Scalability in Online Computer Programming Education: Automated Techniques for Feedback, Evaluation and Equity

Jessica McBroom, Kalina Yacef and Irena Koprinska
School of Computer Science, University of Sydney, Sydney, NSW 2006, Australia
{jmcb6755, kalina.yacef, irena.koprinska}@sydney.edu.au

ABSTRACT

The delivery of programming courses online offers great promise to provide quality programming education in an accessible manner. However, it also introduces new challenges, including how to maintain course quality as the ratio of students to teaching staff increases. In particular, the provision of effective feedback, detailed course evaluations and the promotion of equity can all become more challenging as the size of a course increases. This work explores, integrates and develops potential data mining and artificial intelligence techniques that could be utilised to address these issues in the context of programming education.

Keywords

programming education, automated feedback, course evaluation, equity, computer science education, student behaviour

1. INTRODUCTION

In recent decades, online programming courses have become increasingly numerous, with a diverse range of languages being taught on a variety of platforms. Due to their online nature, these courses are highly accessible and available to a large number of students. In addition, they do not require teachers and students to be in close proximity, making them more robust to disruptions from global events, such as the COVID19 pandemic [1]. Considering these benefits, methods for increasing the effectiveness of these courses are of great significance, with the potential to benefit many thousands of students.

One important challenge in educational settings is ensuring there is effective information flow between teachers and students. In particular, there should firstly be a way for instruction and feedback to flow from teachers to students so that students can learn. In addition, there should also be a way for information about student understanding and progress to flow from students to teachers so that teachers can adapt and improve the instruction and feedback (referred to as "closing the loop" [3]). This challenge becomes increasingly significant in the context of online education, where teachers and students may not be in direct contact and there may be a large number of students per teacher.

Some approaches to improving information flow in online education have focused on opening communication channels between teachers and students. For example, discussion boards [16], one-on-one chats with tutors [7] and student surveys [17] all allow teachers to provide instruction to or receive feedback from students. One issue with these approaches, however, is that they can suffer from scalability issues. For example, as the ratio of students to teachers increases, it becomes more difficult for teachers to monitor every discussion board question, to engage in every chat or to carefully read all open-ended survey responses. In addition, approaches to dealing with this, such as making surveys quantitative, can limit the detail of the feedback.

Since direct communication channels for closing the loop can suffer from scalability issues, another approach is to introduce automated systems as interfaces between teachers and students. In particular, teachers can configure an automated system for providing instruction and feedback to students, thereby allowing information to flow from teachers to students. In addition, automated systems can be used to analyse student behaviour and report back information about student progress to teachers, which they can use to re-configure the system, forming a loop as shown in Figure 1. This model can then be extended to consider additional loops, such as between students and the feedback system. Such systems are a highly promising approach to providing scalable education to students.

This work focuses on addressing a few key challenges associated with this approach that are both particularly important in the context of programming education and also less developed in the field at large. In particular, it first considers automated feedback techniques, and how these can be integrated together to gain a coherent picture of the current state of the field. In addition, it develops new EDM techniques for analysing student behaviour in order to evaluate a course generally, and also in the context of equity. As such, this work acts as a step towards improving the scalability and effectiveness of online programming education.

2. CONTRIBUTIONS

2.1 HINTS: A Framework for Automated Hints

Jessica Mcbroom, Kalina Yacef and Irena Koprinska "Scalability in Online Computer Programming Education: Automated Techniques for Feedback, Evaluation and Equity" In: *Proceedings of The 13th International Conference on Educational Data Mining (EDM 2020)*, Anna N. Rafferty, Jacob Whitehill, Violetta Cavalli-Sforza, and Cristobal Romero (eds.) 2020, pp. 802 - 805

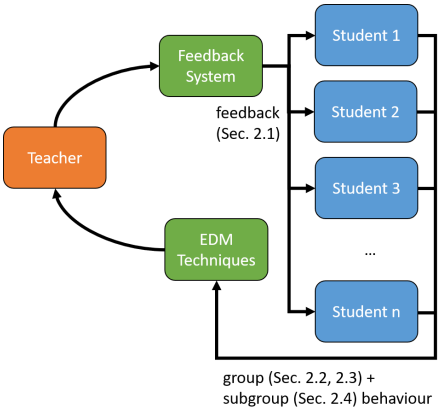

Figure 1: An example of scalable information flow between teachers and students. First, the teacher configures a system to provide automated feedback to students. Then, EDM techniques are used to analyse the behaviour of students and inform the teacher of their progress, thereby "closing the loop". The teacher can then make improvements to the system based on this information. The section labels indicate how the work described in the next section relates to this model.

