A PROJECT-BASED CURRICULUM FOR ALGORITHM DESIGN AND NP-COMPLETENESS CENTERED ON THE SUDOKU PROBLEM*

Andrea F. Lobo and Ganesh R. Baliga
Computer Science Department
Rowan University
Glassboro, NJ 08028
(856) 256-4805
lobo,baliga@rowan.edu

ABSTRACT

Algorithms and complexity are fundamental to Computer Science (CS). This paper presents the Sudoku version of an NSF-funded, project-based curriculum for algorithm design that includes strategies for intractable problems and NP-Completeness. This curriculum is a sequence of laboratory projects comprising increasingly sophisticated solvers for a single NP-Complete problem. The curriculum is designed to integrate into existing, one-term, undergraduate courses that teach algorithm design and/or intractability without sacrificing traditional course content. The curriculum includes feasible approaches for tackling intractable problems, such as probabilistic algorithms. Three versions of the curriculum have been developed to facilitate its sustained adoption, each centered on a well-known problem: Traveling Salesperson (TSP), Satisfiability (SAT) and Sudoku. This paper presents the version of the curriculum that is centered on the Sudoku Problem, including student learning outcomes, descriptions of eight laboratory projects, many recommended project sequences, and preliminary assessment results.

INTRODUCTION

Algorithms and complexity are fundamental to Computer Science (CS). This paper presents the Sudoku version of an NSF-funded, project-based curriculum for algorithm design that includes strategies for intractable problems and NP-Completeness. This curriculum is a sequence of laboratory projects comprising increasingly sophisticated solvers for a single NP-Complete problem[1,2]. The curriculum is designed to integrate into existing, one-term, undergraduate courses that teach algorithm design and/or intractability without sacrificing traditional course content. Three versions of the curriculum have been developed to facilitate its sustained adoption, each centered on a well-known problem: Traveling Salesperson (TSP)[3,4], Satisfiability (SAT) and Sudoku. Project descriptions, rubrics, data sets, solutions, recommended one-term project sequences, and an adopter forum are available[5].

The curriculum is designed to target the following student learning outcomes:

- To understand classic algorithm design techniques,
- to implement algorithms for intractable problems using more advanced algorithm design techniques,
- to understand NP-Completeness, and
- to apply the scientific method.

Over the past decade, the authors have adopted many versions of the curriculum to teach the Design and Analysis of Algorithms course that is required for CS majors at their institution. Previous assessment efforts focused on the effectiveness of the TSP curriculum for the learning outcomes on intractability and NP-Completeness, as well as student understanding of traditional content in Algorithms and Complexity as defined by the ACM/IEEE-CS Computer Science Curricula 2013 (CS2013)[4].

This paper presents the set of eight projects centered on the Sudoku Problem and many suggested project sequences that facilitate the sustained adoption of the curriculum over many terms. In the fall semester of 2015, the authors adopted and assessed a version of the Sudoku curriculum at their institution. This paper also presents preliminary assessment results from this adoption.

A SET OF PROJECTS CENTERED ON THE SUDOKU PROBLEM

The National Science Foundation is funding the development, evaluation and dissemination of three versions of the curriculum, each centered on a different problem. This paper presents the version centered on the Sudoku Problem. The typical Sudoku puzzle that one finds in newspapers or pastime books consists of a square 9x9 grid with pre-set values in some of its 81 cells. The grid is subdivided into 9 3x3 boxes. The objective of the puzzle is to fill each blank cell so that each column, each row, and each box contains all the digits from 1 to 9. Sudoku stated in a general from for squares of size n^2 x n^2, for arbitrary values of n, is NP-complete [6].

The Sudoku problem can be stated in a general form for squares of size (**w** x **h**), for **w** and **h** > 0. Figure 1 illustrates a Sudoku puzzle of size (3 x 2) with box boundaries. Each box has width **w** and height **h**. Some of the $(\mathbf{w} \times \mathbf{h})^2$ cells in the grid contain preset

values; in Figure 1, we show the preset values in their corresponding cells and an underscore in the blank cells. A solution to a Sudoku puzzle contains all the integers from 1 to (**w** x **h**) in each row, each column, and each box. A Sudoku puzzle of size (**w** x **h**) consists of **w** rows of boxes, and each row consists of **h** boxes.

1 _ 3 4 _ 6	4 _ _ _ 3 _
2 _ 1 5 _ _	2 _ _ _ 6 _
3 _ 2 6 - 5	_ 4 _ _ _ 2

Figure 1. A Sudoku instance of size (3 x 2).

