The Development of Computational Thinking Concepts in Course Participants' Programming Solutions

Niklas Humble[a]

[a]*Mid Sweden University, Akademigatan 1 - Building Q, 831 40 Östersund, SWEDEN*

Abstract

With low student satisfaction and high drop-out rates, programming education has been labelled by many as problematic learning in higher education. Many countries have integrated programming in kindergarten to grade 12 educations to better prepare students for a digitalised society. A common concept in this context is computational thinking. In this study, the concept is applied in a teacher professional development course on fundamental programming. The aim of the study is to identify and discuss the development of computational thinking in course participants' programming solutions in a teacher professional development course on fundamental programming.

The study was conducted with an action research approach. Collected data consists of 35 course participants' programming solutions from two iterations of a teacher professional development course on fundamental programming. A combination of text mining and process mining were used to analyse the collected data. First, an algorithm was developed with the Python programming language that analyse the collected data. Second, the algorithm's output was converted to a spreadsheet for a manual control of the coding. Third, a process mining tool was used to analyse the coded data.

Results of the study show that several indicators of development in computational thinking can be identified in the programming solutions. The conclusion of the study is that computational thinking concepts in programming solutions can serve as indicators of course participants' development as programmers. This can be drawn upon by stakeholders for course completion forecast or drop-out interventions.

Keywords

Computational thinking, Programming education, Text mining, Process mining, Professional development

1. Introduction

Programming education has been labelled as problematic learning by many due to high drop-out and low course satisfaction among students [1] [2]. At the same time, forecasts for the future labour market indicates an increased need of competence in programming among professionals [3]. In an effort to meet this demand, and better prepare students for a digitalised society, many countries have integrated programming and computer science in school curriculum at K-12 level (kindergarten – grade 12) [4].

Nordic Learning Analytics (Summer) Institute 2021, KTH Royal Institute of Technology, Stockholm, 23 August 2021
niklas.humble@miun.se (N. Humble)
https://www.miun.se/en/personnel/niklashumble/ (N. Humble)
0000-0002-5791-4765 (N. Humble)

In relation to the integration of programming in K-12 education, computational thinking is a common term discussed. The basic idea of the concept is to capture the skills and ways of thinking when problem-solving through programming. [4] The concept of computational thinking is commonly used in research on programming and computer science in a K-12 setting. In this study, the concept is applied in the context of adult learning, namely teacher professional development.

The aim of the study is to identify and discuss the development of computational thinking in course participants' programming solutions in a teacher professional development course on fundamental programming. The two research questions that have guided this work are: 1) What computational thinking concepts can be identified in the programming solutions? 2) How does the computational thinking concepts in the programming solutions develop as the course progresses?

2. Theoretical framework

Computational thinking was coined by Seymour Papert and increased in popularity through the latter work of Jeanette Wing [4] [5]. According to Papert [6] computational thinking can be used as a powerful and accessible tool for learning. Wing [7] expanded on this, stating that computational thinking is a set of skills derived from computer science for problem-solving, system building and understanding the behaviour of humans.

Computational thinking has since Papert and Wing been further developed and discussed in research [8]. One such example is the theoretical framework of Brennan and Resnick [9]. The framework is based in the Scratch block programming tool, but the authors argue that the concepts in the framework are common in other programming languages as well [9].

The framework encompasses three dimension that are key to computational thinking according to the authors: 1) *computational concept*, 2) *computational practices*, and 3) *computational perspectives*. These have been identified through the study of programming activities in the online community of Scratch and workshops in Scratch. [9]

The first key dimension of the framework, computational concepts, are the concepts that the programmer engage with when programming. The framework provides seven computational concept which are: *sequences, data, operators, conditionals, loops, events,* and *parallelism.* [9]

The second key dimension of the framework, *computational practices*, are the practices that the programmer develop when programming and engaging with the computational concepts. The framework provides four main sets of computational practices: *abstracting and modularising, testing and debugging, reusing and remixing, and being incremental* and *iterative.* [9]

The third key dimension of the framework, *computational perspectives*, are the perspectives that the programmers form about themselves and the world around them as they program. The framework proves three examples of perspectives that form when engaging with programming: *expressing* (a medium for self-expression), *connecting* (creating with and for others), and *questioning* (empowering to make solutions and not rely on others). [9]

3. Method

The study has been conducted with an action research approach for studying and developing the context of the author's own work [10] [11]. The context is a teacher professional development course on fundamental programming and the author had the double role of course teacher and researcher.

