄く表示されるようになる．表示されるソースコードの情報にはユーザー ID，スナップショットの日時，ソースコードの ID，1 つ前の状態のソースコードの ID，ソースコードが属するクラスタの ID（後述），スナップショットがとられた理由，エラーの有無，コ

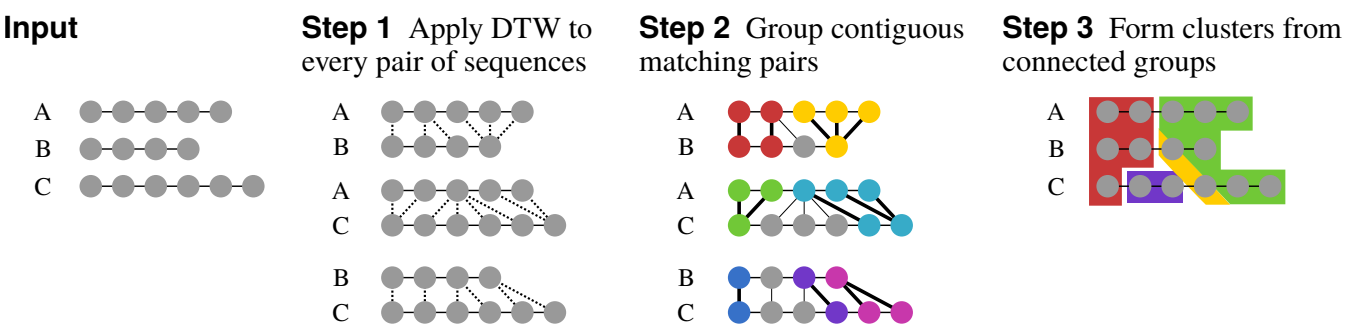

図 3: 提案するプロセスクラスタリングアルゴリズムの図解.

ンパイルの有無，実行の有無，そしてソースコードの実際の内容が含まれる．

このツールを使うことで，例えば，学生らがどのように課題を解き進めていったかを確認することができる．この図は 1 回分の授業における活動を表示している．この授業では複数の課題が出題されており，マークが密集している箇所はそれぞれが 1 つの課題に対応していると考えることができる．そのため，学生の軌跡がこれらの密集箇所をどんな順番で通過しているかを調べることで，それぞれの演習の進め方を把握できる．さらに軌跡の大部分が束になっていれば，クラス全体として同じように演習を進めていたと考えることができる．

また，密集している箇所から外れたマークについてはソースコードが特異な状況にあると考えることができ，例えば課題を解き終わって独自に演習を進めている場合や課題を解けずに試行錯誤している場合などが考えられる．このようなソースコードをクリックすれば，教師は実際のソースコードを確認して，どのような状況にあるか，その前後のソースコードはどうなっているか，というコーディングの過程について調べることができる．

このようにソースコードツールは演習活動の進行について多くの情報を提供するが，必ずしもこれらの情報全てが必要という訳ではない．例えば，クラス全体としての進み方を知りたい場合には，共通パターンをより強調するような情報提供の仕方が適していると考えられる．先に見たように，マークが密集したり軌跡が束になっている様子から，演習活動には一定のパターンがあると考えられ，そのようなパターンを陽に示すことができればある種の情報要約を行えると考えられる．そこで本研究では，もう 1 つのツールとしてクラスタフローツールを提案する．

2.2 クラスタフローツール

クラスタフローツールはクラスタレベルでの演習活動遷移を可視化したものであり，ここではまずクラスタを発見するためのアルゴリズムについて説明する．図 3 はアルゴリズムを図解したものである．アルゴリズムはソースコード系列の集合を入力として受けとる．1 つのソースコード系列は 1 人の学生に対応し，1 セッション（例えば授業など）におけるソースコードのスナップショットを時系列順に並べたものである．

アルゴリズムの最初のステップは Dynamic Time Warping (DTW) [12] アルゴリズムを用いて任意の 2 つの系列について要素ごとのマッチングをとることである．図ではマッチングがとられた異なる系列の要素間に点線が引かれている．

次のステップでは，要素のグルーピングを行う．先程と同様に任意の系列対について，マッチングがとれている要素対を時系列順に確認していき，以下のルールによりグルーピングを行う．

- 両方の要素が空文字列でない場合，相違率（後述）を計算してその値が閾値未満の場合は同じグループにまとめる．
- どちらかの要素が空文字列の場合，両方とも空文字列である場合は同じグループにまとめる．

