Algorithm assignments to be embedded into the learning's of Applied Algorithms

Using Sudoku to teach advanced topics

Jacob Sjöblom

June 2022

Acknowledgement

Abstract

1 Introduction

Algorithms has been a fundamental part within Computer Science since it's birth, and have been used by mathematicians for millennia before the first computer came into existence. It's importance as an area of education can easily be identified, as it's prevalent mentioned in the Association for Computing Machinery's Curriculum 68 [2], which gave recommendations to how undergrad and master grad programs could be composed in academic programs. In later iterations including the latest CC2020 report comprised by ACM together with IEEE released in December 2020, the understanding of algorithms, their implementation and complexity is considered as part of the "Software Fundamentals", and listed as competencies under the Computer Science education [10].

Although having a fundamental understanding of algorithms are important, the term itself is very broad and accounts for many different types ranging from sorting to genetic and randomized algorithms to mention a few. In addition to the variety of algorithms, the complexity and difficulty of understanding and implementing them also varies a huge amount. To address this disparity, it is often important to have a clear idea of the whats, the whys and the hows when teaching algorithms.

As part of the intended learning outcomes of the Applied Algorithms course taught at the IT University of Copenhagen, students should be able to implement algorithms from high-level descriptions, evaluate the correctness and run experiments to properly present results in regards to performance and scalability [14]. To showcase the acquirement of these skills, a set of mandatory assignments are given during the semester with relation to the taught curriculum of the course.

This thesis will address the intersection between some proposed educational topics and approaches, and how they fit into the ILO of the Applied Algorithms course, with the goal of producing an assignment that can be embedded as one of the mandatory assignments to be completed in the course. As a main focus area Sudoku was chosen as an assignment idea based on a prior research project conducted by René Schou and Jacob Sjöblom leading into the master thesis, due to its potential relatable properties and the accessibility of real life data [21].

2 Didactics within Computer Science

In order to address what an assignment should be comprised of and how it should be constructed, it is relevant to briefly examine some of the prerequisites taught prior to the course, in addition to identify potential new additions that could be suitable from a curriculum point of view. Introducing new topics into a course always holds some levels of uncertainty since it is untested in the current form of the course. Is it a feasible to teach students? How we can create an interesting assignment that shows applicability, while still being founded in the theory it addresses? Can a practical assignment follow the ILO as stated? To answer some of these questions, it is necessary to take a brief look into the current state of the curriculum and what elements constitutes a "good" assignments.

2.1 Teachings within an Algorithm Curriculum

Current prerequisites for the course include taking an "Algorithms and Data Structures" [13] course, which in turn requires some variation of "Introductory Programming" [28]. Teachings within these courses has an almost identical composition as the structure proposed by Ricardo A. Baeza-Yates [3], which in turn is compared to the ACM/IEEE Computing Curricula from 1991 [6][p.40-44]. Formally these two courses are commonly placed under "CS1-CS2". Baeza-Yates highlights a paradigm approach to teaching algorithms, which could be considered as an overarching approach for the Applied Algorithms course, or on a assignment level, a way to teach the differences by horse racing algorithms from different paradigms against each other.

Although the proposed course is dealing with the design and analysis of algorithms, the updated core curriculum despite having evolved from identifying "Knowledge Units" with specific topics to "Competencies" [10][p.111], still marks the identification of different algorithmic approaches as being one of the competencies computer science students should obtain. A similar notion is presented by Justus H. Piater, where he argues that most popular algorithmic textbooks introduces an algorithm and associates it with a problem, where real life scenarios tends to present a problem, and it is up to the individual to identify the appropriate approach to find a solution [20]. This offers the students a lot of freedom, but can be challenging to implement if taught in a sequential manner, since a good understanding of the different approaches are required.

Besides the focus on algorithmic paradigms/approaches, Baeza-Yates furthermore mentions NP-Complete problems and the SAT-problem as two potential topics, which is not currently part of the AA curriculum. These topics are additionally covered by Robert Sedgewick and Kevin Wayne under the section "Intractability" which is currently in the works [22]. This is of special interest since the book "Algorithms Fourth Edition" by the before mentioned authors, is the textbook currently used in the ADS course taught at ITU, and the topics could therefore be a natural progression from the prerequisite course.

We can reasonably illustrate the correlation between what is proposed as core curriculum for undergrad and master grad students by the community and its applicability to AA as presented in Figure 1. As illustrated the core curriculum affects the ILO which in turn is directly affecting how the course is acted out. While the course itself has little to no impact on the guidelines presented in the CC, it has a direct impact on the ILO in terms of the close connectivity between the two. To this extent it is essential that any new topics not previously included in the course, can be reflected in the ILO in order to have a meaningful spot in the execution of the course itself.

