

An Interactive Tutorial for NP-Completeness

Nabanita Maji

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science and Applications

Clifford A. Shaffer, Chair
Lenwood S. Heath
Chris North

May 26, 2015
Blacksburg, Virginia

Keywords: NP Completeness, Complexity Theory, Reductions, Algorithm Visualization, Computer
Science Education, Automated Assessment
Copyright 2015, Nabanita Maji

An Interactive Tutorial for NP-Completeness

Nabanita Maji

(ABSTRACT)

A Theory of Algorithms course is essential to any Computer Science curriculum at both the undergraduate and graduate levels. It is also considered to be difficult material to teach or to learn. In particular the topics of Computational Complexity Theory, reductions, and the NP-Complete class of problems are considered difficult by students.

Numerous algorithm visualizations (AVs) have been developed over the years to portray the dynamic nature of known algorithms commonly taught in undergraduate classes. However, to the best of our knowledge, the instructional material available for NP-Completeness is mostly static and textual, which does little to alleviate the complexity of the topic.

Our aim is to improve the pedagogy of NP-Completeness by providing intuitive, interactive, and easy-to-understand visualizations for standard NP Complete problems, reductions, and proofs. In this thesis, we present a set of visualizations that we developed using the OpenDSA framework for certain NP-Complete problems. Our paradigm is a three step process. We first use an AV to illustrate a particular NP-Complete problem. Then we present an exercise to provide a first-hand experience with attempting to solve a problem instance. Finally, we present a visualization of a reduction as a part of the proof for NP-Completeness.

Our work has been delivered as a collection of modules in OpenDSA, an interactive eTextbook system developed at Virginia Tech. The tutorial has been introduced as a teaching supplement in both a senior undergraduate and a graduate class. We present an analysis of the system use based on records of online interactions by students who used the tutorial. We also present results from a survey of the students.

This research was funded in part by NSF grants IIS-1258571 and DUE-1432008.

Acknowledgments

I would like to express my heartfelt gratitude to all those who have impacted my journey and enriched my experience at Virginia Tech.

First and foremost, many thanks to my advisor Dr. Cliff Shaffer for his invaluable guidance, support, time and, patience. He always found time for me, in spite of his busy schedule, for discussions, reviews and feedback. I am grateful for his efforts in arranging for my much-needed financial support. He has been a great source of encouragement, inspiration and, knowledge this past one year.

I would like to thank Dr. Lenwood Heath and Dr. Chris North for serving in my committee. I am grateful to Dr. Heath for his valuable feedback and suggestions at all stages of development of this tutorial. He provided us with the opportunity of introducing the tutorial in two of his classes without which the data collection for this thesis would not be possible.

I am thankful to my colleagues Sally, Eric and, Mohammed for their prompt help and assistance. My special thanks to Mohammed for his suggestions and technical guidance, particularly in analyzing the data for this thesis.

I will be forever indebted to my friends Rishi, Soham, Vishnu, Swathi, Clint, Cyril, Shreya, Suraj, Gargi, Sarthak, Rubasri, Sriram, Krish, Tania, Vivek, Chaitra, who have been there through thick and thin, who have always served as constant sources of inspiration, reassurances, love and, support. Suraj, Sarthak and, Gargi have lent me the comfort of a family during the final steps of my Masters. My experience would not have been so smooth and pleasant without any one of them.

My special thanks to Balaji for his endearing companionship, insightful suggestions, and his constant encouragement that helped me sail through my time in Virginia Tech.

Last but not the least, no words can express my gratitude for my parents Jharna and Lakshman Maji and my family back home, who have unconditionally loved and supported me at each and every step. It is their enthusiasm and passionate teachings that have paved the path for all my accomplishments in life.

Contents

1	Introduction	1
2	Background	3
2.1	NP-Completeness Terminology	3
2.2	Related Work	3
2.3	OpenDSA and Related Development Tools and Libraries	4
3	The NP-Complete Problems	6
3.1	Motivation	6
3.2	The Visualizations	7
3.2.1	Circuit Satisfiability (Circuit-SAT)	7
3.2.2	Formula Satisfiability (SAT)	9
3.2.3	3-CNF Satisfiability (3-SAT)	10
3.2.4	The Clique Problem	11
3.2.5	The Independent Set Problem	12
3.2.6	The Vertex Cover Problem	13
3.2.7	The Hamiltonian Cycle problem	14
3.2.8	The Traveling Salesman Problem	15
4	Exercises for NP-Complete Problem Instances	16
4.1	Motivation	16
4.2	Salient Features	17
4.2.1	Generating a Problem Instance	17

4.2.2	Level of Problem Difficulty	17
4.2.3	Solving the Problem Instance	18
4.2.4	Verification and Evaluation	18
4.2.5	Framework	18
4.3	The Exercises	19
4.3.1	3 CNF Satisfiability	19
4.3.2	Clique Problem	19
4.3.3	Independent Set Problem	20
4.3.4	Vertex Cover Problem	22
4.3.5	Hamiltonian Cycle Problem	23
4.3.6	Traveling Salesman Problem	25
5	Proofs of NP-Completeness	27
5.1	Reduction as a BlackBox	27
5.2	Proving NP Completeness using Reduction	28
5.3	The Reductions	30
5.3.1	Reducing Circuit-SAT to SAT	30
5.3.2	Reducing SAT to 3-SAT	31
5.3.3	Reducing 3-SAT to Clique	32
5.3.4	Reducing Clique to Independent Set	33
5.3.5	Reducing Independent Set to Vertex Cover	34
5.3.6	Reducing 3-SAT to Hamiltonian Cycle	35
5.3.7	Reducing the Hamiltonian Cycle Problem to Traveling Salesman	36
6	Evaluation of Usage Logs and Survey Feedback	38
6.1	Research Population	38
6.2	Evaluation of Usage Based on Event Logs	38
6.2.1	Analysis for CS 5114	39
6.2.2	Analysis for CS 4104	41
6.2.3	Analysis of Exercises	43

6.2.4	Usage of the tutorial for the finals	43
6.3	Feedback from students	44
6.3.1	CS 5114	44
6.3.2	CS 4104	45
7	Conclusion and Future Work	50
	Bibliography	51
A	Student Survey	53
B	Distribution in problem instances generated for exercises	55

List of Figures

1.1	Table of Contents for the tutorial.	2
3.1	Slides from the circuit-SAT problem visualization.	7
3.2	Slides from the SAT Problem Visualization.	9
3.3	Slides from the 3-SAT Problem visualization.	10
3.4	Slides from the the Clique Problem visualization.	11
3.5	Slides from the Independent Set Problem visualization.	12
3.6	Slides from the the Vertex Cover Problem visualization.	13
3.7	Slides from the the Hamiltonian Cycle Problem visualization.	14
3.8	Slides from the Traveling Salesman Problem visualization.	15
4.1	Exercise for 3-CNF Satisfiability.	20
4.2	Exercise for the Clique Problem.	21
4.3	Exercise for the Independent Set Problem.	22
4.4	Exercise for the Vertex Cover Problem.	23
4.5	Exercise for the Hamiltonian Cycle Problem.	24
4.6	Exercise for the Traveling Salesman problem.	25
5.1	BlackBox diagram for general process of reduction	28
5.2	Example of reductions.	29
5.3	Order of reduction for NP-Complete proofs	30
5.4	Screenshots of Circuit-SAT to SAT reduction visualization.	31
5.5	Screenshots of SAT to 3-SAT reduction visualization.	32
5.6	Screenshots of 3-SAT to Clique reduction visualization.	33

5.7	Screenshots of Clique to Independent Set reduction visualization.	34
5.8	Screenshots of Independent Set to Vertex Cover reduction visualization.	35
5.9	Screenshots of 3-SAT to Hamiltonian Cycle reduction visualization.	36
5.10	Screenshots of Hamiltonian Cycle to TSP reduction visualization.	37
6.1	Categorized usage in CS 5114 by the entire class.	41
6.2	Usage per module in CS 5114 by the entire class.	42
6.3	Usage of slideshows in CS 5114 for the entire class.	43
6.4	Categorized usage in CS 4104 for the entire class.	45
6.5	Usage per module in CS 4104 for the entire class.	46
6.6	Usage of slideshows in CS 4104 by the entire class.	47
6.7	Cumulative usage of exercises.	48
6.8	Usage of tutorial for CS 5114 over the semester.	48
6.9	Usage of tutorial for CS 4104 over the semester.	49

List of Tables

6.1	Curriculum for CS 5114.	40
6.2	Curriculum for CS 4104.	44
B.1	The distribution in problem instances generated for the exercise on Clique Problem.	55
B.2	The distribution in problem instances generated for the exercise on Vertex Cover Problem.	55
B.3	The distribution in problem instances generated for the exercise on Independent Set Problem.	55
B.4	The distribution in problem instances generated for the exercise on Traveling Salesman Problem.	56

Chapter 1

Introduction

According to Lewin, “There’s nothing so practical as good theory” [18]. A typical Computer Science Curriculum mandates a course on Algorithms and Complexity Theory. In the “Algorithms and Complexity” knowledge area of the ACM 2013 Computer Science Curriculum [9], NP-Completeness has been identified as a core topic. The importance of learning NP-Completeness is rooted in the fact that its concepts pervasively influence several disciplines like statistics, artificial life, automatic control, nuclear engineering, cryptography [19], neural networks [11], and many more. Unfortunately, NP-Completeness is also known to be one of the hardest topics to teach and learn. We conducted a survey among students (see Appendix A) where 78% of the survey-respondents stated NP-Completeness to be more difficult than other topics.

Continual efforts have been made to make the pedagogy of Computer Science more interactive and involving. With the ubiquity of the internet, increasing use of online media to provide adequate resources to replace or supplement learning in a traditional classroom comes as no surprise. A digital platform for delivery of information enables us to use new and interesting methods for presenting material to students. Visualization in general is one such technique. For our purpose of teaching Complexity Theory, Algorithm Visualization (AV) is a potentially effective method that can help explain otherwise difficult concepts with the help of graphics and interactivity. However, unfortunately as we show in our literature review in Chapter 2, few AVs for NP-Completeness have been developed.

This thesis presents our work on developing visualizations and exercises that we hope will provide a better understanding of the NP-Complete class of problems, the concept of reductions, and specific NP-Completeness proofs. The material has been introduced as a supplement to a graduate level and a senior undergraduate level course, and data has been collected about the students’ experience in using the material. Figure 1.1 shows a screenshot of the table of contents for the tutorial.

Major contributions of this Thesis include :

- Visualizations to introduce and define certain NP Complete problems
- Exercises to enable students to try solving problem instances from selected NP-Complete problems
- Visualizations to explain reduction of one NP Complete problem to another
- Data and analysis for the use of the material.


	CS5114 Spring 2015 TABLE OF CONTENTS	Report a bug Logout nabanita
Show Source About		Contents 0.1. Limits to Computing
Chapter 0 Limits to Computing		Search the book <input type="text"/> <input type="button" value="Go"/> <small>Enter search terms or a module, class or function name.</small>
<ul style="list-style-type: none"> 0.1. Limits to Computing 0.2. Reductions 0.2.1. Notes 0.3. NP-Completeness 0.3.1. Hard Problems 0.3.2. The Theory of NP-Completeness 0.3.3. Examples of NP-Complete Problems 0.4. Circuit Satisfiability 0.5. Formula Satisfiability 0.6. 3-CNF Satisfiability 0.7. The Clique Problem 0.8. The Independent Set Problem 0.9. The Vertex Cover Problem 0.10. The Hamiltonian Cycle Problem 0.11. The Traveling Salesman Problem 0.12. NP-Completeness Proofs 0.12.1. Examples of reductions for proof of NP Completeness 0.13. Reduction of Circuit SAT to SAT 0.14. Reduction of SAT to 3-SAT 0.15. Reduction of 3-SAT to Clique 0.16. Reduction of Clique to Independent Set 0.17. Reduction of Independent Set to Vertex Cover 0.18. Reduction of 3-SAT to Hamiltonian Cycle 0.19. Reduction of Hamiltonian Cycle to Traveling Salesman 0.20. Coping with NP-Complete Problems 		
Chapter 1 Appendix		
<ul style="list-style-type: none"> 1.1. Glossary 1.2. Bibliography Gradebook Index Search Page 		

Figure 1.1: Table of Contents for the tutorial.