A set of eight laboratory projects centered on the Sudoku problem has been developed. The learning outcomes of these projects include the application of traditional algorithm design strategies, strategies for intractable problems, and applying the scientific method. Several projects have options and variants. One project can be assigned very early in the term. Additionally, one laboratory project applies problem reduction as a problem solving strategy and allows the students to leverage an existing codebase, the published product of earlier scientific work. Two projects allow the students to apply the scientific method in computer science. Projects 1 through 3 use traditional algorithms content. Projects 4 and 5 apply an incomplete or probabilistic algorithm design strategy. Detailed project descriptions, variants, solutions, input sets, rubrics and a faculty forum are available on the website for the curriculum[5]. A brief description of each laboratory project follows.

Project 1: Brute force

The students are asked to develop a Sudoku solver, a program that receives a Sudoku instance and solves it. This solver uses an enumeration approach that generates all possible combinations of values for the blank cells. If the solver finds a combination of values that satisfies the Sudoku constraints, this solution is returned; if no such combination is found, then the input instance is unsolvable. The worst-case number of value combinations that this brute force enumeration approach verifies is (w x h)^(number of empty cells). The students run their solver on given puzzles and write a laboratory report with solver runtime observations.

Project 2: Backtracking

The students are asked to develop a backtracking Sudoku solver. A comparison of the runtimes of a Backtracking solver and a brute force solver is meaningful since both explore the entire problem solution space. The students write a laboratory report that includes their solver runtime observations on a given set of input instances.

Project 3: Backtracking with greedy heuristic

In this project, students are asked to develop a backtracking Sudoku solver with a greedy heuristic for selecting the next cell to be explored. A chooseNextCell method computes, for each blank cell, the number of reasonable values that this cell might take, that is, the number of values in [1 . . **w** x **h**] that do not appear in this cell's row or column or box. The method chooses a cell with the minimum number of possible values, breaking ties randomly. The students might be asked to implement the greedy heuristic as an enhancement after developing a backtracking solver, or as extra credit along with the backtracking solver. It is reasonable to require a comparison of the runtimes of a Backtracking solver with a heuristic, a Backtracking solver, and/or a brute force solver since all these approaches explore the entire problem solution space. The students write a laboratory report that includes observed solver runtimes on a given input set.

Project 4: Randomized: Hill climbing with random walk

A Sudoku solver might use a randomized approach to exploring its search space. Randomized Sudoku solvers typically execute in stages, and each stage has some number of steps. A stage starts at a random point in the search space, that is, some random assignment of values to the blank cells in the input puzzle. In each step, the solver checks whether the current point satisfies the Sudoku constraints. If it does, the solver is done. Otherwise, the solver searches for a promising point (assignment) in the neighborhood of the current point, and moves to that point in the next step. If the solver checks a maximum number of neighboring points during a step but none is promising, the current point may be a local optimum and a random neighbor is chosen to be the current point in the next step. Thus, the solver has three configuration parameters: MaxNumStages, MaxNumSteps, and MaxNumNeighbors. A solver that has executed for the configured number of stages and steps, but has not found a solution, is stopped with an inconclusive outcome. Keep in mind that a randomized exploration of the search space is not systematic, and thus a randomized solver is not able to declare that any puzzle is unsolvable.

Two heuristics come into play in the Hill climbing with random walk Sudoku solver. The first computes the neighbors of the current point. The second is an evaluation function that estimates how close a point is to a solution. The evaluation function is used to select a promising neighbor to be the next current point.

The students are asked to develop a Hill climbing with random walk Sudoku solver. The students write a laboratory report that includes runtime observations for a given input set, and possibly a comparison with other solver implementations.

Project 5: Randomized: Simulated annealing

Simulated annealing[7] is a randomized approach similar to Hill climbing. An important difference is that a simulated annealing solver moves to the neighboring point when the neighbor has an equal or better distance, or per a random function. This random function changes with time so that a move to a "worse" point in the search space becomes less and less likely over the duration of a stage. This approach derives from metallurgy

and the random function depends on the temperature of the system. The temperature is set to a high value at the start of each stage and it cools with each step per a cooling constant. This solver has four parameters: MaxNumStages, MaxNumSteps, StartingTemperature and CoolingConstant.

The students are asked to develop a randomized simulated annealing Sudoku solver and write a laboratory report that includes runtimes on a given input set. The report may include a comparison with other solvers, such the randomized Hill climbing with random walk solver. This comparison might involve connections to metallurgy, as described above, and to natural selection, where a variation of an organism is successful when it is superior to its predecessor.