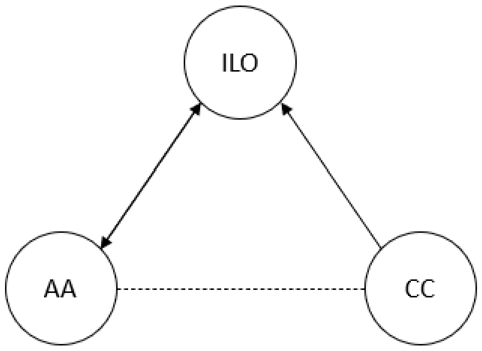

Figure 1: A simple model illustrating the curriculum relation between Core Curriculum, Intended Learning Outcomes and what is taught in the Applied Algorithms Course

To achieve this interrelation between the course and its ILO, the emphasis of applying the scientific method to theoretical topics is intrinsic. As discussed by Gordana Dodig Crnkovic, Computer Science as a field of study, has evolved from being a tool used by other sciences, into a scientific field of its own [8]. Being able to understand theoretical aspects in terms of paradigms, levels of abstraction and efficiency and translate it into an scientific experiment to measure performance and correctness, holds a lot of merit within the teachings of algorithms. Based on this notion new topics can readily be included into the curriculum of AA, on the premise that their theoretical aspect can be implemented, tested and scrutinized by applying the scientific method, as stated by the ILO [14].

2.2 Creating a ”good” assignment

Posting good assignments to students, is as important as presenting interesting and engaging topics during the semester [1] [9]. The quotation marks around good is to highlight how something that is obviously desired, can be hard to properly quantify, due to the varying subjective opinions on the matter, from students and teachers alike. To make matters worse, this single gut feeling in relation to the quality of an assignment, can be broken down into numerous variables that needs to be taken into consideration individually and carefully weighting their impact on the overall product.

Some of the notable areas that constitutes a good assignment, lies in the aspect of assignment size / running time and the problem domain it tries to address. In the before mentioned research project, a survey was conducted with 31 students participating, whom all attended the Autumn 2021 class of AA. When asked about the problem domain a vast majority (96,8%) answered they preferred assignments with an emphasis on algorithms real world application or a mixed approach between theoretical experimentation and real world application. In regards to size / running time of an assignment (61,3%) answered they saw the current structure which spans across 2-3 weeks as being reasonable [21].

Based on their experience running a CS2 course, Joseph A. Turner and Joseph L. Zachary identified when aiming towards teaching both theoretical knowledge and the know how to address and solve real-world problems, the latter is often sacrificed in the process when providing multiple smaller programming assignments for students to work with [26]. Instead they focused on a single course-long assignment split into smaller iterations to better encourage continuous development, refactoring and proper use of testing. Students were as the real-world problem assigned to create a day planner with certain capabilities in addition to encouraging them to build functionalities they thought up themselves that would entice them to use it personally. Downsides to this, is the possibility for students to fall behind, due to being busy or personal reasons, which ultimately could render them incapable of catching up to the rest of the class. This scenario however did not occur, partly credited to the availability of teaching assistants being able to work one-to-one with the students, which they in turn credited to the implementation of automated testing. The need for automated testing is also backed up by Kirsti Ala-Mutka and Hannu-Matti Järvinen, when running larger courses that includes programming, since it allows students to test their solution independent of teachers presence and gives valuable feedback to areas where students are struggling based on the logfiles, which in turn can be used for further improvement of the individual assignments [1].

On the notion of students falling behind can also be due to students underestimating the time it takes to complete an assignment as an observation made by Timothy Urness in a paper comparing performance of students working on short real-world job interview questions, to their counterpart working on long-term programming assignments. Contrary to the study done by Turner and Zachary, Urness characterizes long-term assignments as spanning between 1-2 weeks and not over an entire course [27]. From a performance perspective, students performed very similarly with the differences surrounding particular questions, where long-term students having performed the implementation of certain data structures outperformed the interview question students, but struggled on questions relating to differences between certain data structures and their properties. Urness concludes that short-term interview question assignment are useful to establish fundamentals of data structures which in large part is very similar to what is covered in ADS prerequisites course. For upper-division courses that seek to develop deeper understanding and practice tools learned, long-term "nifty" assignments are deemed more suitable. The Applied Algorithms course arguably falls better into the latter category, based on prior iterations of the course and the ILO.