When undertaking a programming course, students are often presented with various programming exercises to complete in order to gain practical experience. Since students commonly find these tasks challenging, an important component of an automated feedback system is the ability to provide hints to students experiencing difficulty on programming tasks. Such hints could include suggestions about where the student went wrong, how to proceed or concepts to revise.

A wide range of interesting techniques have been developed to provide automated hints to students, including hints that use data from peers, model solutions and test cases and utilise a diverse range of methods to produce hints, including neural networks [2], Markov Decision Processes [15], program synthesis techniques [6] and a variety of other approaches [4, 5, 13]. In addition, systems employing automated hint techniques differ with respect to the programming language taught, evaluation method, student cohort and learning context. While this diversity offers great promise for producing high quality hints and represents the great interest in the area, it also increases the difficulty of understanding the types of techniques that are available and how they fit together.

The first contribution of this work, described in detail in [9], is a framework and survey to draw these techniques together into a coherent picture. In particular, the contribution is the *Hint Iteration by Narrow-down and Transformation Steps* (HINTS) framework. This framework focuses on understanding hint techniques by decomposing the process they use to produce hints into a series of smaller steps, which can then be fit into two categories: narrow-down and transformation steps. For example, in [14] the first step is to find the closest solution to a student's program, and the second step is to then find edits from the student's program to that solution. Once hint techniques are considered as a series of smaller steps, it becomes much easier to relate techniques based on their steps to see how they fit together and the potential avenues for further developing them.

A summary of the five most important insights gained by surveying automated hint techniques in this manner are as follows:

1. Even if hint techniques appear very different overall, they can be related together by their smaller steps. This allows, for example, for them to be collected together into a single diagram to provide a coherent picture of the current state of hint techniques.
2. It is important to consider individual steps of hint techniques when evaluating, discussing and developing hint techniques, since these can often be mixed and matched.
3. there appears to be a theoretically motivated relationship between some hint technique steps and data-driven evaluation techniques, which would be interesting to explore further
4. The question of why hint techniques can be fit together in this way provides insight into the nature of automated hint generation. In particular, the fact that hint techniques can be decomposed into a series of smaller steps that fit into only two simple categories suggests this structure may be necessitated by the problem in general.
5. Further work on hint technique evaluation methods is necessary - there are so many possible combinations of steps that the efficiency of evaluation techniques must be improved.

Since understanding the types of hint techniques that are available and how they fit together is an important step in developing and evaluating automated feedback systems, this work contributes to improving such systems, which are an important component of scalable programming education.

2.2 A Technique for Clustering Student Programs

An important aspect of understanding student learning in a programming course is the ability to visualise the behaviour of students on programming exercises. In particular, understanding how beginner students behave during the first few exercises is of particular importance, since these students are more likely to make many mistakes, be least equipped to correct these mistakes and potentially be discouraged from the area if they experience too much difficulty.

The second contribution of this work is a technique for clustering beginner student programs in order to visualise trends in student behaviour when completing programming tasks. This technique, described in more detail in [12], involves first applying transformations to group logically equivalent programs. The resulting groups are then further combined if the structures of the programs in the groups are the same up to a customisable threshold and they pass the same test cases. In this way, programs with a similar functionality are clustered together, and it is possible to understand all possible programs in a cluster using a single sample program from the cluster.

After the clustering is performed, the clusters can be organised into a network with edges showing how students transition between clusters as they work on an exercise. Sequential pattern mining can also be performed to discover the most frequent transitions. This can then reveal information about where students experience difficulty (indicated by loops in the network) and the strategies they follow to complete an exercise. Example applications are also discussed in [12].

Planned extensions of this work include a more thorough evaluation beyond the case study in [12], and also an exploration of this technique in the context of equity, as discussed in Section 2.4.