The course is aimed towards K-12 teachers in mathematics and technology and teaches the fundamentals of programming in the Python programming language. The course is of 5 ECTS and given at a study pace of 25 percent, since the course participants work as teacher while taking the course. The course is structured with three full day meetings (one in the beginning, one in the middle, and one at the end of the course) with lectures, seminars and workshops. Between the course meetings the participants have course assignments in the form of programming tasks, essay writing and forum discussions in the course online learning management system.

3.1. Data collection

Data have been collected from two iterations of the course given at the fall semester of 2020 and the spring semester of 2021. For this study, the course participants programming solutions from two programming assignments in the course where collected, one in the beginning of course and one at the end of the course. The first programming assignment was to build a speed converter for miles per hour to kilometres per hour. The last programming assignment (at the end of the course) was to build a prime number sieve. In total, 54 programming solutions from 35 different course participants were collected from the two iterations of the course. 35 programming solutions from the first programming assignment and 19 from the last programming assignment.

3.2. Data analysis

The collected data were analysed with a combination of text mining and process mining in three steps. Text mining was used to identify new meanings in the collected data [12]. Process mining was used to extract knowledge from the collected data and monitor the process in the programming solutions [13].

The first step in the analysis process was to develop an algorithm in the Python programming language. This algorithm reads all the collected data and divides it in lines of code. The algorithm then removes irrelevant data, such as blank lines, assign each line of code a fictional timestamp, which is needed for the process mining, and conduct the coding of the material. The algorithm's coding of the collected material is based on the theoretical framework of computational thinking, more specifically the computational concepts presented by Brennan and Resnick [9].

However, since the course in question did not touch upon the computational concept of *parallelism*, this was removed from the algorithm's coding scheme. Instead, the use of a *main* function was added to the coding since this was a central programming convention in the course. The main coding scheme of the algorithm consisted of *data* (the use of variables and lists), *operators* (the use of mathematical operators), *conditionals* (the use of if statements and try and except constructions), *loops* (the use of for and while loops), *events* (the use of functions), *sequences* (the structure of the programming solutions), and *main* (the use of a main function).

In the second step, the algorithm's output was converted to a spreadsheet for manual control of the coding. The output consisted of a total of 1446 lines of code with additional coding, timestamps, and solution id. After the manual control of the coding, where lines of code that the algorithm had labelled as *uncertain* were sorted, the spreadsheet was imported to a process mining tool, Disco (https://fluxicon.com/), for the final step of the analysis.

In the third and final step of the analysis, the process mining tool was used to discover the general flow (*sequences*) in the programming solutions; and the solution frequency of different codes (computational concepts). The flow of the programming solutions was identified through discovering the most common path between the different codes, that is, in what order the computational concepts are used in the solutions. The frequency of codes was determined by the number of solution where the code (or combination of codes) was used at least once. The process mining was conducted with three different combinations of the collected data: 1) all the collected, 2) the collected data relating to the first programming assignment, 3) the collected data relating to the last programming assignment.

4. Results and analysis

This section is structured in two sub-headings to present and analyse the results of the study. The first sub-heading present and analyse the results relating to the first research question: What computational thinking concepts can be identified in the programming solutions? The second sub-heading present and analyse the results relating to the second research question: How does the computational thinking concepts in the programming solutions develop as the course progresses?