This document is organized as follows. Chapter 2 consists of the literature review, the description of the tools and frameworks used (OpenDSA, JSAV, Khan Academy), and some background about NP Completeness. Chapter 3 discusses some NP Complete problems, and the visualizations developed to introduce these problems. Chapter 4 describes the exercises that let students explore some NP-Complete problems. Chapter 5 describes the modules on reduction and proofs of NP Complete problems, and the visualizations developed to explain the concepts. Chapter 6 presents the efforts made to gauge use of the material by students. Chapter 7 summarizes our contributions and provides recommendations for future work.

Chapter 2

Background

In this chapter we first provide a short explanation of the theory of NP-Completeness that we believe is necessary to understand the rest of the document. We review the literature on the use of visualizations in the pedagogy of NP-Complete problems. Finally, we talk about the OpenDSA Framework, JSAV library, and the exercise framework that our tutorial is based on.

2.1 NP-Completeness Terminology

In computational complexity theory, there exist various complexity classes of problems. In this context, a “class” is a collection of problems that are related in some way, typically in terms of the runtime cost required by any solution. Relevant to this document, it is helpful to understand the “NP-Complete” class. But before defining the “NP-Complete” class, it is required to define the “NP” and “NP-hard” classes. “Non deterministic polynomial” or “NP” problems can be solved in polynomial time on a non-deterministic parallel machine. A problem H is “NP-hard” if any problem in “NP” can be reduced to H in polynomial time. “NP-Complete” consists of problems which are both in “NP” and “NP-hard”. These are defined as decision problems, where each problem instance has a “yes” or a “no” solution. Some of the “NP-Complete” problems may also be expressed as an equivalent optimization problem. Problems considered in our tutorial are presented on both their decision and optimization variants, where applicable. In this document, we refer to problems as “hard” when no algorithm is known to exist that can solve the problem in polynomial time.

2.2 Related Work

The use of visualizations in the pedagogy of algorithms dates back to the 1980s. The earliest known effort is a video titled “Sorting out Sorting” by Robert Baecker in 1981 [8, 10]. The first system for implementing algorithm animation was named Balsa [13]. In the past 30 years, hundreds of visualizations have been developed (reported by Shaffer, et al [22]). However, the content of these AVs mostly focuses on introductory data structures and algorithms. Little has been documented about the use of AVs in the instructional material for computational complexity theory, which is an integral and difficult part of a course on algorithms.

Some early visualizations to teach complexity theory and NP-Completeness appeared in the late nineties. Pape [21] lays out ideas about visualizations that can help in teaching NP-Completeness. He presents the idea of developing exercises on reduction of one NP-Complete problem to another. He describes a Java-based exercise on the proof of NP Completeness for the Tiling problem. Pape [20] presents an automatic tutor for an exercise on reduction to prove MONOTONE-3SAT is NP-Complete. James Ten Eyck [16] describes an interactive Reduction Engine built in Java that can reduce one given NP Complete problem to another. Crescenzi [15] introduces an AV environment named AlViE [1] that provides animations for reduction from 3-SAT to 3-coloring, Hamiltonian path, Subset Sum, and Vertex Cover. These visualizations are delivered as Java applets and are not interactive in nature. Vegdahl, et al. [26] discusses the idea of teaching NP-Complete problems by mapping Circuit-SAT into the given problem using intuitive visualizations. But the visualizations provided are static images.

Few visualizations have previously been developed to teach NP-Completeness. These are mostly focused on the Reductions. Shaffer, et al. [7, 23] presents visualizations that allow students to try out problem instances of NP-Complete problems but limits it only to the Vertex Cover problem. Brandle, et al. [4, 12] present several Java-based AVs for NP-Completeness, where the user can specify the input for a particular problem instance or reduction and the corresponding output is generated by the AV.

2.3 OpenDSA and Related Development Tools and Libraries

OpenDSA [24] is an open-source system for developing active e-Textbooks for Data Structures and Algorithms courses. It consists of numerous interactive AVs and exercises presented with explanatory textual content. OpenDSA supports assessment of the the exercises attempted by the student to provide adequate feedback. Each HTML page is a separate module. A book instance is a collection of selected modules. Each module is configurable on its own, hence the content of a book instance can be tailored according to the requirements of a specific course or instructor. Our tutorial on NP Completeness was delivered as two separate book instances in OpenDSA, namely CS5114S15 (<http://algoviz.org/OpenDSA/Books/CS5114S15>) and CS4104S15 (<http://algoviz.org/OpenDSA/Books/CS4104S15>), to act as supplements for a graduate and undergraduate class, respectively.

The JavaScript Algorithm Visualization Library (JSAB) [2, 17] is a JavaScript library that provides easy to use programming interfaces to support development of visualizations for complex data structures and algorithms. For example, it provides APIs to display graphs, arrays, linked lists etc. It also supports various levels of user interactions on these. It also supports various graphical primitives, which can be useful in developing new wrapper functionality on top of JSAB without much effort. (We use this to develop functionality for circuit elements in the Circuit Satisfiability problem described in Section 3.2.1). The library provides APIs to develop a visualization in the form of a slideshow, which can be integrated with textual content and which the student can view at their own pace. JSAB also provides support for developing interactive exercises and their assessment. It enables the recording of every student interaction event for later analysis. The content developed using JSAB can be easily embedded in a HTML file. JSAB provides the backbone for the OpenDSA framework. All the dynamic content (AVs and exercises) in OpenDSA have been developed using JSAB.

The Khan Academy [5, 6, 25] provides open-source software for developing practice exercises. A whole range of exercises can be implemented using their software. In our case, we integrate JSAB with the Khan

Academy framework to develop exercises. This enables us to use the rich features of JSAV to visualize our problem instance on top of the Khan Academy control structure. The Khan Academy framework also facilitates providing hints that can be used to guide the student towards the correct solution. OpenDSA provides easy integration with the Khan Academy software, thus making Khan Academy an ideal choice for our set of practice exercises described in Chapter 4.

Chapter 3

The NP-Complete Problems

This chapter discusses the set of visualizations used in the tutorial for the purpose of introducing and explaining the problem statement for some standard NP-Complete problems. We start by explaining why these visualizations are necessary. Then we list the NP-Complete problems covered in the tutorial, and finally we describe the design and implementation of each visualization.

3.1 Motivation

The first and arguably most important part of a problem is understanding the problem statement. NP-Complete problem statements are often defined in mathematical terms using formalisations that take some time and effort to comprehend. We believe a visual illustration will reduce the cognitive load of understanding the problem statement. For example, a pictorial representation of Vertex Cover is much easier to understand than its textual formal definition. The satisfiability of a long boolean formula in conjunctive normal form (CNF) can be made visually obvious by using a truth table.

In most cases, before defining the problem statement, it is necessary to provide some context which may include explanations of concepts and processes that are dynamic in nature. For example, to understand the Circuit Satisfiability problem, it is necessary to understand the elements of a circuit and how signals propagate in a circuit. Our tutorial uses slideshows to provide step by step descriptions of dynamic processes.

The definition of a problem alone often does not suffice. The definition can be supported by relevant examples. For each problem, our tutorial graphically illustrates at least one example each for problem instances with an “yes” and a “no” answer. (Recall from Section 2.1, NP-Complete problems are expressed as decision problems with a “yes” or a “no” answer.) We discuss the visualizations in the next section individually in the context of each problem.

3.2 The Visualizations

3.2.1 Circuit Satisfiability (Circuit-SAT)

Statement for Decision Problem:

INSTANCE: A Boolean Circuit C .

QUESTION: Is C satisfiable ?

Description of the Visualization: Our visualization starts with a description of the circuit elements, the logic gates with their corresponding truth tables. The use of input signals and wires is described before laying out an example circuit. Figure 3.1 shows some slides from the visualization.

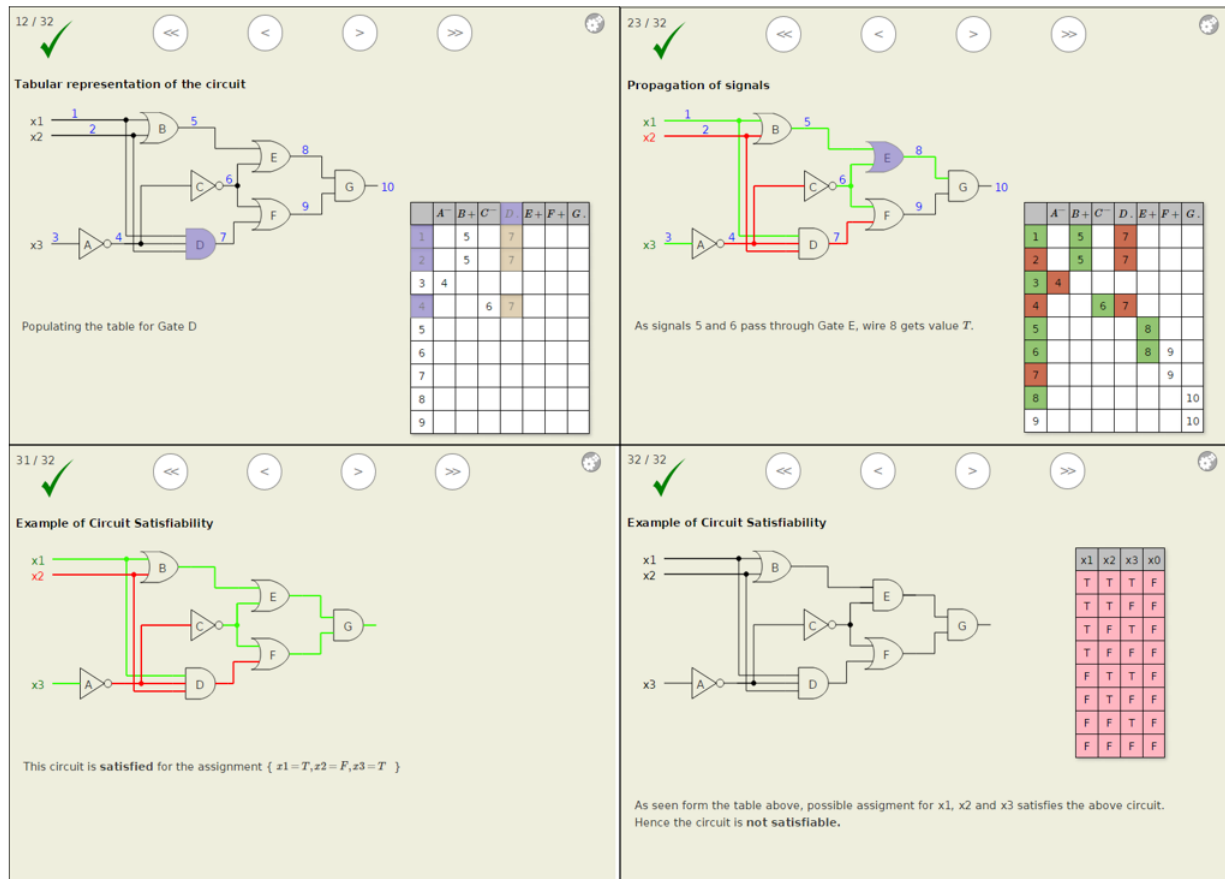


Figure 3.1: Slides from the circuit-SAT problem visualization.