Project 6: Sudoku solver via reduction to Satisfiability

The well-known, NP-Complete Satisfiability Problem (SAT) can be stated as follows: Is a given a Boolean formula satisfiable? For this project, the students are asked to develop a Sudoku solver as follows:

a. read an input Sudoku instance from a file,
b. translate the input Sudoku instance into an equivalent SAT instance[8],
c. solve the SAT instance using a public domain SAT solver[9],
d. convert the obtained solution of the SAT instance into a solution of the input Sudoku instance, and
e. output the obtained solution of the Sudoku instance.

This approach results in a very efficient Sudoku solver. It may appear to be many steps but each step is relatively simple. Implementing this solver allows the students to experience direct benefits from the products of prior scientific research. The students write a laboratory report that includes observed runtimes for a given input set and a comparison to other solvers.

Project 7: A Performance Study of Two Sudoku Solvers

In this lab project the students apply the scientific method to compare the performance of two Sudoku solvers: A hill climbing with random walk Sudoku solver and a pure hill climbing solver. The comparison is based on two aspects of solver performance: The number of instances solved by the solvers, and the amount of work that each solver does. The students conduct the scientific experiment and document it in a lab report. A specification of the format of the lab report is provided to the students. This format contains sections on Objective of the experiment, Hypothesis, Procedure, Results, Analysis and conclusion, Future work, and References.

Project 8: Fine tuning a Sudoku Solver

The students use a profiler to identify opportunities for improving the runtime of a Sudoku solver. They implement a modification to the code that is hypothesized to improve the solver's runtime and apply the scientific method to compare the performance

of both implementations. The hypothesis of the experiment is that the modified code is more efficient than the original. The students conduct the scientific experiment and document it in a lab report. A specification of the format of the lab report is provided to the students. This format contains sections on Objective of the experiment, Hypothesis, Procedure, Results, Analysis and conclusion, Future work, and References.

Many project sequences based on the Sudoku projects

Some laboratory projects ask students to apply traditional algorithm design strategies or strategies for intractable problems. Others apply reduction, or the scientific method. An adopter of the curriculum selects a sequence of some of these projects for any given term. Many project sequences can be created from the set of eight projects described above. This enables the sustained adoption the curriculum, term after term. We recommend the adoption of any of the following project sequences, with or without the addition of project 8:

- Projects 1, 2 and/or 3, and optionally 6: complete solvers, exponential complexity and reduction;
- Projects 1, 2, 4, and 6: emphasizes breadth;
- Projects 1, 3, 4 or 5, optionally 6, and optionally 7: more sophisticated breadth, heuristics, and randomized;
- Projects 3, 4 or 5, 6, and 7: skipping preliminary work, scientific method;
- Projects 4, 5, 6 , and 7: focus on design strategies for intractable problems.

ASSESSMENT

A version of the project-based curriculum centered on Sudoku was adopted and assessed in an offering of the Design and Analysis of Algorithms course at their institution during the fall semester of 2015. The following sequence of laboratory projects was adopted: Project 1 on brute force, Project 2 on backtracking, and Project 8 on the scientific method.

The traditional course topics were presented in the usual manner, applying them to Sudoku when appropriate. Perhaps the only change to the presentation order was that problem reduction may have been presented a bit earlier than in other offerings of the course. Project 1 was assigned early in the semester. Since the students were already familiarity with Sudoku, it was used as a first example in the presentation of the Backtracking algorithm design strategy.

The students produced three laboratory reports, answers to 9 quizzes, and answers to a two-hour final exam. The laboratory projects were described above. The quizzes addressed the following topics:

1. Instruction counting
2. Upper bound of runtimes
3. Recurrence relations and runtimes of recursive code

4. Heaps
5. Priority Queues
6. Prim's minimum spanning tree algorithm
7. Dynamic programming applied to coin changing
8. Backtracking with branch and bound applied to 0-1 Knapsack
9. NP-Completeness definitions and concepts

The final exam consisted of five questions on the following topics: NP-Completeness definitions and concepts, structure of an NP-Completeness proof, NP-Completeness proof for Independent Set Problem, Backtracking with branch and bound, and Dynamic Programming including complexity.

The laboratory project reports were assessed against rubrics created by the authors. These rubrics are available on the website for the project-based curriculum[5]. Assessment criteria for the projects address important concepts in algorithm design and analysis such as evidence of correctness, efficiency of identified crucial component, justification of chosen data structures, code structure, organized presentation of data, comparison of multiple TSP solvers, and data interpretation. Additional author-created rubrics were used to assess the students understanding of NP-Completeness definitions and concepts, structure of an NP-Completeness proof, and writing of a proof of NP-Completeness. Indirect assessment data was obtained via an anonymous student survey.