4.1. Identified computational concepts

In the total 54 programming solutions, collected for this study, all codes specified in the algorithm has been identified (Appendix A). The most common identified computational concept is *data*, used at least once in every programming solution collected. This is not surprising since the use of variables and lists is a central part of programming in any language. The least common code identified is the single use of *operator*, used only in 4 solutions. This is also not surprising since operators are usually used in combination with *data* (variables and lists) and *loops*, which is also the case in collected material. *Operators* are used together with *data* in 39 solutions, and together with *loops* in 13 solutions.

Regarding the *sequences* of the collected programming solutions the most common code to start with is *event*. This start path is taken in a total of 33 solutions, which is in line with the programming conventions in the course. That is, that the programming solutions should be structured in functions (*events*), since that enables reuse of the programming code. The most common code to end the solution with is *main*, which is done in 25 solutions. This is also in line with the programming conventions in the course since the use of a main-function acts as the single command for execution.

4.2. Development of computational concepts

In the 35 programming solutions that are collected in relation to the first programming solution, all codes specified in the algorithm has been identified (Appendix B). The most and least common codes in the solutions are, as in the full body of solutions (Appendix A), *data* (used in all 35) and single used *operator* (used in 4). However, almost all solutions (31) use *operators* in relation to *data* (variables and lists). *Loops* and *conditionals* are quite uncommon in the solutions (used in 7 solutions each), as is *main*, used in less than half of the solutions (14).

Regarding the *sequences* of the solutions to the first programming assignment, the most common path in the beginning and end of the solutions are still in line with the conventions in the course. That is, the most common code to start with is *event*, which is done in 19 solutions; and the most common to end the solution with is *main*, done in 13 solutions. However, it is a close call since many solutions also start with *data* (16 solutions) and ends with a combination of *data* and *operator* (12 solutions). Which is not in line with the conventions in the course.

In the 19 programming solutions that are collected in relation to the last programming solution, all codes specified in the algorithm has been identified (Appendix C). However, the most and least common codes are different from the full body of solutions (Appendix A). The most common codes are *data*, *loop*, and *conditional*, which are all used in all 19 solutions. In comparison to the full body of solutions (Appendix A) and the solutions for the first programming assignment (Appendix B), there is no longer a clear code that is uncommon. The least used code, *main*, is still used in over half of the solutions (12 solutions); and the least used combination of codes, *data* and *operator*, is used in almost half of the solutions (8 solutions).

Regarding the *sequences* of the solutions to the last programming assignment, the most common path in the beginning and end of the solutions are still in line with the conventions in the course. That is, the most common code to start with is *event*, which is done in 14 solutions; and the most common to end the solution with is *main*, which is done in 12 solutions. Further, there are some paths in the solutions that are shared by all, or almost all, solutions. All 19 solutions have a path between *conditional* and *data*. Almost all solutions (18) have a path between *loop* and *conditional*. More than half have a path between *event* and *data* (14); and between *data* and *loop* (13). Looking at these four paths together they indicate a common programming practice that is taught in the course, which is: 1) define a function (*event*), 2) declare variable or list (*data*) at the beginning of that function, 3) declare a *loop* (still in the function), 4) given the *conditionals* within that *loop*, 5) change or append to the initial declared *data*.

Comparing the solutions for the first programming assignment (Appendix B) and the last programming assignment (Appendix C), what developments can be identified in the solutions as the course progresses? The most obvious is that the most common paths between different codes in the solutions develop to be stronger (that is, shared by a larger number of solutions). The most common path in the solutions for the first assignment, between *data* and the combination code of *data* and *operator*, is only shared by 25 of 35 solutions. At the same time, that path does not say anything interesting about the solutions' programming practice. Only that the solutions first declare a variable or list, and after that declare another variable or list with the support of mathematical operators. Meanwhile, there are several paths in the solutions for the last programming assignments that are shared by all, or almost all, of the solutions. Further, these paths, as mentioned in the previous paragraph, indicates programming practices in the solutions that are taught in the course.