The slideshow then explains a tabular representation for the circuit. The tabular representation is useful in explaining the propagation of signals. It is expected to prove useful if we incorporate a practice exercise for Circuit-SAT in future. Our visualization provides a detailed description of building the table from a circuit diagram. We highlight each gate and populate the cells in the table corresponding to that gate. Once the

table is populated, we use this representation to explain propagation of input signals through the circuit, corresponding to an example assignment of input signals. Again, each gate is considered one by one as all the input wires get assigned. The gate under consideration is highlighted in the circuit diagram and its output wire is colored (Green if True, Red if False) based on its inputs. The corresponding cell in the table is also colored accordingly.

The visualization then defines satisfiability of a circuit and provides an example circuit (using circuit diagrams), first for a satisfiable and then for a non-satisfiable circuit. For a satisfiable circuit, a satisfying assignment is provided as input, and the wires are colored (Green for True, Red for False) to visually depict that the output wire is indeed “True”. For the non-satisfiable circuit, a table is provided listing the output of the circuit for all possible input assignments, which clearly shows that the output is always “False”.

Drawing the circuit diagram: One of the major challenges in creating this visualization was that JSAV does not provide API support for drawing circuit elements like it does for graphs or linked lists. Hence we developed a wrapper class that uses graphical primitives of JSAV to draw logic gates and perform certain operations on them. An object to denote the circuit board can be created by calling the function `newCircuit()`. This class provides the following methods (All the APIs listed below do not return anything):

- *hide () / show ()* : Used to show or hide a circuit diagram.
- *addGate (av, GateType, x, y, r)* : Adds a new gate to the circuit diagram. The JSAV object corresponding to the visualization containing the circuit is passed as parameter ‘av’. The type, position and size of the gates is passed as input parameters ‘GateType’, ‘x’, ‘y’ co-ordinates and radius ‘r’.
- *getGates ()* : Returns an array of gates that are a part of the given circuit diagram.
- *clearGate (gate)* : Removes the specified gate from the circuit diagram.
- *connectGates (av, gate1, gate2, ip, brkpoints)* : Connects the output pin of ‘gate1’ to the input pin ‘ip’ of ‘gate2’. A gate is designed to have 3 input pins. The wire can bend at specified breakpoints (passed as an array of points ‘brkpoints’), to give a neat look to the circuit diagram.
- *inputToGate (av, sname, startpt, gate, ip, brkpoints)* : Connects an input signal ‘sname’ to the input pin ‘ip’ of ‘gate’. The wire corresponding to the input signal is drawn from position ‘startpt’ and bends at points specified in ‘brkpoints’.
- *changeGate (av, gate, type)* : Replaces ‘gate’ by a new gate of type ‘type’ in the diagram.
- *addSignals (arr)* : Adds input signals passed as an array ‘arr’ to the circuit.
- *getSignals ()* : Returns an array of input signals used in the circuit.
- *assignSignal (sname, bool)* : Assigns the truth value ‘bool’ to the signal ‘sname’.
- *unassignSignal (sname)* : Unassigns the signal ‘sname’.
- *assignOP (gate, bool)* : Assigns the truth value ‘bool’ to the output pin of ‘gate’.
- *unassignOP (gate)* : Unassigns the output pin of ‘gate’.
- *assignIP (gate, ip, bool)* : Assigns the truth value ‘bool’ to the input pin ‘ip’ of ‘gate’.

- *unassignIP (gate, ip)*: Unassigns the input pin 'ip' of 'gate'.

This wrapper class has also been used in the visualization to explain the reduction of Circuit-SAT to SAT (discussed in Chapter 5), and can be reused in future for other visualizations using circuits.

3.2.2 Formula Satisfiability (SAT)

Statement for Decision Problem:

INSTANCE: A boolean formula ϕ in conjunctive normal form.

QUESTION: Is ϕ satisfiable ?

4 / 21

Background

Boolean variables are variables that can have a value from {T,F}, where 'T' stands for TRUE and 'F' for FALSE. For example : x_1, x_2, x_3

Boolean operators are AND (+), OR (.), NOT (−) which follow the truth table

The NOT Operator

x	\bar{x}
T	F
F	T

The AND Operator

x	y	$x.y$
T	T	T
T	F	F
F	T	F
F	F	F

The OR Operator

x	y	$x+y$
T	T	T
T	F	T
F	T	T
F	F	F

A Boolean formula is composed of boolean variables and operators. For example $x_1 + x_2.x_3$

11 / 21

Background

An assignment to the boolean variables in a formula is known as a **truth assignment**.

A truth assignment of variables is said to be **satisfying**, if it causes the formula to evaluate to "TRUE".

A CNF is said to be **satisfiable** if it has a satisfying assignment.

For example $(x_1 + x_2.x_3)$ is "True" for $x_1=T, x_2=T, x_3=T$, hence satisfiable.
 $(x_1.\bar{x}_1)$ is always "False", hence not satisfiable.

16 / 21

Example of SAT

$P = (x_1 + x_2).(x_2 + \bar{x}_3 + x_4).(x_1 + \bar{x}_2 + x_3 + x_4).(\bar{x}_1 + x_3)$

Truth Table for P

x_1	x_2	x_3	x_4	P
F	F	F	F	F
F	F	F	T	F
F	F	T	F	F
F	F	T	T	F
F	T	F	F	F
F	T	F	T	F
F	T	T	F	T
F	T	T	T	T
T	F	F	F	F
T	F	F	T	F
T	F	T	F	F
T	F	T	T	T
T	T	F	F	F
T	T	F	T	F
T	T	T	F	T
T	T	T	T	T

There exists assignments that makes the formula true (The green rows)
P is satisfiable

20 / 21

Example of SAT

$P = (x_1 + x_2).(x_2 + \bar{x}_3 + x_4).(x_3 + \bar{x}_4).(x_1 + \bar{x}_1).(x_1 + \bar{x}_2 + x_3 + x_4)$

Truth Table for P

x_1	x_2	x_3	x_4	P
F	F	F	F	F
F	F	F	T	F
F	F	T	F	F
F	F	T	T	F
F	T	F	F	F
F	T	F	T	F
F	T	T	F	F
F	T	T	T	F
T	F	F	F	F
T	F	F	T	F
T	F	T	F	F
T	F	T	T	F
T	T	F	F	F
T	T	F	T	F
T	T	T	F	F
T	T	T	T	F

There does not exist any assignment that makes the formula true (No green rows)
P is non satisfiable

Figure 3.2: Slides from the SAT Problem Visualization.

Description of the Visualization: The slideshow starts by defining boolean operators (AND, OR, and NOT) and providing a truth table for each operator. It provides definitions of literals, clauses, conjunctive normal forms, and boolean formula, supporting each definition with examples. After providing the necessary context, the meaning of satisfiability for a boolean formula is explained. Two examples are provided. The first example portrays a satisfiable boolean formula in CNF and its corresponding truth table. The rows in the truth table that yields a value "True" for the formula are marked in green, and the rows yielding a "False" are marked in red. The presence of green rows in the truth table illustrate the fact that the example formula

is satisfiable. The second example is of a non-satisfiable formula and its truth table. All the rows of the truth table being red demonstrates that the formula is non-satisfiable. Figure 3.2 shows some slides from the visualization.

3.2.3 3-CNF Satisfiability (3-SAT)

Statement for Decision Problem:

INSTANCE: A boolean formula ϕ in conjunctive normal form with each clause containing exactly three literals.

QUESTION: Is ϕ satisfiable ?

6 / 18

✓

<<

<

>

>>

⚙

Background

A 3-CNF is a Boolean formula that is an AND of clauses, each of which is an OR of exactly 3 distinct literals.

Example of 3-CNF: $(x_1 + x_2 + x_3), (\bar{x}_1 + x_4 + x_6), (x_2 + \bar{x}_5 + \bar{x}_3), (x_1 + \bar{x}_3 + \bar{x}_6)$.

An assignment to the boolean variables in a formula is known as a **truth assignment**.

A truth assignment of variables is said to be **satisfying**, if it causes the formula to evaluate to "TRUE".

A 3-CNF is said to be **satisfiable** if it has a satisfying assignment.

12 / 18

✓

<<

<

>

>>

⚙

Example of 3-SAT

$P = (x_1 + x_2 + x_3), (x_4 + \bar{x}_2 + \bar{x}_1), (x_3 + \bar{x}_2 + x_1), (\bar{x}_2 + x_4 + x_1), (\bar{x}_4 + \bar{x}_2 + \bar{x}_1), (x_1 + x_4 + x_3), (x_3 + x_2 + x_4)$

Truth Table for P

x_1	x_2	x_3	x_4	P
F	F	F	F	F
F	F	F	T	F
F	F	T	F	F
F	F	T	T	F
F	T	F	F	F
F	T	F	T	F
F	T	T	F	F
F	T	T	T	F
T	F	F	F	T
T	F	F	T	T
T	F	T	F	T
T	F	T	T	T
T	T	F	F	T
T	T	F	T	F
T	T	T	F	F
T	T	T	T	F

There exists assignments that makes the formula true (The green rows)

P is satisfiable

17 / 18

✓

<<

<

>

>>

⚙

Example of 3 - SAT

$P = (x_1 + x_2 + x_3), (x_1 + \bar{x}_3 + \bar{x}_4), (\bar{x}_1 + x_3 + x_4), (\bar{x}_2 + x_4 + \bar{x}_3), (x_1 + \bar{x}_2 + x_3), (x_2 + \bar{x}_3 + x_4), (\bar{x}_1 + \bar{x}_2 + \bar{x}_4), (\bar{x}_1 + x_3 + x_4)$

Truth Table for P

x_1	x_2	x_3	x_4	P
F	F	F	F	F
F	F	F	T	F
F	F	T	F	F
F	F	T	T	F
F	T	F	F	F
F	T	F	T	F
F	T	T	F	F
F	T	T	T	F
T	F	F	F	F
T	F	F	T	F
T	F	T	F	F
T	F	T	T	F
T	T	F	F	F
T	T	F	T	F
T	T	T	F	F
T	T	T	T	F

There exists no assignments that makes the formula true (No green rows)

P is not satisfiable

18 / 18

✓

<<

<

>

>>

⚙

Insights

Size of the truth table is 2^n where n is the number of boolean variables involved

Hence the problem gets exponentially harder as number of variables increase.

Figure 3.3: Slides from the 3-SAT Problem visualization.

Description of the Visualization: This is a short visualization similar to the formula satisfiability slideshow. First, the definition of 3-CNF (conjunctive normal form with exactly three literals in each clause) is provided. The meaning of satisfiability for such a formula is explained and the 3-SAT problem is stated. Two examples are provided, one for a satisfiable formula and other for a non-satisfiable one. The rows in the truth table that yield "True" are marked in green, and the rows yielding "False" are marked in red. The presence of green rows in the truth table corresponding to a satisfiable boolean formula in 3-CNF prove that the formula

is satisfiable. The example of a non-satisfiable formula has a truth table with all rows in red, showing that the formula is not satisfiable. Figure 3.3 shows some slides from the visualization.

3.2.4 The Clique Problem

Statement for Decision Problem:

INSTANCE: A graph G and an integer bound k .

QUESTION: Does G contain a clique of size $\geq k$?