Before we look further into nifty assignments, it is worthwhile to address some of the discrepancies encountered so far. When discussing real-world problems and real-world application, the term real-world seems to be used fairly ambiguously. So far we seen it used to describe a program that can be used in every day life and questions you can encounter later on in life when applying for a job as a programmer. In the paper by Daniel E. Stevenson and Paul J. Wagner on developing real-world programming assignments, they discuss some of the problems arising from using real-world data opposed to toy data [23]. Circumstances could easily arise where focus would need to be put on refactoring the code to suite the new data type which is a good lesson to learn, but might overshadow the intentions of AA to conduct scientific experiments. To clear up this ambiguity, we will for further reference, based on the definition of an algorithm [4] and the context of applying them, refer to the use of real-world data when discussing their real-world application, since the problem any algorithm seeks to solve, exists in the real-world in general. When discussing problems we will instead try to view them as being meaningful to examine or not, both from a theoretical and practical point of view. Although whether something being meaningful is still a very subjective measure, it is a bit more concrete, rather to categorize something as being real-world or not.

Nifty assignments lead by Nick Parlante from Stanford University, is a forum dedicated to presenting and sharing assignment ideas among educators including required materials, to further the goal of creating great assignments [18]. Submissions are open, and the top 6-7 new ideas are presented annually at the SIGCSE conference resulting in a distributed article highlighting the contributions [19]. Common across Nifty assignments, is presenting students with fun and clear problem statements, often revolving around a game or that it produces an entertaining output. They should be topical in order to be included in course curriculum and be independent of specific languages and platforms. Additionally they should be scaleable allowing for advanced students to expand on the initial assignment, and ideally be easily adoptable by other educators.

One of the co-administrators of the Nifty assignments Julie D. Zelenski did a case study together with Todd J. Feldman, on prerequisites necessary to design excellent assignments [9]. With the intent of teaching recursion, which can be conceptually hard for students to fully grasp, they introduced an assignment based around the game Boggle. Based on iterations made over a couple of years, of posting the same assignment, they identified that certain aspects of the exercise could be difficult and time consuming, which did not pertain to the subject they wanted to teach. In an effort to overcome this hurdle, sections of the program was offered to students, in order to keep them focused on the actual intent of the assignment. In regards to accessibility they discovered that Boggle as a game was very fitting since it did not hold a bias towards culture, gender or academic background. The game aspect itself also proved useful since it offers a clear goal with concise well defined rules. As an outcome the prerequisites they for creating a good assignment is listed as follows:

1. The material that an assignment is intended to teach must lie at the heart of the problem it poses
2. An assignment must only be challenging with respect to the material it is intended to teach
3. An assignment must be engaging
4. An assignment must be accessible to all students
5. The end results of an assignment must be worth the time and effort required to achieve it
6. Focus the assignment
7. Don't cut corners on materials provided to students
8. Don't add features indiscriminately
9. Make assignment easier for students to test and debug

The Boggle assignment later went on to become a Nifty assignment in 2002, with the assistance of Owen Astrachan [18].

Though nifty assignments provides a good basis for constructing fun and clearly defined exercises, there is plenty of room for improvement to create or modify assignments, to be meaningful in regards to practical context and social relevance [15]. In this case practical context refers to the idea of the project having some sort of value for external users, or just for the person developing it. Having students create solutions that can be utilized in every day life by others can be challenging, but providing students with assignments that can be related to the context of what they see and experience in the world, could encourage them and help them to better conceptualize why the things they are learning are meaningful. Arguably this can be difficult to achieve in relation to performing scientific experiments on algorithms in order to understand and verify that their implementation conforms to their theoretical properties. Though perfect fitting examples can be hard to come by, actions can be made to bridge the gap by utilizing data from the real world in addition to providing context to what issues are being addressed in the field of algorithms, and the impact solving these issue would potentially have on society.

2.3 Sudoku as an algorithmic topic

Sudoku as a puzzle game is a special variation of the Latin-Squares game and was created by the Swiss mathematician Leonhard Euler in the late 18th Century. The modern version as we know today is credited to the puzzle inventor and architect Howard Garns, whom got it published in a magazine for the first time in 1979. Since then Sudoku as numbers puzzle saw a huge surge in the mid 2000's and became a world phenomenon [24].