この時，時間的に隣接する要素対については，原則として同じグループを継続する．もし先のルールによりグループにまとめない場合は，次の要素対から新しいグループを開始する．図 3 ではそれぞれの色が異なるグループを表している．また，太い線は相違率が閾値未満であることを示している．2 つの要素（ソースコード）x, y の**相違率** $\mathrm{diffratio}(x, y)$ を以下のように定義する．

$$\mathrm{diffratio}(x,y) = \begin{cases} 0 & (|x| = 0 \vee |y| = 0) \\ \frac{\mathrm{editdistance}(x,y)}{\max(|x|,|y|)} & (\text{otherwise}), \end{cases} \tag{1}$$

最後のステップでは 1 つ前のステップで作ったグループをマージしてクラスタを構成する．任意のグループ対について，1 つ以上の要素を共有するか確認し，共有している場合は両者をマージする．この時，いずれのグループにも属していない要素については，同一系列内の連続する要素の並びが単一のクラスタを形成するようにする．図 3 では最終的なクラスタ分けを色付きの多角形により表している．

アルゴリズムにおける相違率の閾値はクラスタリングの粒度を決めるパラメータと考えることができる．本研究では 0.2 を閾値として利用する．

Vol.2021-CE-162 No.13
Vol.2021-CLE-35 No.13
2021/12/4

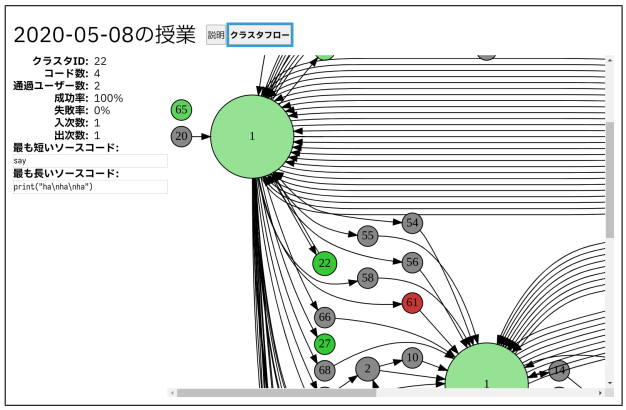

図 4: クラスタフローツール.

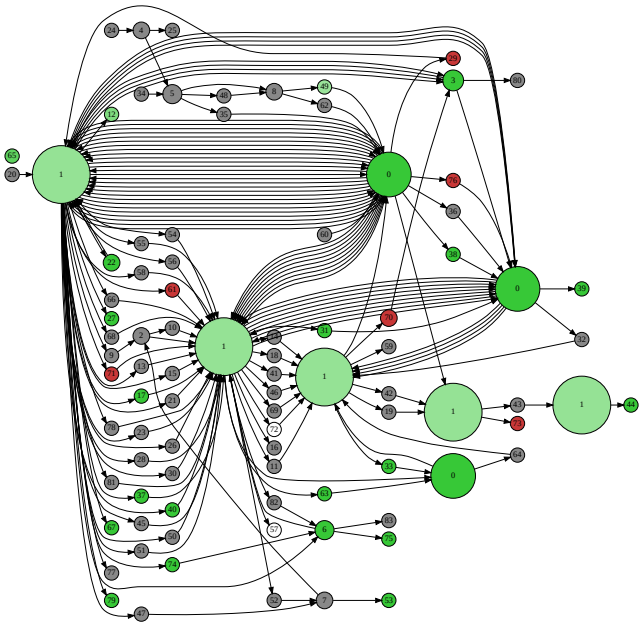

図 5: クラスターフロー図の一例.

図 4 はクラスタフローツールのユーザーインターフェイスを示している．クラスタフローツールもソースコードフローツールと同様に，クラスタ群の可視化とクラスタの情報を確認できる属性パネルの 2 つから構成されている．図 5 はツール内に表示される図を取り出したものである．図中でクラスタは円として表示されている．円の大きさはクラスタに属するソースコードを書いたユーザーの数に対応している．クラスタ間の矢印は，クラスタ間に遷移があったことを示している．クラスタの色についてはソースコードフローツールと同様である．

ソースコードフローツールと同様に，クラスタをクリックするとクラスタに関する情報が左側に表示される．表示される情報には，クラスタの ID，属するソースコード数，属するユーザー数，エラーがあったソースコードの割合，エラーがなかったソースコードの割合，グラフのノードとして見た時の入次数と出次数，クラスタに属する最短のソースコード，及び最長のソースコードがある．

図 6: オンラインプログラミング学習支援環境 WEVL.