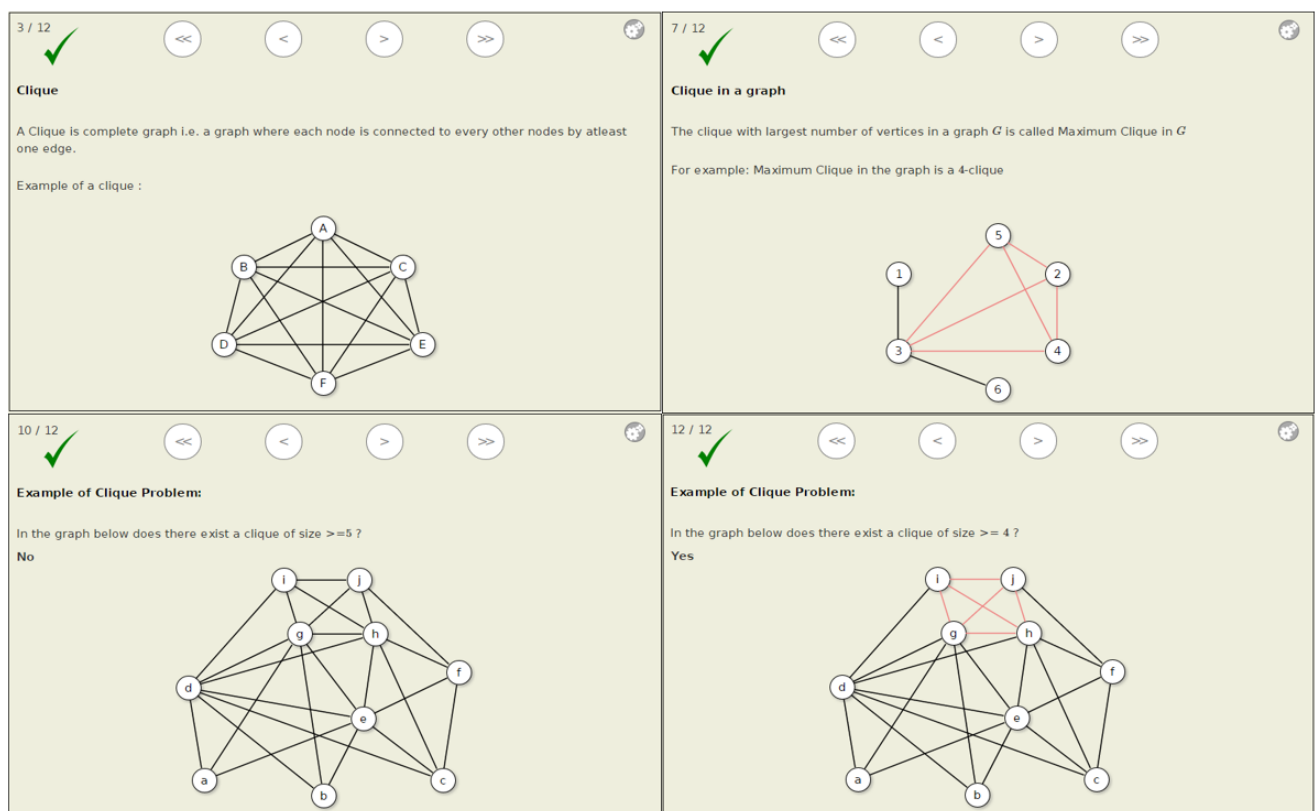


Figure 3.4: Slides from the the Clique Problem visualization.

Description of the Visualization: This visualization starts by defining a clique, with the example of a complete graph. The following slides show examples of finding subgraphs in a graph that form cliques of different sizes. The definition of the clique problem can also be expressed as an optimization problem to find the largest clique in the graph. The slideshow presents the clique problem in both its decision problem and optimization problem forms. It then shows two examples. The first example is of a graph where a clique of a given size does not exist. The second example is a graph where a clique of the given size exists. The subgraph forming the clique is highlighted using a different color. Figure 3.4 shows some slides from the visualization.

3.2.5 The Independent Set Problem

Statement for Decision Problem:

INSTANCE: A graph G and an integer bound k .

QUESTION: Does G contain an independent set of size $\geq k$?

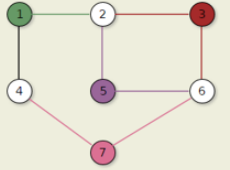
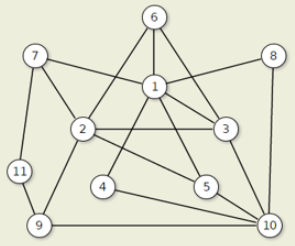
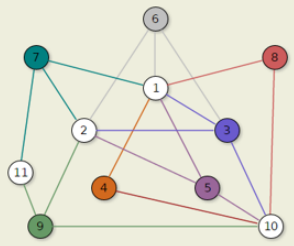
<p>5 / 12 ✓</p> <p>Independent Set</p> <p>An Independent Set of a graph is a set of vertices such that no two of them are connected i.e. there exists no edge between any two vertices of an Independent Set.</p> <p>The largest possible Independent Set of a graph is called the "Maximum Independent Set".</p>  <p>The colored vertices in this graph form an independent set. The Independent set is {1,3,5,7}</p>	<p>8 / 12 ✓</p> <p>The Independent Set Problem</p> <p>The Independent Set Problem can be defined as either of the following:</p> <p>Given a graph $G = (V, E)$, find the Maximum Independent Set in G.</p> <p>Or</p> <p>Given a graph $G = (V, E)$, and a number k, does G contain an Independent Set of size $\geq k$?</p>
<p>10 / 12 ✓</p> <p>Example of Independent Set Problem:</p> <p>Does the graph below have an independent set of size ≥ 9 ?</p> <p>No</p> 	<p>12 / 12 ✓</p> <p>Example of Independent Set Problem:</p> <p>Does the graph below have an independent set of size ≥ 7 ?</p> <p>Yes</p> 

Figure 3.5: Slides from the Independent Set Problem visualization.

Description of the Visualization: The visualization starts with a definition of an independent set and a maximum independent set. The definition is supported by an example with a graph, where the vertices forming the independent set are highlighted using colors. In this visualization, the edges incident on a vertex that is a part of the independent set are highlighted with the same color as the vertex itself. The intent is to make the association of edge and vertex visually obvious so as to portray that the colored vertices do not share an edge with any other colored vertex, and hence form an independent set. Examples are provided in the visualization showing subgraphs in a graph containing independent sets of varying sizes using the same coloring scheme. The independent set problem can also be defined as an optimization problem of finding the largest independent set in the graph. The visualization defines both the decision problem and its optimization variant, and follows it up with two examples. The first shows a graph that does not contain an independent set, that is larger than or equal to a given size. The second example shows a graph containing an independent set of the given size, highlighted using the aforementioned coloring scheme. Figure 3.5 shows

some slides from the visualization.

3.2.6 The Vertex Cover Problem

Statement for Decision Problem:

INSTANCE: A graph G and an integer bound k .

QUESTION: Does G contain a vertex cover of size $\leq k$?

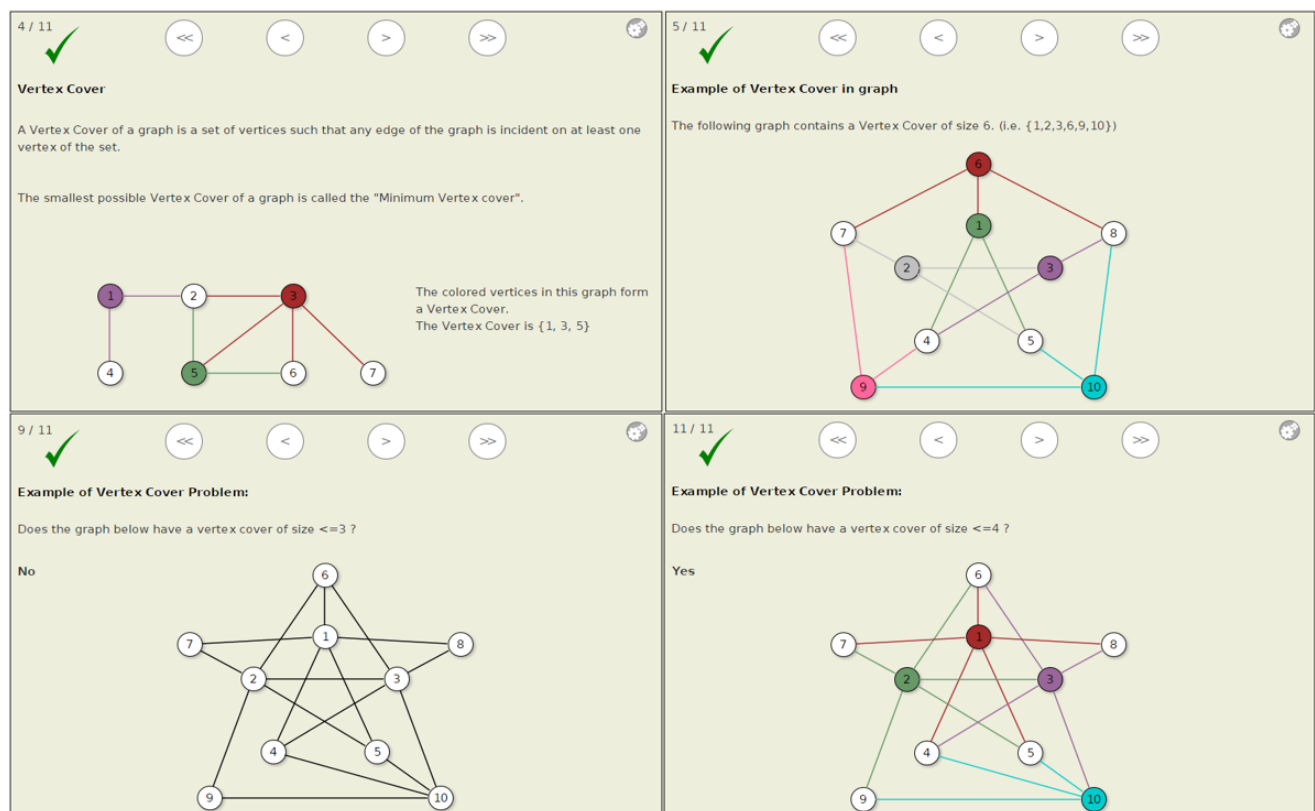


Figure 3.6: Slides from the the Vertex Cover Problem visualization.

Description of the Visualization: This visualization uses the same coloring scheme mentioned earlier for the independent set problem, that is, using the same colors for a vertex and the edges incident on the vertex to show their association. In this case, the goal is to portray that all the edges are colored, implying that they are all incident on one of the colored vertices and the colored vertices indeed form a vertex cover. The visualization starts by defining a vertex cover and portraying examples of graphs, with their vertex covers portrayed using colored vertices. There are a few examples showing graphs containing vertex covers of various sizes. The optimization variant of this problem is to find the smallest vertex cover in the graph. The slideshow defines both the decision problem and its optimization variant. It then illustrates the problem using two examples. The first example is of a graph that does not contain a vertex cover of the specified size. The second example is of a graph that yields a “yes” answer to the Vertex Cover problem, the vertex

cover being highlighted using the color scheme mentioned before. Figure 3.6 shows some slides from the visualization.

3.2.7 The Hamiltonian Cycle problem

Statement for Decision Problem:

INSTANCE: A graph G

QUESTION: Does G contain a Hamiltonian Cycle ?

Figure 3.7: Slides from the the Hamiltonian Cycle Problem visualization.

Description of the Visualization: This is a short visualization that starts by defining a Hamiltonian Cycle. It then shows an example of an undirected graph containing a Hamiltonian Cycle. The edges that form the Hamiltonian Cycle are highlighted using a different color. The Hamiltonian Cycle problem is then defined. Two examples are shown to help explain the problem. The first example shows a directed graph that does contain a Hamiltonian Cycle, which is highlighted in the graph. The second example shows a directed graph where no Hamiltonian Cycle exists. Figure 3.7 shows some slides from the visualization.