2.3 DETECT: A General Clustering Technique for Temporal Trends in Student Behaviour

Many EDM techniques utilise clustering to understand student behaviour since clustering can condense highly complex information into simpler and more manageable subsets. This can allow the results to be more easily interpreted and thereby provide greater insight into student behaviour. However, while existing techniques can be readily applied to discover different types of student behaviour, it can often be more challenging to use them to discover particular behavioural trends in time. This is because the objective functions they use, which guide the cluster formation, often do not consider temporal information.

The third contribution of this work is DETECT (***D**etection of **E**ducational **T**rends **E**licited by **C**lustering **Ti**me-series data*), a customisable hierarchical clustering algorithm for detecting trends in student behaviour over time. This algorithm, described in detail in [11], produces clusters similar in structure to a decision tree, with clusters defined by decision rules (e.g. a cluster may be all examples where the time taken was ≤ 5 mins and the student's grade was A). To form these clusters, DETECT uses a customisable objective function, which governs the types of temporal trends that are found. For example, these could include behavioural changes between the start and end of a course, or behaviours that make an exercise stand out from the ones before and after it. The algorithm then works by recursively dividing the examples into subsets in the manner that maximises the objective function.

Some advantages of DETECT in an educational context are as follows:

1. It is applicable to a wide range of educational datasets. A core feature of educational courses is that they tend to have a repeating structure. For example, they may have a series of lectures, homework tasks, assignments, tutorials, readings or practice exercises, which each have a similar structure. As such, these courses can be divided into time steps with similar features, which is the type of data DETECT is applicable to. For example, time steps could be homework tasks and features could include the time taken and grade. This allows DETECT to be applied to a wide range of educational data set, including programming data.
2. The objective function is customisable and has few constraints. In [11], two different objective function examples are given: one for finding differences between the start and end of a course, and one for finding behaviours that characterise a particular exercise. However, this function can be customised to find other types of trends with minimal constraints (e.g. the function doesn't need to be differentiable). This allows DETECT to be highly flexible.
3. The results are easy to interpret. Since the clusters are defined by decision rules, it is easy to understand exactly which examples belong to each cluster, thereby improving the interpretability of the results
4. The algorithm is more robust to dependencies between features than traditional clustering methods, since the objective function can use temporal information to evaluate cluster quality. In particular, DETECT places higher weight on features that reveal interesting trends in time beyond what has already been found. As such, highly correlated features are penalised, making DETECT more robust to dependencies between features.

Planned extensions of this work include a deeper exploration into potential objective functions beyond the two discussed in [11] and a more thorough evaluation of the algorithm.

2.4 Adapting Techniques to Explore Equity Issues

One important issue when applying data mining techniques to student data is that patterns from under-represented groups can sometimes be obscured by collective trends. For example, if clustering is used to find the general types of student behaviour overall, this may obscure the potentially unique behaviour of subgroups, especially if they are small. As such, an important aspect of closing the loop between teachers and students is ensuring that analysis techniques not only provide information about the majority of students, but also minority groups.

The fourth contribution of this work is an exploration of equity issues in the context of programming education, and how the proposed techniques might be adapted to provide information about under-represented student groups. This work is still in progress, but so far has included:

1. an exploration of gender differences in enrolment and exercise completion rates in a series of programming courses, described in [10]. In particular, the courses were run for school students during a 5 week Python programming challenge in Australia in 2018, and included both block-based and text-based courses of different difficulty levels. In general, there were approximately twice as many male enrolments as female, but little difference in exercise completion rates between genders. Such an analysis acts not only as an important first step in understanding the nature of student differences in a course, but also as a baseline with which to compare data mining techniques
2. adapting DETECT to consider gender and school grade differences among students in the lead up to them dropping out. [8] In particular, this involved selecting 10 evenly spaced out programming exercises for each student who dropped out (i.e. the first exercise they completed, the last exercise they completed and

8 equally spaced out exercises in between). These exercises could then be used as time periods that were relative to the students (i.e. time 1 was when he student first began, time 2 was 10% of the way through their interaction with the course, and time 10 was the last exercise they completed before dropping out). DETECT could then be applied to this data to see the largest changes in student behaviour approaching dropping out, which could then be filtered based on gender and school grade to observe differences in the lead up to dropout for different groups.

Planned extensions to this work include adapting the program clustering technique from Section 2.2 to consider equity, and to extend the analyses by considering data from consecutive years.