The solutions for the last programming assignment also show a larger number of solutions that start and end their solutions in accordance with the conventions in the course; than is done in the solutions for the first programming assignment. That is, 14 of 19 solutions start with defining a function (*event*) and 12 of 19 solutions end with calling the *main* function. The solutions for the last programming assignment also indicate a more advanced use of *loops* in the solution. While *loops* only occurs as single code in the solutions for the first assignment, they also occur as combined code with *operators* in the solutions for the last assignment. This indicate that the solutions for the last programming assignment uses *loops* that are more specified than they are in the first assignment.

5. Discussion

First, it should be mentioned that it is not surprising that all the computational concepts in the algorithm's coding scheme could be found in the solutions. The computational concepts touch upon common programming concepts that was used in the course; and the computational concept that was not used in the course, *parallelism*, was removed from the algorithm's coding scheme. What is more interesting is how the frequency and paths of computational concepts develop in the solutions between the first and last programming assignment. Which indicates that the course participants' computational thinking is developing during the course.

Again, that the solutions develop to resemble each other is expected. The solutions are collected from two iterations of the same course and what happens during a course is usually that the participants learn and adopt to the conventions that are used in the course. What is more interesting is that the general codes of the algorithm (based on computational concepts) can show this development. This can bridge the gap of programming solutions containing programming code that are different but drawing on the same general programming idea (computational concept). That is, the solutions are the same but written differently. This can be used to better identify and address the problematic learning situation of programming education with high drop-out rates and low student satisfaction [1] [2].

Computational thinking is a concept usually used in K-12 education to capture and develop the skills used in programming and computer science [4]. However, as this study show it may also be a relevant concept to draw upon in adult education. To meet the demand of professionals with programming expertise on the future labour market [3], it may not be enough to only educate the young. Papert [6] stated that computational thinking could be used as a tool for learning that is accessible and powerful. This study would also suggest that computational thinking can be used as a tool for identifying learning.

6. Conclusion

The focus of this study has been to identify and discuss the development of computational thinking in programming solutions by participants in two iterations of a professional development course on fundamental programming. Using a theoretical framework of computational thinking concepts, several indicators of computational thinking has been identified in the programming solutions of the course with the support of text mining and process mining. The frequency of computational thinking concepts in the programming solutions develop as the course progresses, as do the patterns between the different computational thinking concepts. The identified changes in frequency, patterns and combination of computational thinking concepts indicates that the course participants' programming skill develops to be more advanced and in line with programming conventions.

The conclusion of the study is that computational thinking concepts can be used for identifying the learning progression in programming solutions and serve as an indicator of how the student programmers develop. With further development of, for example, dashboards this can be drawn upon by stakeholders for course completion forecast or drop-out interventions. It can also be used as a tool for self-assessment, where the learners can keep track on their learning progression as programmers.

7. Limitations and future research

There are some limitations to this study that should be mentioned and addressed in future research. First, the coding scheme of the algorithm used for text mining the programming solution did not include all facets of the computational thinking framework. The coding scheme focused on the specific parts of the framework that was relevant and used in the context where the collected data were produced, a teacher professional development course on fundamental programming. Therefore, it could be argued that this study does not capture the entire picture of computational thinking. To better capture all different aspects of computational thinking, future studies could include multiple data sources in a multimodal learning analytics approach. This would allow the remaining key dimension of the framework, *computational practices* and *computational perspectives*, to be included in the study.

Further, the study does not address which course participants stay in teacher professional development course, which drop out, and what their previous experience in programming is. It is possible that the development of computational thinking concepts in the programming solutions would be less significant if all course participants would have stayed in the course and conducted all programming assignments. To address this, and the relative low number of programming solutions collected for this study, future research should focus on a bigger body of programming solutions. The programming solutions should be conducted by the same course participants and collected at the beginning and the end of the course, but also halfway through the course for better accuracy in the analysis. Since an algorithm is used for the main coding of the data, the volume of data is not the problem for future research but getting access to data might be a problem.