3.2.8 The Traveling Salesman Problem

Statement for Decision Problem:

INSTANCE: A weighted, complete graph G (with the weight of an edge denoting its length) and an integer bound k

QUESTION: Does G contain a simple cycle that includes all vertices and has total length $\leq k$?

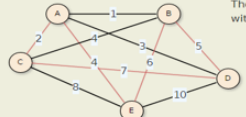
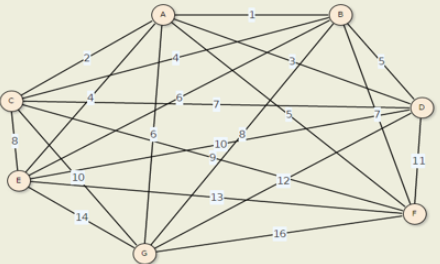
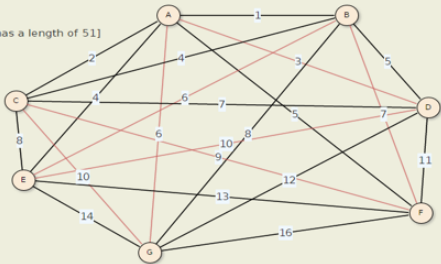
<p>5 / 10 ✓</p> <p>Traveling Salesman</p> <p>There are n cities. Every pair of cities is separated by some distance. A traveling salesman aims to visit them all in a way that no city is visited more than once and the total distance covered during the tour is as short as possible.</p> <p>This can be modelled as a complete graph where each node represents a particular city and the weight of the edges denote the distance between the two cities it connects. The problem now can be stated as finding the shortest simple cycle in the graph that passes through all vertices in the graph. (The length of a cycle being the sum of weights of all the edges included in the cycle)</p> <p>Note: A simple cycle may be defined as a closed walk with no repetitions of vertices and edges allowed, other than the repetition of the starting and ending vertex.</p> <p>An example :</p>  <p>The red edges form a minimum-length tour with total length being 24</p>	<p>6 / 10 ✓</p> <p>The Traveling Salesman Problem</p> <p>The Traveling Salesman problem can be defined either as a decision problem or not. The decision form is known to be NP-complete.</p> <p>Given a graph $G = (V, E)$, find the shortest simple cycle that passes through all vertices of the graph. The length of the cycle is the sum of weights of its edges.</p> <p>OR</p> <p>(Decision Problem Form) Given a graph $G = (V, E)$ and integer k, does the graph contain a simple cycle that passes through all vertices and has length $\leq k$?</p>
<p>8 / 10 ✓</p> <p>Example of Traveling Salesman Problem (decision form)</p> <p>Does this graph have a simple cycle that includes all vertices and has length ≤ 50?</p> <p>No</p> 	<p>10 / 10 ✓</p> <p>Example of Traveling Salesman Problem</p> <p>Does this graph have a simple cycle that includes all vertices and has length ≤ 557?</p> <p>Yes [The cycle has a length of 511]</p> 

Figure 3.8: Slides from the Traveling Salesman Problem visualization.

Description of the Visualization: The Traveling Salesman problem uses a weighted, complete graph and an integer as its input instance. It can also be expressed as an optimization problem to find the shortest simple cycle that includes all the vertices of the graph, where the weight of an edge denotes its distance. This visualization starts by explaining the Traveling Salesman problem. It does so using a weighted complete graph of 5 nodes and highlights the shortest simple cycle covering all vertices, in red. It then formally states the Traveling Salesman problem, both the decision problem and its optimization variant. Then come two examples. In the first example, a graph is shown that does not contain a simple cycle covering all vertices with length less than or equal to the given length. The second example shows a graph that has a “yes” answer to the Traveling Salesman Problem within the given integer bound. The cycle is highlighted in red and its length is mentioned, to convincingly show that this problem instance indeed yields a “yes” answer. Figure 3.8 shows some slides from the visualization.

Chapter 4

Exercises for NP-Complete Problem Instances

This chapter presents the interactive exercises that were created for the NP-completeness tutorial. We start our discussion by explaining the motivation behind these exercises. We then explain some problems that we had to overcome and the guiding philosophy that influenced the design decisions when developing the exercise. We conclude with the design and implementation for each exercise.

4.1 Motivation

While developing the visualizations to introduce and define a particular NP Complete problem, we were faced with an impediment. Recall that an NP Complete problem can be defined as a decision problem with a “Yes ” or a “No” answer. It is fairly simple to graphically illustrate a problem instance with a “Yes” answer. For example, giving a satisfying assignment for a “satisfiable” boolean formula convinces the learner that the formula is indeed satisfiable. A highlighted k-clique or Hamiltonian Cycle in a graph makes the presence of a k-clique or Hamiltonian Cycle self evident. However, that is not the case for an example with a “No” answer. When we present a formula that is NOT satisfiable or a graph that DOES NOT contain a k-clique, it is not visually obvious to the learners that the graph does have this property. In fact this is a hallmark of NP-Complete problems.

There is another important aspect in which textbooks have always been rather authoritative. While talking about NP-Complete problems, they often just state it for a fact that the problem is “hard”, which may not seem convincing to a student right away. Some NP-Complete problems share a close resemblance to problems that can be solved in polynomial time. For example, 3-CNF satisfiability is *hard* whereas 2-CNF satisfiability¹ is not. Checking whether a graph has a Euler tour² can be done in polynomial time, whereas checking the presence of a Hamiltonian Cycle is a NP-Complete problem. This difference in behavior for seemingly similar problems often makes it counter-intuitive for a student to recognize a NP-Complete problem as *hard*.

¹2-CNF is a boolean formula in conjunctive normal form where each disjunction contains at most 2 variables.

²An Euler tour is a closed walk that uses every edge of a graph exactly once.

The Constructivist Theory of education [27] argues that humans learn more from the interaction between their experiences and their ideas. In our case we realized that both of our above-mentioned shortcomings can be remedied by hands-on experience in attempting the problem. If we provide students with an instance of a problem, in the attempt to find a solution, they should begin to appreciate the complexity of the problem and internalize the fact that the problem is indeed hard. For problem instances with a “No” answer (for example, if provided with a *Non-Satisfiable* boolean formula), the student can keep trying to find a satisfying assignment until they are convinced that none exists. A conclusion derived from their own experience is always more convincing for them than stated facts.

We emphasize that we are discussing practice exercises on instances of NP-Complete problems. So far we are not dealing with proofs of NP-Completeness. Visualizations of such proofs are covered in the next chapter.

4.2 Salient Features

Before starting to design and implement an exercise, there were a few important aspects that we had to consider. This section lists and elaborates on those aspects.

4.2.1 Generating a Problem Instance

The first requirement in an exercise is the problem to be solved. Problem instances can either be selected from a bank of pre-compiled questions, or can be generated dynamically at runtime. We use the latter approach. Although it is tougher to implement, generating problems dynamically reduces the chance that the same problem instance gets repeated for a student. It also increases the number and type of instances that are covered by our exercises. Our first thought was to use a randomized graph generation algorithm for all exercises corresponding to graph-related problems. We planned to use a separate algorithm for generating a random instance of 3-CNF formulas. The use of randomized algorithms ensured that a different instance is generated almost every time that a student attempts the exercise. However our initial attempt restricted us from having control on the kind of the graph that was generated. For example, for an exercise on the Hamiltonian Cycle problem, it is desired that the generated graph does or does not include a Hamiltonian Cycle with equal likelihood. That is, the probability of a Hamiltonian Cycle being present whenever a new problem instance is generated should be close to 0.5. However, when we tested our randomized graph generation algorithm, this property was not ensured. Hence, we had to modify our algorithm to suit the requirement of each exercise individually, thus giving us a finer level of control. In all the exercises listed in Section 4.3, the pseudo-random algorithms used to generate problem instances are described for each individual exercise.

4.2.2 Level of Problem Difficulty

The second important question that we needed to address was how hard should the problem be. Since our objective behind developing the exercises is to help the students internalize the fact that NP-Complete problems are indeed *hard*, the exercises should not be too simple. In other words, the problem instance

should not be so easy that the answer becomes obvious. However, the exercises should not be so difficult that the student gets bored or discouraged and gives up. The level of difficulty should be optimal to ensure that it does require the right amount of effort to get the answer.

That raises the question of how to control the level of difficulty for a particular exercise. In our case, the level of difficulty mostly depends on the problem size. But there are other factors specific to the particular problem being addressed. For example, the number of vertices and the number of edges for a graph affects the difficulty level of a Clique Problem. The distribution of weights in the weighted graph determines how difficult an instance of the Traveling Salesman Problem is. In Section 4.3, we provide details of these factors and their reflection in our implementation, in the context of each individual exercise.

Our tuning of factors affecting the level of difficulty is purely based on the developer's usability testing. However, it is fairly simple to adjust the difficulty in future based on continued feedback from students and instructors.

4.2.3 Solving the Problem Instance

The third aspect of an exercise is to generate the correct answer for the problem instance to be able to verify the student's response as well as to show them the solution in case of failure. Since our problem instances are generated at runtime, there needs to be an algorithm that can be triggered at runtime to calculate the answer. The algorithms used for finding a solution by definition are of non-polynomial complexity. However this does not pose a concern for our system because the size of our problem instances are small.

4.2.4 Verification and Evaluation

The fourth aspect of an exercise is validating a student's response. A student can be evaluated at every step of the exercise or on the final response. The nature of NP-Complete problems restricts us to the latter. In order to encourage the students to attempt a problem in whatever way they want, we refrain from alerting them on any wrong intermediate move. Only the submitted response is verified, and the student is informed about the correctness of the response. The grades/penalties assigned for the exercises and the number of allowable attempts can be adjusted according to an instructor's requirement.

4.2.5 Framework

We develop our exercises (as opposed to the other types of proficiency exercise implementations used in OpenDSA) using the JSAV library and the Khan Academy framework. The choice of the Khan Academy framework was guided by our requirement to evaluate only the final response from a student (and not every step). The implementation details are provided in the next section.

4.3 The Exercises

This section describes the collection of exercises that we developed. In the subsequent subsections, we elaborate on the implementation details for each exercise.

4.3.1 3 CNF Satisfiability

Problem definition:

INSTANCE: A boolean formula ϕ in 3-CNF.

SOLUTION: A satisfying assignment if ϕ is satisfiable, “Not Satisfiable” otherwise.

Student’s response: Either provide a satisfying assignment for the formula or claim that the formula is Non-Satisfiable.

Layout Design and User Interactions: Figure 4.1 portrays a screenshot for the exercise. The layout consists of an array of buttons, where each button represents a boolean variable. A variable is in unassigned state to start with. When an unassigned variable is clicked, it gets assigned to *True* (the button turns green). If we click on a variable that’s already *True*, it gets re-assigned to *False* (the button turns red). If we click on a variable that’s already *False*, the variable gets unassigned. When a particular variable is clicked, the clauses in the 3-CNF formula containing that variable are evaluated. At every step, with the current assignment, if any clause evaluates to *True*, the clause shows up as green on the screen. Similarly, if any clause evaluates to *False* with the current assignment, it shows up as red. The “Reset” button can be clicked at any point, to return to the initial state. The checkbox for “Non-satisfiable” can be checked to claim that the given formula is not satisfiable. Clicking the “Check Answer” button submits the answer for validation. If the answer is correct, the student can move on to repeat the exercise with a different problem instance. If the answer is wrong, the student can retry.