3. 評価実験

提案する 2 つのツールの有用性を確かめるために評価実験を行った．実験でツールが使用するデータとしては，著者らの 1 人が担当した 2020 年度前期の九州大学の Python の演習授業において収集したデータを用いた．全授業回のうち 3 回分を対象とし，授業を 1 回分ずつ振り返ることができるようにした．プログラミング演習科目担当の経験がある大学教員 7 名に，ツールを使って学習者の状況把握を行ってもらうよう依頼し，その後アンケートに回答してもらった．

データの収集には我々が開発したオンラインプログラミング学習支援環境である WEVL を利用した．WEVL では学習者のソースコードのスナップショットが記録されるタイミングは 3 つある。1 つはオートセーブで、入力の手が止まってから 3 秒後に記録される。3 秒以内にエディタに何らかの入力があった場合は、そこから再度数えて 3 秒後に記録される。2 つ目は実行のタイミングであり，プログラムの実行時にソースコードがまだ記録されていなければスナップショットの記録を行う。3 つ目はシステム内の表示を切り替えたタイミングでの記録である。ユーザーは主にエディタ機能を利用するが、それ以外の一部の機能を利用する場合に，表示の切り替えが必要であり，そのタイミングでソースコードがまだ記録されていなければスナップショットの記録を行う．

表 1 および表 2 はそれぞれのツールに対するアンケート内容と結果を示している．アンケートは 5 段階のリッカート尺度に基づくもので，各設問項目にどの程度同意できるかを 1 から 5 までの数値で回答してもらった．1 は「まったくそう思わない」，5 は「そう思う」に対応する．表中では回答の平均値と標準偏差をアンケート結果として示している．

表 1 からソースコードフローツールが設問 SF-1 について 4.43 という高いスコアを得ていることが分かる．これはツールが個人のコーディング活動についての詳細をうまく提供できていることを示している．また特にソースコードという観点からは，教師らが問題の起きているソースコー

表 1: アンケート結果：ソースコードフローツール ($N = 7$).

設問 ID	質問	応答	
		平均	標準偏差
SF-1	「ソースコードフロー」はその時々のソースコードについて，その内容や実行時エラーの有無といった状態を確認することで，学生それぞれの演習の過程を細かく把握するのに役立つ.	4.43	0.53
SF-2	「ソースコードフロー」はどのようなコードでつまずきがあったか把握するのに役立つ.	4.29	0.76
SF-3	「ソースコードフロー」は学生が書いたコードの代表例を把握するのに役立つ.	3.57	0.79
SF-4	「ソースコードフロー」はクラスにおけるメジャーなプログラミング活動とマイナーな活動のそれぞれを把握するのに役立つ.	4.14	0.69
SF-5	「ソースコードフロー」は注意を向けるべき学生を把握するのに役立つ.	3.86	1.07
SF-6	「ソースコードフロー」は教師が授業を振り返るのに役立つ.	4.00	0.82

表 2: アンケート結果：クラスタフローツール ($N = 7$).

設問 ID	質問	応答	
		平均	標準偏差
CF-1	「クラスタフロー」はクラス全体の演習の進み方を，ソースコードのクラスタを単位とする状態遷移パターンとして大まかに把握するのに役立つ.	4.14	0.90
CF-2	「クラスタフロー」はどのようなコードでつまずきがあったか把握するのに役立つ.	3.71	1.25
CF-3	「クラスタフロー」は学生が書いたコードの代表例を把握するのに役立つ.	4.29	0.76
CF-4	「クラスタフロー」はクラスにおけるメジャーなプログラミング活動とマイナーな活動のそれぞれを把握するのに役立つ.	3.57	0.53
CF-5	「クラスタフロー」は教師が授業を振り返るのに役立つ.	3.86	0.69

ドの特定に有益であると考えていることが SF-2 のスコアから窺える．対照的に，典型的なコード例を見つける上ではあまり役立たないことも分かった（SF-3）．コーディング活動のパターンについては，共通性の高いパターンや普通とは異なる活動を簡単に把握できると評価されている（SF-4）．学生については，スコアは非常に高いという訳ではないものの，注意すべき学生を把握するのに有益であることが示唆されている（SF-5）．全体としてソースコードフローツールは重要なコードや学生，コーディング活動などを見つけるのに有益な機能を備えていることが示された．また，教師らはツールが授業の振り返りをするのに有益であるという点についても同意している（SF-6）．

表 2 からは，クラスターフローツールがクラスタレベルの遷移パターンを通してクラス全体の進捗を確認するのに有用であると評価されたことが分かった（CF-1）．つまずきのあったソースコードの特定については，ソースコードフローツールと比較すると低いものの，CF-2 の回答からはある程度の有用性が評価されたと言える．反対に，学生が書いた代表的なコードの把握については，クラスターフローツールの方が高いスコアを得ている（CF-3）．また，コーディング活動の把握という観点の設問 CF-4 のスコアは高くなかった．これは，時間的に連続する一連のソースコードがクラスタにまとめられてしまったために，ソースコードレベルでの状況把握ができなくなったからだと考えられる．最後に，授業の振り返りに有用かどうかという点については，教師らはある程度の同意を示したが，顕著に高いスコアという訳ではなかった．全体として，クラスタフローツールは肯定的なフィードバックを得ており，教師が学生のコーディングの過程を把握するのに適切なツールとなっているといえる．