Lastly, all collected data were programming solutions in the Python programming language. The developed algorithm that was used for coding the collected data is also limited to recognising programming solutions written in the Python programming language. Future research should therefore focus on collecting programming solutions written in different programming languages and developing the algorithm to be able to recognise and code solutions in multiple programming languages. An interesting use of this would be to integrate the algorithm directly in a learning management system for real-time analysis of the submitted programming solutions. This would further allow for comparisons between different types of programming courses and different types of programming languages.

References

[1] A. S. Marcolino, E. Barbosa, A survey on problems related to the teaching of programming in brazilian educational institutions, in: 2017 IEEE Frontiers in Education Conference (FIE), IEEE, Indianapolis, IN, USA, 2017, pp. 1–9.

[2] A. Gomes, A. J. Mendes, An environment to improve programming education, in: Proceedings of the 2007 international conference on Computer systems and technologies, Association for Computing Machinery, New York, NY, United States, 2007, pp. 1–6.

[3] S. Smit, T. Tacke, S. Lund, J. Manyika, L. Thiel, The future of work in Europe, McKinsey Global Institute, 2020.

[4] J. Nouri, L. Zhang, L. Mannila, E. Norén, Development of computational thinking, digital competence and 21st century skills when learning programming in k-9, Education Inquiry 11 (2020) 1–17. doi:10.1080/20004508.2019.1627844.

[5] D. Weintrop, U. Wilensky, Robobuilder: a computational thinking game, in: SIGCSE ’13, 2013, p. 736.

[6] S. Papert, An exploration in the space of mathematics educations, International Journal of Computers for Mathematical Learning 1 (1996) 95–123.

[7] J. M. Wing, Computational thinking, Communications of the ACM 49 (2006) 33–35.

[8] V. J.Shute, C. Sun, J. Asbell-Clarke, Demystifying computational thinking, Educational Research Review 22 (2017) 142–158.

[9] K. Brennan, M. Resnick, New frameworks for studying and assessing the development of computational thinking, in: Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada, 2012, p. 25.

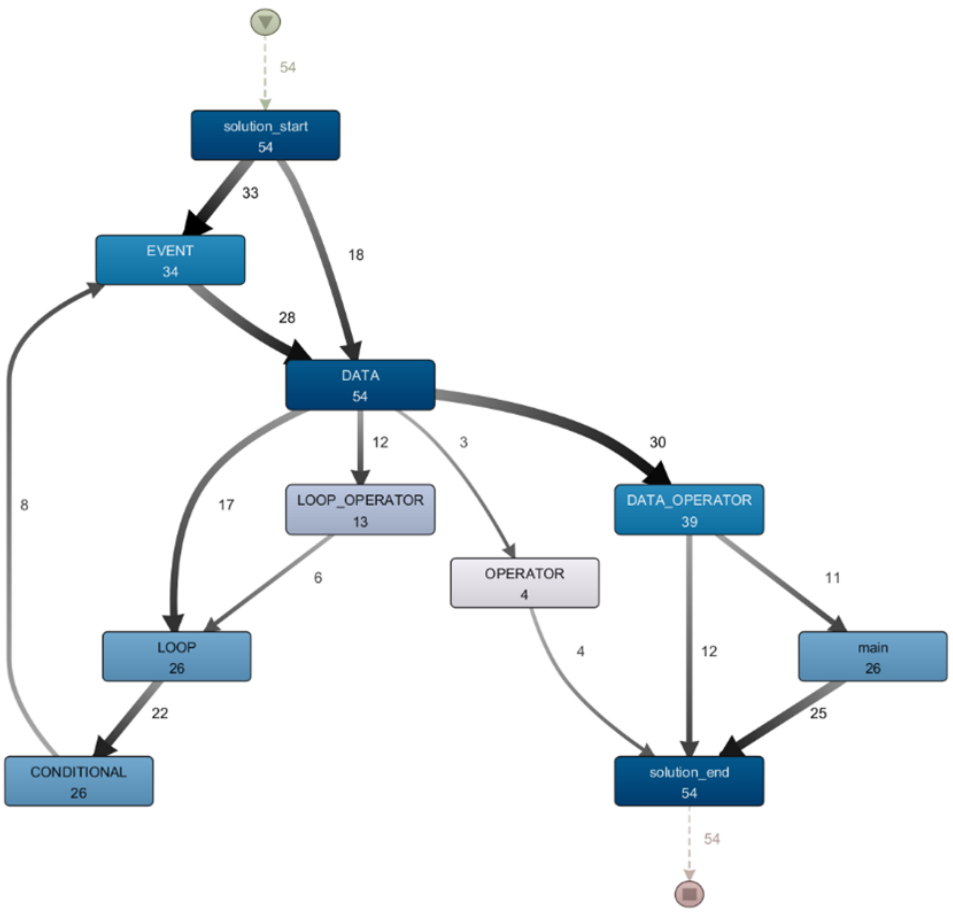
[10] A. Burns, Action research, in: J. Heigham, R. A. Croker (Eds.), Qualitative Research in Applied Linguistics, Palgrave Macmillan, London, 2009, pp. 112–134.