Generation of problem instance: As a general observation about the 3-SAT problem, it is well known that when a formula contains many variables and few clauses, or few variables and many clauses, it becomes easy to determine its satisfiability. Hence an optimal balance between the number of variables and clauses is desirable for our purposes. For our case, we vary the number of variables from 4 to 6 and number of clauses from 8 to 12. This is easily adjustable. The number of clauses and literals are randomly picked from the given range. For each clause, three literals are randomly picked from the set of given variables and their compliments. Finally, a problem instance is generated using these clauses.

4.3.2 Clique Problem

Problem definition:

INSTANCE: A graph G .

SOLUTION: The largest clique in G .

Student’s response: The student needs to provide a set of vertices that forms the largest clique in the given graph.

3-sat problem.
Current score: 1 out of 1

Your task in this exercise is to find out an assignment of the boolean variables that makes the expression true or declare that no such assignment exists.

Assign the variables appropriately to satisfy the expression.

Click on the variable once to assign it true , twice to assign it false and thrice to reset.

Reset
☐ Not Satisfiable

x_1

x_2

x_3

x_4

x_5

x_6

$(\overline{x_3} + \overline{x_2} + x_6).$
 $(x_2 + x_5 + x_4).$
 $(x_4 + x_3 + x_6).$
 $(\overline{x_5} + x_1 + \overline{x_4}).$
 $(x_4 + x_5 + x_3).$
 $(\overline{x_4} + x_2 + \overline{x_1}).$
 $(x_1 + \overline{x_2} + x_6).$
 $(\overline{x_1} + x_5 + x_2).$
 $(\overline{x_4} + x_1 + \overline{x_5}).$
 $(x_3 + \overline{x_1} + x_4).$
 $(x_5 + \overline{x_1} + \overline{x_3}).$

Answer

Check Answer

Need help?

I'd like a hint

Figure 4.1: Exercise for 3-CNF Satisfiability.

Layout Design and User Interactions: Figure 4.2 portrays a screenshot of the exercise. Clicking on a vertex toggles it's state between selected and unselected. If two vertices connected by an edge in the given graph is selected, the connecting edge gets highlighted in red to assist the student in judging the connections. The “Reset” button can be clicked at any point, to clear all selections and return to the initial state. Clicking the “Check Answer” button submits the answer for validation. Validation is done by determining whether the selected vertices form a clique in the graph and whether the size of this clique is equal to the largest clique. The largest clique in the graph is pre-calculated using a non-polynomial algorithm. If the answer passes the validation, the student can move on to repeat the exercise with a different graph. If the answer is wrong, the student can retry.

Generation of problem instance: For a clique problem, the level of difficulty depends on the size and density of the graph. It is relatively easier to spot a clique in a sparse graph than on a dense one. For our problem instances, we first randomly chose the number of vertices from the range [10,12]. If the number of vertices is n , the number of edges vary from $2n + 3$ to $3n$. With the number of vertices and edges decided, our algorithm generates a random adjacency matrix which we use to generate a problem instance. The number of vertices and edges for the problem instance are adjustable parameters. Our tuning of these parameters are solely based on our own usability testing.

4.3.3 Independent Set Problem

Problem definition:

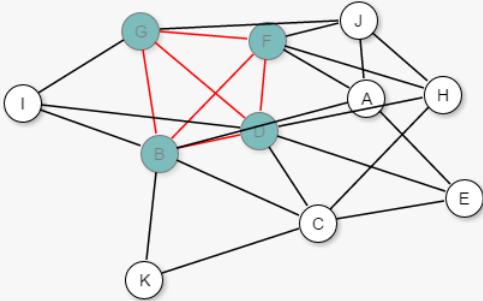
INSTANCE: A graph G .

Maximum Clique Problem

Current score: 1 out of 1

Your task in this exercise is to find the largest clique.

Select the vertices that can form the largest clique.



Answer

😊 Correct! Next Question...

Figure 4.2: Exercise for the Clique Problem.

SOLUTION: The largest Independent Set in G .

Student's response: The student needs to provide a set of vertices that forms the largest independent set in G .

Layout Design and User Interactions: Figure 4.3 portrays a screenshot of the exercise. Clicking on a vertex toggles its state between selected and unselected. The “Reset” button can be clicked at any point, to clear all selections and return to the initial state. Clicking the “Check Answer” button submits the answer for validation. Validation is done by determining whether the selected vertices form a independent set for the graph G and whether the size of this Independent Set is equal to the largest Independent Set. The largest Independent Set in the graph is calculated beforehand by using a non-polynomial algorithm. If the answer is validated, the student can try the same exercise with a different graph. If the answer is wrong, the student can retry.

Generating a problem instance: For the Independent Set problem, the level of difficulty depends on the size and density of the graph. It is relatively simple to find an independent set on a sparse or dense graph, compared to a graph of moderate density. For our problem instances, we first set the number of vertices to a random number from the range $[10,12]$. If the number of vertices is n , the number of edges vary from $n + 3$ to $2n$. With the number of vertices and edges decided, our algorithm randomly populates the adjacency matrix maintaining the constraint on the number of vertices and edges. This adjacency matrix is used to generate the graph that acts as our problem instance.

Maximum Independent Set Problem
Current score: 1 out of 1

Your task in this exercise is to find the maximum independent set.

Select the vertices that can form a maximum independent set.

Answer

😊
Correct! Next Question...

[Show hints](#)

Figure 4.3: Exercise for the Independent Set Problem.

4.3.4 Vertex Cover Problem

Problem definition:

INSTANCE: A graph G .

SOLUTION: The smallest Vertex Cover in G .

Student's response: A student needs to provide a set of vertices which forms the smallest vertex cover in G .

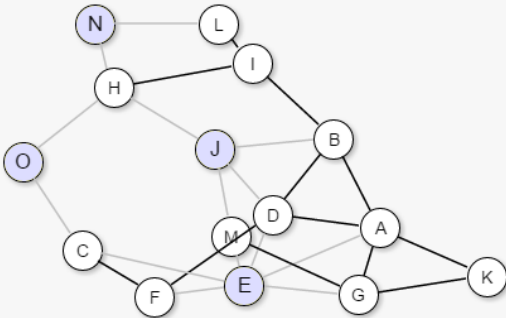
Layout Design and User Interactions: Figure 4.4 portrays a screenshot of the exercise. Clicking on a vertex toggles its state between selected and unselected. Once a vertex is selected, the edges adjacent to the vertex turn gray. This helps the students to see which edges are covered by their selection. The “Reset” button can be clicked at any point, to clear all selections and return to the initial state. Clicking the “Check Answer” button submits the answer for validation. Validation is done by determining whether the selected vertices form a vertex cover for the graph and whether the size of this vertex cover is equal to the largest vertex cover. The largest vertex cover in the graph is calculated beforehand by using a non-polynomial algorithm. If the answer is validated, the student can move on and repeat the exercise with a different problem instance. If the answer is wrong, the student can retry.

Minimum Vertex Cover Problem

Current score: 0 out of 1

Your task in this exercise is to find the minimum vertex cover.

Select the vertices that can form a minimum vertex cover.



Answer

Check Answer

Need help?

I'd like a hint

Figure 4.4: Exercise for the Vertex Cover Problem.

Generation of problem instance: Similar to independent set, the level of difficulty for a Vertex Cover problem also depends on the size and density of the graph. It is relatively simple to find a vertex cover on a very sparse or very dense graph, compared to a graph of moderate density. Our graph generation for the Vertex Cover problem is similar to that used in the Independent Set problem. That is, the number of vertices is set to a random number from the range $[10, 20]$. If the number of vertices is n , the number of edges vary from $n + 3$ to $2n$. Then our algorithm randomly generates the adjacency matrix maintaining the constraint on the number of vertices and edges. This adjacency matrix is used to generate the problem instance.

4.3.5 Hamiltonian Cycle Problem

Problem definition:

INSTANCE: A graph G .

SOLUTION: A Hamiltonian Cycle in G if there exists one, “No Cycle” otherwise.

Student’s response: Either declare that the graph G does not contain a Hamiltonian Cycle or provide a set of edges that form a Hamiltonian Cycle in G .

Design: Figure 4.5 portrays a screenshot of the exercise. This exercise uses a directed graph. Clicking on an edge toggles it’s state between selected and unselected. The “Reset” button clears all selection and brings

the exercise to its initial state. A student can check the “No Cycle” checkbox to declare that there is no Hamiltonian Cycle present. Clicking on “Check Answer” submits the answer to be validated. If the “No Cycle” button is clicked, then the answer is declared correct if there indeed exists no Hamiltonian Cycle. Otherwise the selected edges are verified to see if they form a Hamiltonian Cycle. The presence or absence of a Hamiltonian Cycle is computed beforehand. If the answer is validated, then the student can repeat the exercise with a different problem instance. If the answer is incorrect, then the student can retry the same problem instance.

Hamiltonian Cycle Problem

Current score: 1 out of 1

Your task in this exercise is to find a Hamiltonian Cycle.

Select the edges that can be a part of a Hamiltonian Cycle.

☐ No Cycle

Answer

Correct! Next Question...

Show hints

Figure 4.5: Exercise for the Hamiltonian Cycle Problem.

Generation of problem instance: Unlike the other exercises, this exercise uses a directed graph. We also ensure in our problem generation algorithm that the graph used for this exercises is connected. The number of vertices for this problem vary from 7 to 9. If the number of vertices is n , number of edges varies from $n + 5$ to $2n$. Once the number of edges and vertices are randomly picked from the given range, the adjacency matrix is populated and a graph is generated. However, when we tested these graphs, most of the time the generated graphs contained no Hamiltonian Cycles. To remedy this we tweaked our graph generation algorithm so that a Hamiltonian Cycle is induced with a probability close to 0.5. We generate a random number and check if its even. If even, we generate a graph using the algorithm similar to previous problems. To this graph, we add extra edges such that a Hamiltonian Cycle is induced. Using this algorithm, we generated 1000 problem instances for this exercise, out of which, 526 instances contained a Hamiltonian Cycle and the remaining 474 instances did not.

4.3.6 Traveling Salesman Problem

Problem definition:

INSTANCE: A weighted, complete graph G where weight of an edge denotes its length.

SOLUTION: The shortest simple cycle passing through all vertices in G .

Student's response: The student needs to provide the set of edges that form the shortest simple cycle containing all vertices in G .

Design: Figure 4.6 portrays a screenshot of the exercise. Clicking on an edge toggles its state between selected and unselected. The “Reset” button clears all selections and returns the exercise to its initial state. A student can click on “Click Answer” to submit an answer. The answer is then validated. Validation happens in steps. It is first checked if the selected edges form a cycle. The second check is if the cycle formed visits all vertices exactly once. The third check is if the cycle is the shortest in length. The length of the shortest cycle is pre-calculated by a non-polynomial time algorithm. If the answer is validated, then the student can do the exercise with a different problem instance. If the answer is incorrect, then the student can retry.