3. CONCLUSION

In the context of programming education, one approach to increasing information flow between teachers and students in a scalable manner is to introduce automated systems as interfaces between teachers and students. This work aims to address a few key theoretical challenges in working towards this, with a focus on the automated techniques necessitated by such an approach. In particular, it presents a framework for integrating automated programming hint techniques, two clustering techniques for analysing student behaviour and an exploration of how such techniques can be adapted to investigate equity issues. As such, it acts as a step towards increasing information flow between teachers and students in a scalable manner, with the ultimate aim of improving programming education.

4. ADVICE SOUGHT

Any general feedback on this work, including its contributions and the problems it addresses would be most welcome, particularly in the context of thesis writing. Additionally, any suggestions for improving the coherence or completeness of the work, or other ideas for improvement would be of great value. For context, this work has been conducted over 2.5 years of PhD study, with up to 1.5 years remaining.

5. REFERENCES

[1] 290 million students out of school due to covid-19: Unesco releases first global numbers and mobilizes response. https://en.unesco.org/news/290-million-students-out-school-due-covid-19-unesco-releases-first-global-numbers-and-mobilizes. Accessed: 14/04/2020.

[2] S. Bhatia and R. Singh. Automated correction for syntax errors in programming assignments using recurrent neural networks. *arXiv preprint arXiv:1603.06129*, 2016.

[3] D. Clow. The learning analytics cycle: closing the loop effectively. In *Proceedings of the 2nd international conference on learning analytics and knowledge*, pages 134–138, 2012.

[4] B. Edmison and S. H. Edwards. Applying spectrum-based fault localization to generate debugging suggestions for student programmers. In *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 93–99. IEEE, 2015.

[5] A. Gerdes, B. Heeren, J. Jeuring, and L. T. van Binsbergen. Ask-elle: an adaptable programming tutor for haskell giving automated feedback. *International Journal of Artificial Intelligence in Education*, 27(1):65–100, 2017.

[6] A. Head, E. Glassman, G. Soares, R. Suzuki, L. Figueredo, L. D'Antoni, and B. Hartmann. Writing reusable code feedback at scale with mixed-initiative program synthesis. In *Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale*, pages 89–98, 2017.

[7] B. Jeffries, T. Baldwin, M. Zalk, and B. Taylor. Online tutoring to support programming exercises. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*, pages 56–65, 2020.

[8] J. McBroom, I. Koprinska, and K. Yacef. How does student behaviour change approaching dropout? a study of gender and school year differences. *Unpublished, accepted at EDM2020.*

[9] J. McBroom, I. Koprinska, and K. Yacef. A survey of automated programming hint generation - the HINTS framework. *arXiv preprint arXiv:1908.11566 (Unpublished, submitted to ACM Computing Surveys)*, 2019.

[10] J. McBroom, I. Koprinska, and K. Yacef. Understanding gender differences to improve equity in computer programming education. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*, pages 185–194. ACM, 2020.

[11] J. McBroom, K. Yacef, and I. Koprinska. DETECT: A hierarchical clustering algorithm for behavioural trends in temporal educational data. *arXiv preprint arXiv:2005.10640 (Unpublished, accepted at AIED2020.)*.

[12] J. McBroom, K. Yacef, I. Koprinska, and J. R. Curran. A data-driven method for helping teachers improve feedback in computer programming automated tutors. In *International Conference on Artificial Intelligence in Education*, pages 324–337. Springer, 2018.

[13] B. Paaßen, B. Hammer, T. W. Price, T. Barnes, S. Gross, and N. Pinkwart. The continuous hint factory-providing hints in vast and sparsely populated edit distance spaces. *arXiv preprint arXiv:1708.06564*, 2017.

[14] T. Price, R. Zhi, and T. Barnes. Evaluation of a data-driven feedback algorithm for open-ended programming. In *Proceedings of the 10th International Conference on Educational Data Mining*. International Educational Data Mining Society, 2017.

[15] T. W. Price, Y. Dong, and T. Barnes. Generating data-driven hints for open-ended programming. In *Proceedings of the 9th International Conference on Educational Data Mining*. International Educational Data Mining Society, 2016.

[16] J. Suler. In class and online: Using discussion boards in teaching. *CyberPsychology & Behavior*, 7(4):395–401, 2004.

[17] S. Watson. Closing the feedback loop: Ensuring effective action from student feedback. *Tertiary education and management*, 9(2):145–157, 2003.