Beside being widely popular and well known around the world, Sudoku has been proven to be a NP-Complete problem by Takayuki Yato in 2003 [25] as part of his research on Another Solution Problems. This was proven by performing the reduction to Latin-Squares which in turn was proven to be a NP-Complete problem by Charles J. Colbourn back in 1984 [5]. In relation to being NP-Complete, a Sudoku is also an intractable problem, since as of now, no polynomial time algorithms is proven to be able to solve them [11][p.8]. This is an interesting subject to touch upon as students are often only exposed to polynomial algorithms in introductory courses, as is also the case at ITU. In a study conducted by Modestina Modestou and Athanasios Gagatsis, they identified that students of all ages had a tendency to apply linearity and proportional reasoning to most problems, even when not applicable [17]. Therefore introducing students to more non-polynomial problems and how to address them, seems like a proper challenge to highlight some of the complexities one can face within the domain of algorithms.

To help introduce some of these concepts to students, based on a NSF-funded project, Andrea F. Lobo and Ganesh R. Baliga, identified and proposed eighth different laboratory based projects revolving around Sudoku [16]. In their research five different combinations involving seven of the projects with the optional addition of applying the scientific method for algorithmic comparison was suggested. When adopting one of the versions into an Design and Analysis of Algorithms course, 52 out of 53 students were able to demonstrate an understanding of NP-Completeness and concepts. In regards to writing a correct proof of NP-Completeness, 30 out of 53 students managed to do so. Though students didn't perform as well on writing a correct proof, this parameter is not as essential in the context of this use-case, since the focus is centered around the application of algorithms as opposed to the design part. This sort of question would probably be more applicable to the Algorithm Design course also taught at ITU [12].

Inadvertently when deciding on paradigms/ algorithmic approaches to focus on, the choices fell on a combination, identical to one of the project sequences suggested by Lobo and Baliga namely: Brute force, Backtracking and SAT-Solvers. As opposed to their proposal of running these as separate projects throughout the course to cover certain aspects, all approaches will be covered in a single assignment with the addition of an scientific experiment to performance test them against each other. These approaches were selected due to their utilization of recursion and to show an use-case for a general solver, since any NP problem can be reduced to an instance of a Boolean satisfiable problem in polynomial time, as described by the Cook-Levin theorem [7].

3 An analysis of Sudoku

3.1 A Naive Brute-force Approach

3.2 Algorithm X

3.3 SAT-Solvers

4 Sudoku as an algorithmic assignment

5 Conclusion

6 Future work

References

[1] K. Ala-Mutka and H.-M. Jarvinen. "Assessment process for programming assignments". In: *IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings.* 2004, pp. 181–185. DOI: `10.1109/ICALT.2004.1357399`.

[2] William F. Atchison et al. "Curriculum 68: Recommendations for Academic Programs in Computer Science: A Report of the ACM Curriculum Committee on Computer Science". In: *Commun. ACM* 11.3 (Mar. 1968), pp. 151–197. ISSN: 0001-0782. DOI: `10.1145/362929.362976`. URL: `https://doi.org/10.1145/362929.362976`.

[3] Ricardo A. Baeza-Yates. "Teaching Algorithms". In: *SIGACT News* 26.4 (Dec. 1995), pp. 51–59. ISSN: 0163-5700. DOI: `10.1145/219817.219828`. URL: `https://doi.org/10.1145/219817.219828`.

[4] Britannica. *Algorithms and Complexity.* URL: `https://www.britannica.com/science/computer-science/Algorithms-and-complexity`. (accessed: 2022-04-16).

[5] Charles J. Colbourn. "The complexity of completing partial Latin squares". In: *Discrete Applied Mathematics* 8.1 (1984), pp. 25–30. ISSN: 0166-218X. DOI: `https://doi.org/10.1016/0166-218X(84)90075-1`. URL: `https://www.sciencedirect.com/science/article/pii/0166218X84900751`.

[6] *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force.* Tech. rep. New York, NY, USA, 1991.

[7] Stephen A. Cook. "The Complexity of Theorem-Proving Procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing.* STOC '71. Shaker Heights, Ohio, USA: Association for Computing Machinery, 1971, pp. 151–158. ISBN: 9781450374644. DOI: `10.1145/800157.805047`. URL: `https://doi.org/10.1145/800157.805047`.

[8] Gordana Dodig Crnkovic. "Scientific Methods in Computer Science". In: (Dec. 2002).

[9] Todd J. Feldman and Julie D. Zelenski. "The Quest for Excellence in Designing CS1/CS2 Assignments". In: *SIGCSE Bull.* 28.1 (Mar. 1996), pp. 319–323. ISSN: 0097-8418. DOI: `10.1145/236462.236564`. URL: `https://doi.org/10.1145/236462.236564`.

[10] CC2020 Task Force. *Computing Curricula 2020: Paradigms for Global Computing Education.* New York, NY, USA: Association for Computing Machinery, 2020, pp. 49, 111. ISBN: 9781450390590. DOI: `10.1145/3467967`. URL: `https://doi.org/10.1145/3467967`.