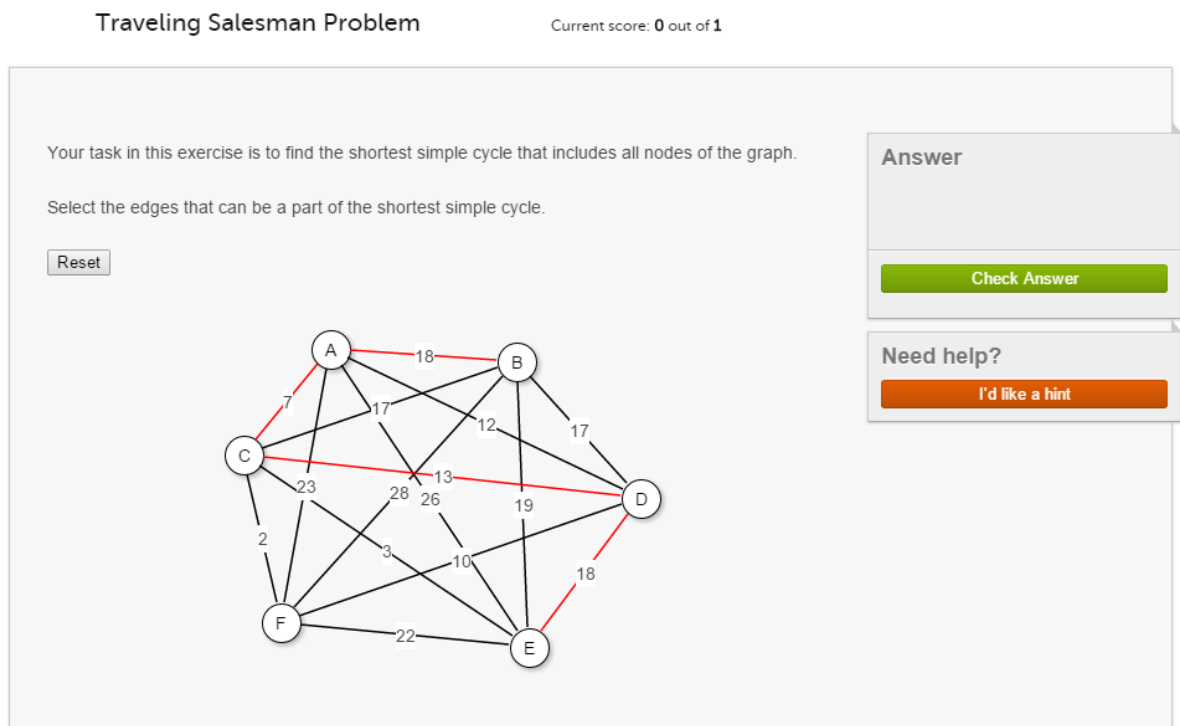


Figure 4.6: Exercise for the Traveling Salesman problem.

Generation of problem instance: The level of difficulty for a Traveling Salesman problem instance depends on the size of the graph. We limit the number of vertices to be either 5 or 6. This is mainly due to the restrictions on the on-screen space. Any more vertices makes the graph look too congested with overlapping

edges, making them difficult to be clicked. This severely hampers usability. However, the level of difficulty also depends on the distribution of the value of weights. If the weights vary a lot, the choice of one edge over another becomes obvious. However if the weights are close enough to each other, then the problem becomes difficult. The number of edges in a 6-node complete graph is 30. We randomly pick the weights of the edges from the range [1,30]. This gives us a wide range of minimum weights for the shortest Hamiltonian Cycle in a problem instance. [See Appendix B]

Chapter 5

Proofs of NP-Completeness

In this chapter we discuss the visualizations created to explain first the concept of reduction and then to explain the special form of reduction used in NP-Completeness proofs. The next section presents the visualizations developed to explain reduction in general. Then, we discuss the use of reduction in proving NP-Completeness and the reduction order that we used for our proofs. In the last section, we discuss the design and implementation details of the visualizations that we created specific to each NP-Complete problem.

5.1 Reduction as a BlackBox

In Computational Complexity Theory, reduction is a way of transforming one problem into another. That is, reduction allows us to solve one problem in terms of another. This makes it an important method for proving bounds on the complexity and degrees of unsolvability of a problem, when that of the other is known. For example, if a problem A can be transformed efficiently to problem B, then the solution of B can be used to solve A. Therefore, problem A can not be harder than problem B, and more importantly (for our purpose), B can not be easier than A.

Reduction is centrally important to complexity theory proofs, but is not straightforward for a beginner to grasp. For example, the mapping of instances between two problems or the correct order of reduction for writing a proof can seem confusing to begin with. Hence we include reduction in our tutorial as a separate module whose objective is to drive the concept of "Reduction as a BlackBox", as shown in Figure 5.1 for transforming an arbitrary instance of problem A to an instance of problem B. It explains how knowing the existence of an efficient transformation and reverse transformation function can be used to construct a lower bound proof for a problem.

We provide visualizations in the form of slideshows to explain three example reductions.

Sorting to pairing:

This example explains how a sorting problem can be solved in terms of a pairing problem. The slideshow explains the steps involved in transforming an instance of sorting to an instance of pairing. It also provides a step by step explanation of how to interpret (reverse transform) the solution of pairing problem to get a

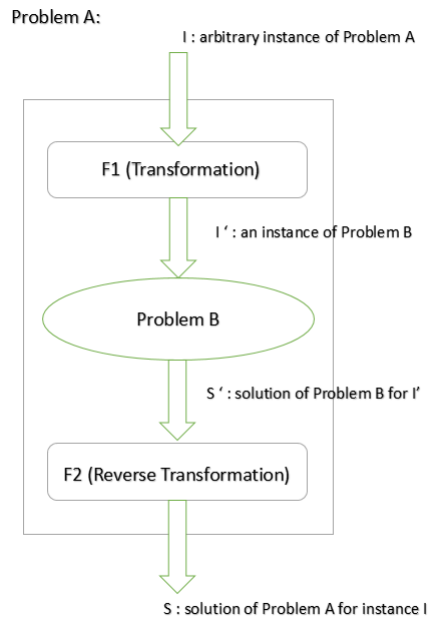


Figure 5.1: BlackBox diagram for general process of reduction

solution for the sorting problem. Figure 5.2 shows a screenshot of the final slide that gives the blackbox representation for this reduction.

Pairing to sorting

This example explains how a pairing problem can be solved in terms of the sorting problem. The slideshow explains the steps involved in transforming an instance of pairing to an instance of sorting, then interpreting (reverse transform) the solution of sorting problem to get a solution for pairing. Figure 5.2 shows the screenshot of the final slide that gives the blackbox representation for this reduction.

Matrix multiplication to Symmetric matrix multiplication

This example explains how a general matrix multiplication problem can be solved if we know how to solve symmetric matrix multiplication. The slideshow explains each step involved in transformation and reverse transformation. Symmetric matrix multiplication might seem to be computationally faster than general matrix multiplication. This example is used to show that it is computationally at least as hard as general matrix multiplication. Figure 5.2 shows the screenshot of the final slide that gives the blackbox representation for this reduction.

5.2 Proving NP Completeness using Reduction

To prove that a given problem is NP-Complete involves proving two things: 1) The problem is in NP (i.e. the problem can be solved in polynomial time on a non-deterministic parallel machine) and 2) It is NP-Hard (i.e.

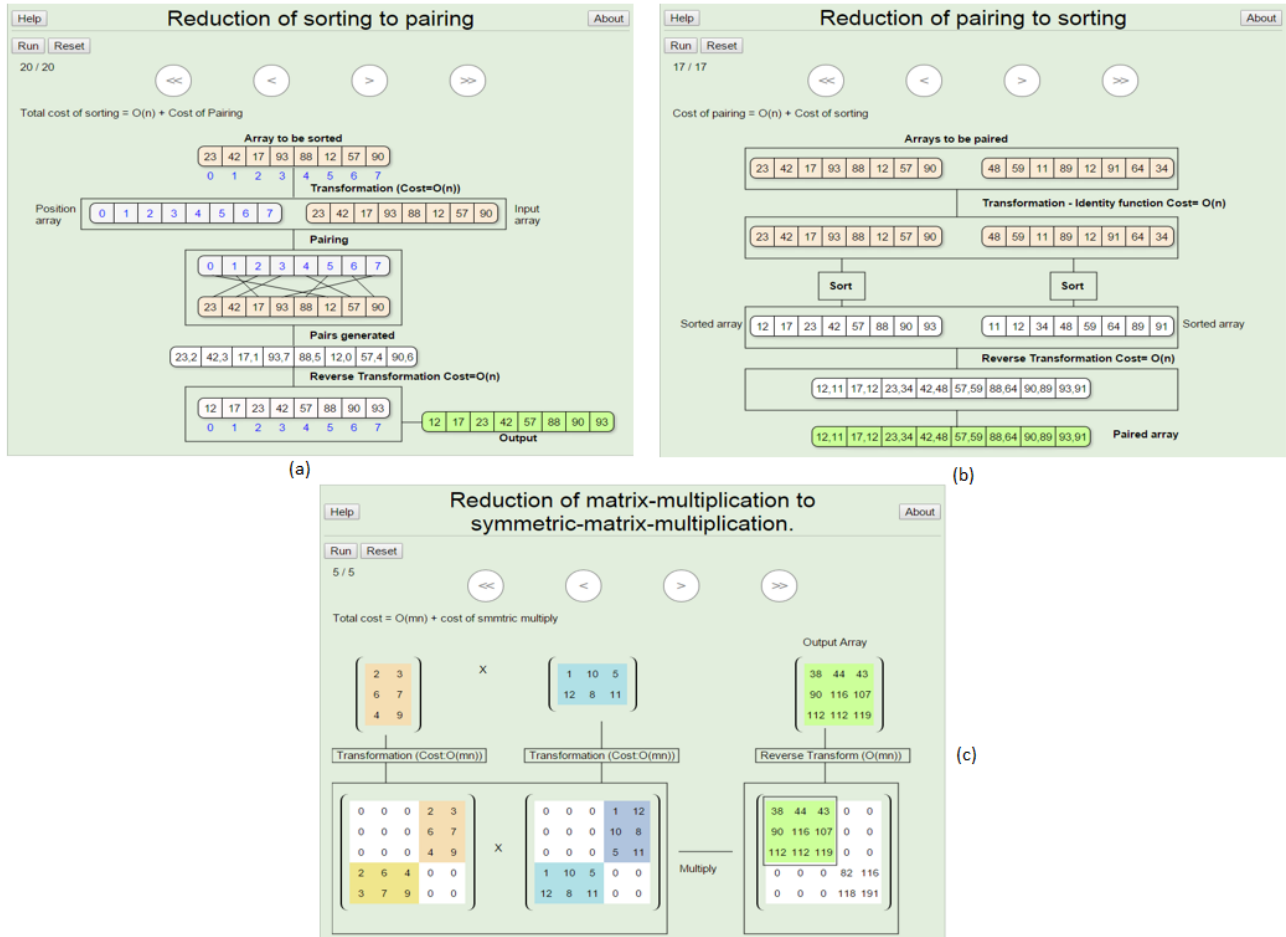


Figure 5.2: Example of reductions.

all problems in NP can be reduced to this problem in polynomial time). The first part of the proof is usually fairly straightforward for a NP-Complete problem. The second part involves proving that the given problem is reducible to a known NP-Complete problem, in polynomial time. These reductions often involve complex transformation functions and constructions that are neither intuitive nor easy to perceive for a beginner in the subject. Some of these reductions involve two problems from entirely different genres, so they appear to a novice to be dissimilar. This adds to the complexity of the proofs. For example, reduction of a 3 CNF Satisfiability problem to a Hamiltonian Cycle problem is not intuitive

Our approach is to develop visualizations in the form of slideshows that use graphical aids to explain the reductions. As mentioned above for the second part of the proof for a particular problem, it is required to reduce it to a known NP-Complete problem in polynomial time. This means that we need to know at least one NP-Complete problem. For our set of proofs, we start from the Circuit Satisfiability problem, that is we take it for a fact that circuit-SAT is a NP-Complete problem. We use this to develop proofs for other problems in the order mentioned in Figure 5.3. We traverse the chart top to bottom. We use reduction of Circuit-SAT to SAT to prove SAT is NP-Hard. We use reduction of SAT to 3-SAT to prove 3-SAT is

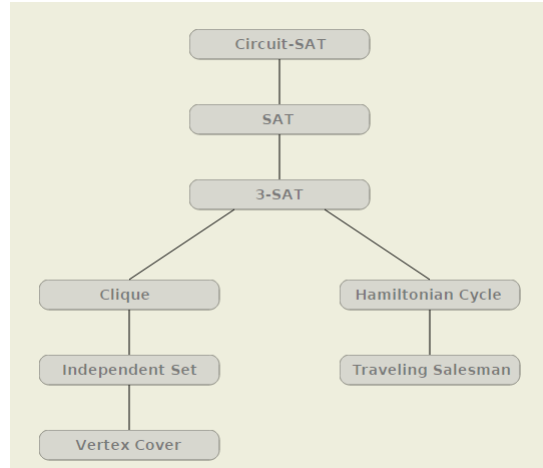


Figure 5.3: Order of reduction for NP-Complete proofs

NP-Hard, and so on.