さらに，アンケートには自由記述の設問もあり，ツールについての意見などがあれば記入してもらった．将来的な実装を最も期待された機能は両ツールの相互連携機能であった．教師らは一方のツールから他方のツールへとジャンプできるような機能が欲しいと記述していた．次に要望の多かった機能は授業中に使えるリアルタイムな情報提供機能であった．現在のツールは授業の振り返りを目的として設計されているため，リアルタイムな状況確認までは行うことができない．このようなリアルタイムフィードバック機能の実現は，我々の将来的な課題の 1 つである．

4. さいごに

本稿では，プログラミング演習授業における教師への支援として，学生のコーディング過程の把握に資する情報を教師にどのようにしてフィードバックするかという問題に取り組んだ．ソースコードの高頻度スナップショットデータを利用して，学生のソースコードが変化する過程を可視化するツールを開発した．また，ソースコード系列のクラスタリング手法を提案して，クラスタレベルでの変遷を可視化するツールも開発した．アンケートによる評価実験か

ら，これらのツールが学生のコーディング過程の把握に有用なツールとなっていることが示された．

本研究の限界としては，リアルタイムに情報提供をできていない点がある．現状では授業後の支援しか実現できていないため，今後は授業中の支援を行えるよう提案したツールの改良を行う予定である．

謝辞 本研究は科研費若手研究 JP21K17863 の支援を受けた．

参考文献

[1] Jenkins, T.: The motivation of students of programming, *Proceedings of the 6th annual conference on Innovation and technology in computer science education*, pp. 53–56 (2001).

[2] Law, K. M., Lee, V. C. and Yu, Y.-T.: Learning motivation in e-learning facilitated computer programming courses, *Computers & Education*, Vol. 55, No. 1, pp. 218–228 (2010).

[3] Robins, A., Rountree, J. and Rountree, N.: Learning and teaching programming: A review and discussion, *Computer science education*, Vol. 13, No. 2, pp. 137–172 (2003).

[4] Aleven, V., McLaren, B., Roll, I. and Koedinger, K.: Toward meta-cognitive tutoring: A model of help seeking with a Cognitive Tutor, *International Journal of Artificial Intelligence in Education*, Vol. 16, No. 2, pp. 101–128 (2006).

[5] Fu, X., Shimada, A., Ogata, H., Taniguchi, Y. and Suehiro, D.: Real-time Learning Analytics for C Programming Language Courses, *Proceedings of the Seventh International Conference on Learning Analytics & Knowledge*, LAK '17, New York, NY, USA, ACM, pp. 280–288 (online), DOI: 10.1145/3027385.3027407 (2017).

[6] Diana, N., Eagle, M., Stamper, J., Grover, S., Bienkowski, M. and Basu, S.: An Instructor Dashboard for Real-Time Analytics in Interactive Programming Assignments, *Proceedings of the Seventh International Learning Analytics & Knowledge Conference*, LAK '17, New York, NY, USA, Association for Computing Machinery, p. 272–279 (online), DOI: 10.1145/3027385.3027441 (2017).

[7] Van der Maaten, L. and Hinton, G.: Visualizing data using t-SNE., *Journal of machine learning research*, Vol. 9, No. 11 (2008).

[8] Alon, U., Zilberstein, M., Levy, O. and Yahav, E.: Code2Vec: Learning Distributed Representations of Code, *Proc. ACM Program. Lang.*, Vol. 3, No. POPL, pp. 40:1–40:29 (online), DOI: 10.1145/3290353 (2019).

[9] Azcona, D., Arora, P., Hsiao, I.-H. and Smeaton, A.: user2code2vec: Embeddings for profiling students based on distributional representations of source code, *Proceedings of the 9th International Conference on Learning Analytics & Knowledge*, pp. 86–95 (2019).

[10] Wang, K., Su, Z. and Singh, R.: Dynamic Neural Program Embeddings for Program Repair, *International Conference on Learning Representations*, (online), available from ⟨https://openreview.net/forum?id=BJuWrGW0Z⟩ (2018).

[11] Wang, L., Sy, A., Liu, L. and Piech, C.: Learning to Represent Student Knowledge on Programming Exercises Using Deep Learning., *International Conference on Educational Datamining* (2017).

[12] Berndt, D. J. and Clifford, J.: Using dynamic time warping to find patterns in time series., *KDD workshop*, Vol. 10, No. 16, Seattle, WA, USA:, pp. 359–370 (1994).