ASSESSMENT RESULTS

Fifty-three students used the project-based curriculum in their Design and Analysis of Algorithms course in the fall semester of 2015. All students submitted three projects that satisfied their respective rubrics, thus demonstrating that they know how to provide evidence of program correctness, implement efficient algorithms, choose efficient data structures and justify their choices, write code with appropriate structure, present data in an organized manner, compare the performance of multiple solvers for a single problem, and interpret data.

All students submitted work products that demonstrated their understanding of several algorithm design strategies and their ability to apply algorithm design strategies to multiple problems.

Based on the answers produced on the closed-book, in-class final exam, 52 out of 53 students demonstrated their understanding of NP-Completeness definitions and concepts, and all 53 students demonstrated knowledge of the structure of an NP-Completeness proof. Thirty out of 53, or 57% of the students wrote a correct proof of NP-Completeness in the closed-book, in-class final exam. The primary difficulty encountered by the students was in writing the reduction. This is consistent with the data from the student survey where 62% of respondents stated that they understand how to reduce one NP-Complete problem to another.

CONCLUSIONS AND FUTURE WORK

This paper presents the Sudoku version of an NSF-funded, project-based curriculum for algorithm design that includes strategies for intractable problems and NP-Completeness. This curriculum is a sequence of laboratory projects comprising increasingly sophisticated solvers for a single NP-Complete problem that can be integrated into existing, one-term, undergraduate courses. This paper presents the version of the curriculum that is centered on the Sudoku Problem, including student learning outcomes, descriptions of eight laboratory projects, many recommended project sequences, and preliminary assessment results from an adoption at the author's institution.

A sequence of three projects centered on Sudoku was used in a fall semester of 2015 offering of the Design and Analysis of Algorithms course. The projects were integrated into a traditional presentation of the course material. Rubrics were used to obtain direct assessment data on the effectiveness of the project-based curriculum. Direct and indirect assessment data indicate that the curriculum is effective in achieving all targeted student learning outcomes except for Writing a proof of NP-Completeness. 57% of the students wrote a correct proof of NP-Completeness in the closed-book, in-class final exam and only 62% of student stated that they understand how to reduce one NP-Complete problem to another. These results are less favorable than those obtained during an adoption of the TSP version of the curriculum during the fall semester of 2014, when 75% of students wrote a correct proof during the final exam. A possible contributing factor to this result is the absence of laboratory project on reduction in the Sudoku project sequence that was adopted in the fall of 2015. This hypothesis will be tested in future evaluations of the project-based curriculum.

ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1140753.

REFERENCES

[1] Andrea F. Lobo and Ganesh R. Baliga, NP-Completeness for All Computer Science Undergraduates: A Novel Project-Based Curriculum, *Journal of Computing Sciences in Colleges*, 21 (6), 53-63, June 2006.

[2] Andrea F. Lobo and Ganesh R. Baliga, Teaching algorithm design and intractability with a project-based curriculum centered on a single intractable problem: Three domains to choose from, in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, 741-741, 2014. doi>10.1145/2538862.2539031

[3] Andrea F. Lobo and Ganesh R. Baliga, Teaching algorithm design and intractability: A project-based curriculum focused on the traveling salesperson problem, *Journal of Computing Sciences in Colleges*, 29, (3), 7-8, January 2014.

[4] Lobo AF, Baliga GR. Assessment of a Project-Based Curriculum for Algorithm Design and Intractability Centered on the Traveling Salesperson Problem, *Journal of Computing Sciences in Colleges*, 31(3), 62-69, 2016.

[5] Andrea F. Lobo and Ganesh R. Baliga, Curriculum for Algorithm Design and Intractability, 2016, www.rowan.edu/~lobo/AlgosCurriculum, retrieved April 2, 2016.

[6] Takayuki Yato. Complexity and completeness of finding another solution and its application to puzzles, Masters thesis, University of Tokyo, 2003.

[7] Simulated Annealing, http://en.wikipedia.org/wiki/Simulated_annealing, accessed January 19, 2014.

[8] Andrea F. Lobo, Teaching Problem Reduction: NP-Completeness via Sudoku, in *Proc. XV Ibero American Congress in Computer Science Higher Education*, CIESC92.1-6, 2007.

[9] Sat4j the boolean satisfaction and optimization library in Java, http://www.sat4j.org/, accessed November 19, 2015.