[11] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* Mathematical Sciences Series. W. H. Freeman, 1979, pp. 1–14. ISBN: 9780716710448. URL: `https://books.google.com.br/books?id=fjxGAQAAIAAJ`.

[12] Thore Husfeldt. *Algorithm Design: Intended Learning Outcomes.* URL: `https://learnit.itu.dk/local/coursebase/view.php?ciid=1004`. (accessed: 2022-04-16).

[13] Riko Jacob and Thore Husfeldt. *Algorithms and Data Structures: Intended Learning Outcomes.* URL: `https://learnit.itu.dk/local/coursebase/view.php?ciid=849`. (accessed: 2022-04-12).

[14] Riko Jacob and Matti Karppa. *Applied Algorithms: Intended Learning Outcomes.* URL: `https://learnit.itu.dk/local/coursebase/view.php?ciid=1044`. (accessed: 2022-04-12).

[15] Lucas Layman, Laurie Williams, and Kelli Slaten. "Note to Self: Make Assignments Meaningful". In: *SIGCSE Bull.* 39.1 (Mar. 2007), pp. 459–463. ISSN: 0097-8418. DOI: `10.1145/1227504.1227466`. URL: `https://doi.org/10.1145/1227504.1227466`.

[16] Andrea F. Lobo and Ganesh R. Baliga. "A Project-Based Curriculum for Algorithm Design and Np-Completeness Centered on the Sudoku Problem". In: *J. Comput. Sci. Coll.* 32.3 (Jan. 2017), pp. 110–118. ISSN: 1937-4771.

[17] Modestina Modestoz and Athanasios Gagatsis. "Students' Improper Proportional Reasoning: A result of the epistemological obstacle of "linearity"". In: *Educational Psychology* 27.1 (2007), pp. 75–92. DOI: `10.1080/01443410601061462`. eprint: `https://doi.org/10.1080/01443410601061462`. URL: `https://doi.org/10.1080/01443410601061462`.

[18] Nick Parlante. *Nifty Assignments.* URL: `http://nifty.stanford.edu/`. (accessed: 2022-04-14).

[19] Nick Parlante et al. "Nifty Assignments". In: *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 2.* SIGCSE 2022. Providence, RI, USA: Association for Computing Machinery, 2022, pp. 1067–1068. ISBN: 9781450390712. DOI: `10.1145/3478432.3499268`. URL: `https://doi.org/10.1145/3478432.3499268`.

[20] Justus H. Piater. “Planning readings: a comparative exploration of basic algorithms”. In: *Computer Science Education* 19.3 (2009), pp. 179–192. DOI: 10.1080/08993400903255226. eprint: https://doi.org/10.1080/08993400903255226. URL: https://doi.org/10.1080/08993400903255226.

[21] René Schou and Jacob Sjöblom. “An exploratory analysis of algorithms to be embedded into the learning's of Applied Algorithms”. In: *Research Project* (2022).

[22] Robert Sedgewick and Kevin Wayne. *Algorithms Fourth Edition*. URL: https://algs4.cs.princeton.edu/66intractability/. (accessed: 2022-04-12).

[23] Daniel E. Stevenson and Paul J. Wagner. “Developing Real-World Programming Assignments for CS1”. In: *SIGCSE Bull.* 38.3 (June 2006), pp. 158–162. ISSN: 0097-8418. DOI: 10.1145/1140123.1140167. URL: https://doi.org/10.1145/1140123.1140167.

[24] Sudoku.com. *The History of Sudoku*. URL: https://sudoku.com/how-to-play/the-history-of-sudoku/. (accessed: 2022-04-16).

[25] Yato Takayuki and Seta Takahiro. “Complexity and Completeness of Finding Another Solution and Its Application to Puzzles”. In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 86-A (2003), pp. 1052–1060.

[26] Joseph A. Turner and Joseph L. Zachary. “Using Course-Long Programming Projects in CS2”. In: *SIGCSE Bull.* 31.1 (Mar. 1999), pp. 43–47. ISSN: 0097-8418. DOI: 10.1145/384266.299674. URL: https://doi.org/10.1145/384266.299674.

[27] Timothy Urness. “Using Interview Questions as Short-Term Programming Assignments in CS2”. In: *J. Comput. Sci. Coll.* 32.5 (May 2017), pp. 170–177. ISSN: 1937-4771.

[28] Mahsa Varshosaz. *Introductory Programming: Intended Learning Outcomes*. URL: https://learnit.itu.dk/local/coursebase/view.php?ciid=736. (accessed: 2022-04-12).