[11] J. McNiff, You and Your Action Research Project, 4 ed., Routledge, London, 2016.

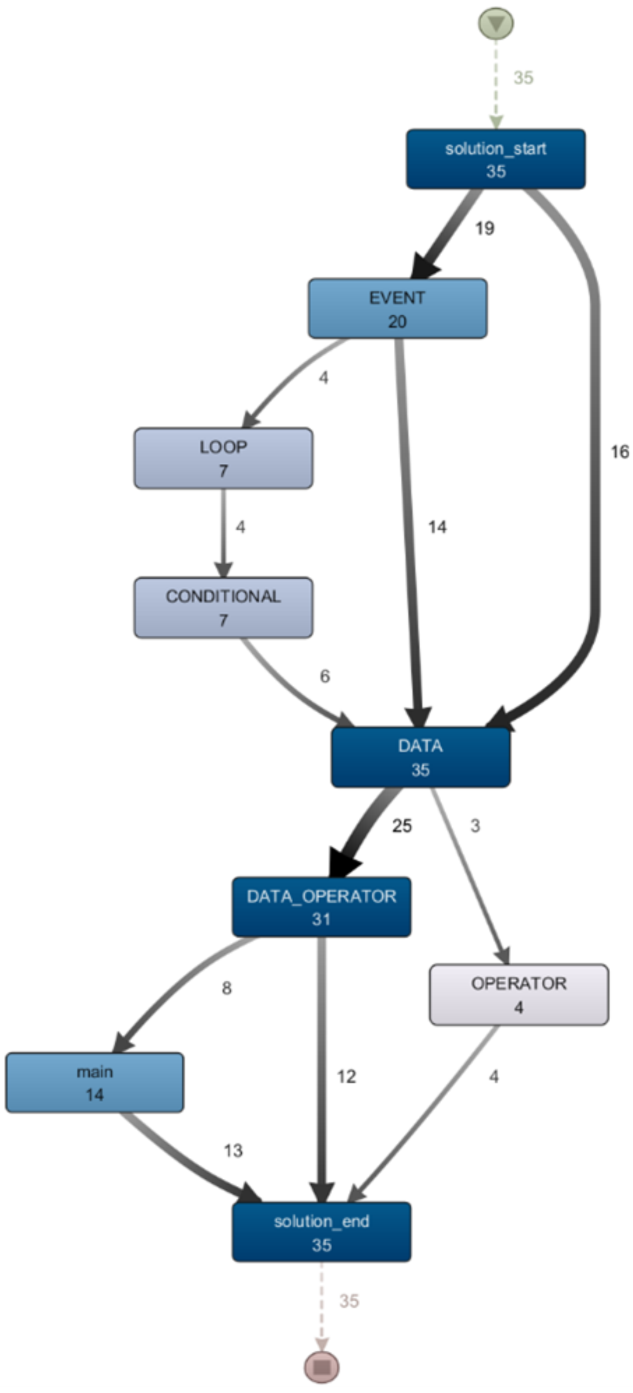
[12] M. Hearst, What is text mining?, SIMS, UC Berkeley (2003).

[13] W. van der Aalst et al., Process mining manifesto, in: F. Daniel, K. Barkaoui, S. Dustdar (Eds.), Business Process Management Workshops, Lecture Notes in Business Information Processing, Springer, Berlin, 2012, pp. 169–194.

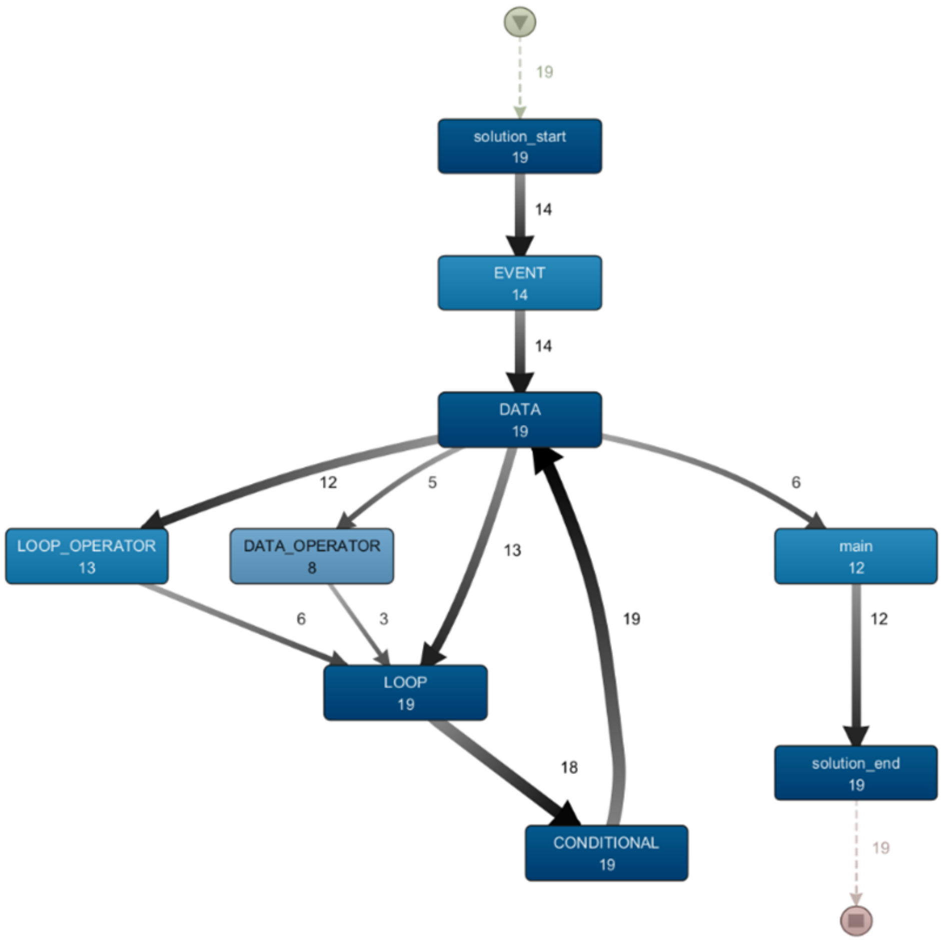
A. Figure 1: Process mining of all collected data



B. Figure 2: Process mining of first programming assignment



C. Figure 3: Process mining of last programming assignment


19
solution_start
19
14
EVENT
14
14
DATA
19
12
5
13
6
LOOP_OPERATOR
13
DATA_OPERATOR
8
main
12
6
3
19
12
LOOP
19
18
solution_end
19
CONDITIONAL
19
19