For each reduction, in our visualizations, we first formalize an if-and-only-if relation between two problems. For example, for reduction of SAT to 3SAT, the formal statement is the SAT formula is satisfiable if and only if its reduced 3-SAT instance is satisfiable. We then provide a detailed proof for the correctness of the if-and-only-if relation, providing detailed descriptions for the transformations used. We then illustrate the transformations with examples for in-depth understanding.

5.3 The Reductions

This section describes the NP-Complete reduction proofs for which we provide slideshows. Design and implementation details are provide for each visualization.

5.3.1 Reducing Circuit-SAT to SAT

Formal statement of claim: An instance C of the Circuit Satisfiability problem can be reduced to a CNF formula ϕ in polynomial time, such that the circuit C is satisfiable if and only if ϕ is satisfiable.

Description : This slideshow starts with a brief background on the relevant laws of Boolean Algebra. We provide an equivalent CNF expression for each type of gate in a circuit. We acknowledge the fact that it is not always easy to perceive why the reduced boolean formula's behavior is equivalent to the gate. So we provide an elaborate step-by-step derivation of the formula. For AND and OR gates, we show a detailed derivation when the number of inputs is 2 or 3, and extend the pattern for multiple inputs. After providing an equivalent CNF formula for each gate, we describe how these can be used to derive a formula for the circuit as a whole. We illustrate this idea with an example circuit, and its equivalent CNF formula using color coding to denote each gate. Figure 5.4 show some screenshots. Finally, we provide an explanation for how the circuit is satisfiable if and only if this equivalent boolean formula in CNF is satisfiable.

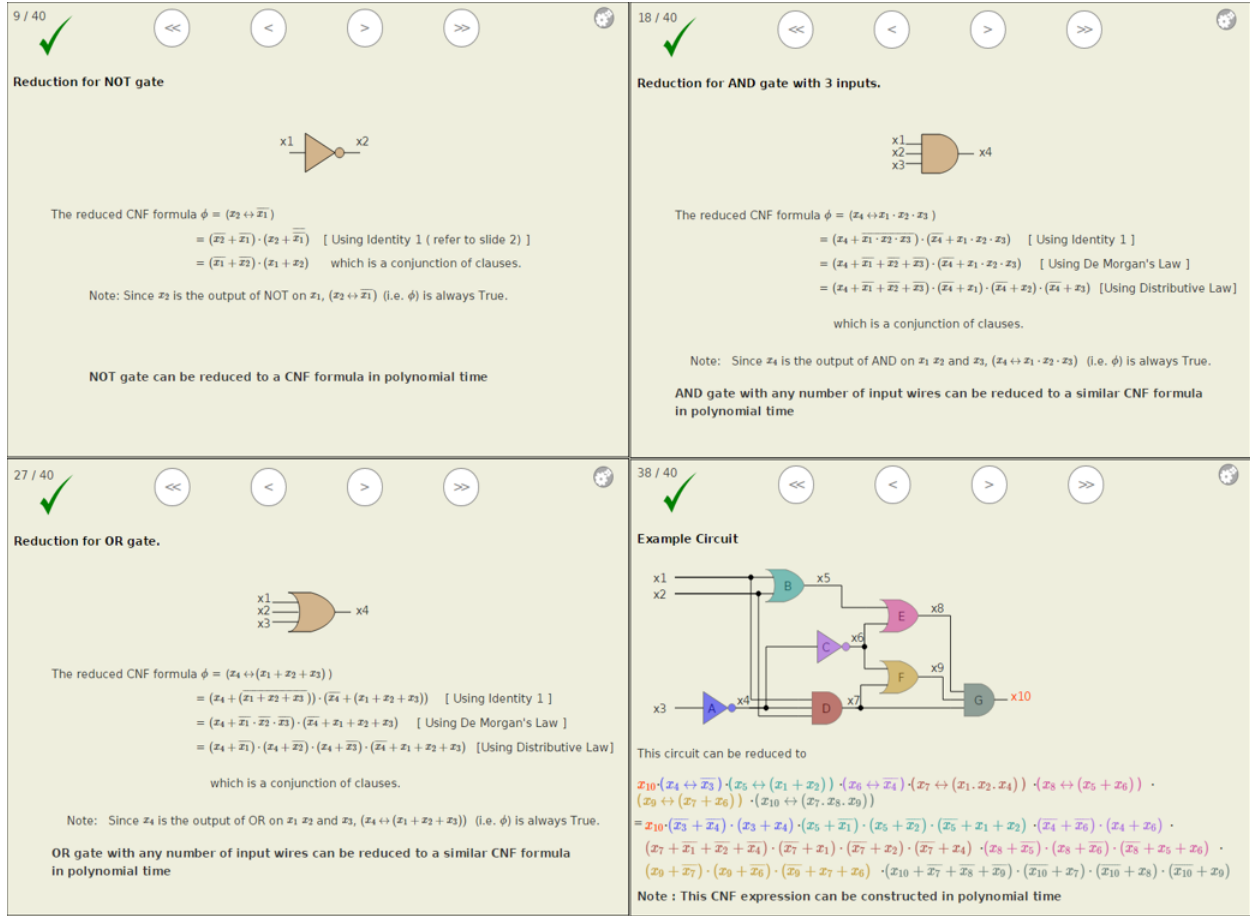


Figure 5.4: Screenshots of Circuit-SAT to SAT reduction visualization.

5.3.2 Reducing SAT to 3-SAT

Formal statement of claim: An instance ϕ of the Formula Satisfiability problem can be reduced in polynomial time to a 3-CNF formula β in polynomial time, such that ϕ is satisfiable if and only if β is satisfiable.

Description: We explain the reduction of SAT to 3-SAT on a case by case basis. We provide an equivalent 3-CNF formula for each type of clause in the CNF formula that might be a part of the input to a SAT problem. A clause in a CNF can have either 1, 2, 3, or more than 3 literals. We consider each of these possibilities individually. For clauses containing 1 or 2 literals, we reduce the clause to a 3 CNF formula and provide truth tables to prove that they are actually equivalent. For clauses with more than 3 literals, we provide a 3-CNF formula and provide a step by step explanation using color coding and animation to explain why the 3 CNF formula is satisfiable when the clause is satisfiable, and not-satisfiable when the clause is not satisfiable. Figure 5.5 shows some screenshots.

16 / 54

Case 2: Reduction of clauses containing one literal

Let $C_i = l_i$ where l_i is a literal.

Introduce 2 new variables $y_{i,1}$ and $y_{i,2}$.

Replace C_i by conjunction of clauses Z_i where

$$Z_i = (l_i + y_{i,1} + y_{i,2}) \cdot (l_i + \overline{y_{i,1}} + y_{i,2}) \cdot (l_i + y_{i,1} + \overline{y_{i,2}}) \cdot (l_i + \overline{y_{i,1}} + \overline{y_{i,2}})$$

I II III IV

Truth Table :

l_i	$y_{i,1}$	$y_{i,2}$	I	II	III	IV	Z_i
T	T	T	T	T	T	T	T
T	T	F	T	T	T	T	T
T	F	T	T	T	T	T	T
T	F	F	T	T	T	T	T
F	T	T	T	T	T	F	F
F	T	F	T	T	T	F	F
F	F	T	T	T	T	F	F
F	F	F	T	T	T	F	F

When C_i (i.e. l_i) is True, Z_i is true.
When C_i (i.e. l_i) is False, Z_i is false.

Hence C_i can be reduced to Z_i where each clause in Z_i contains exactly 3 literals and $C_i \iff Z_i$.

28 / 54

Case 3: Reduction of clauses containing two literals.

Let $C_i = (l_{i,1} + l_{i,2})$ where $l_{i,1}$ and $l_{i,2}$ are literals.

Introduce a new variable y_i .

Replace C_i by conjunction of clauses Z_i where

$$Z_i = (l_{i,1} + l_{i,2} + y_i) \cdot (l_{i,1} + l_{i,2} + \overline{y_i})$$

I II

Truth Table :

$l_{i,1}$	$l_{i,2}$	y_i	C_i	I	II	Z_i
T	T	T	T	T	T	T
T	T	F	T	T	T	T
T	F	T	T	T	T	T
T	F	F	T	T	T	T
F	T	T	T	T	T	T
F	T	F	T	T	T	T
F	F	T	T	T	T	T
F	F	F	T	T	T	T

When C_i is true, Z_i is true.
When C_i is false, Z_i is false.

Hence C_i can be reduced to Z_i where each clause in Z_i contains exactly 3 literals and $C_i \iff Z_i$.

52 / 54

Case 4b. When C_i is not satisfiable.

$C_i = (l_{i,1} + l_{i,2} + l_{i,3} + \dots + l_{i,k})$ where $k > 3$.

$$Z_i = (l_{i,1} + l_{i,2} + y_1) \cdot (\overline{y_1} + l_{i,3} + y_2) \cdot (\overline{y_2} + l_{i,4} + y_3) \cdot (\overline{y_3} + l_{i,5} + y_4) \cdot (\overline{y_4} + l_{i,6} + y_5) \cdot (\overline{y_5} + l_{i,k-1} + y_k) \cdot (\overline{y_k} + l_{i,k} + y_{k+1})$$

When C_i is not satisfiable NO literal in $\{l_{i,1} \dots l_{i,k}\}$ is True.

For Z_i to be satisfiable, all its clauses must evaluate to True

For the first clause to be True, $y_1 = \text{True}$

Now for the second clause to be True, $y_2 = \text{True}$

Similarly for the third clause to be True, $y_3 = \text{True}$

$y_1 = \text{True} \Rightarrow y_4 = \text{True} \Rightarrow y_2 = \text{True} \Rightarrow y_3 = \text{True} \Rightarrow y_5 = \text{True} \Rightarrow y_6 = \text{True} \Rightarrow y_7 = \text{True} \Rightarrow y_8 = \text{True} \Rightarrow y_9 = \text{True}$

The last clause evaluates to False. Hence Z_i is NOT SATISFIABLE

43 / 54

Case 4a. When C_i is satisfiable.

$C_i = (l_{i,1} + l_{i,2} + l_{i,3} + \dots + l_{i,k})$ where $k > 3$.

$$Z_i = (l_{i,1} + l_{i,2} + y_1) \cdot (\overline{y_1} + l_{i,3} + y_2) \cdot (\overline{y_2} + l_{i,4} + y_3) \cdot (\overline{y_3} + l_{i,5} + y_4) \cdot (\overline{y_4} + l_{i,6} + y_5) \cdot (\overline{y_5} + l_{i,k-1} + y_k) \cdot (\overline{y_k} + l_{i,k} + y_{k+1})$$

C'

When C_i is satisfiable atleast one literal in $l_{i,1} \dots l_{i,k}$ is True.

If $l_{i,j}$ (where $j \in \{1, 2, k-1, k\}$) is True, set $y_1 \dots y_{j-2}$ to True and $y_{j-1} \dots y_{k-3}$ to False.

Let us call the clause in Z_i containing $l_{i,j}$ in as C'

The third term of all the clauses in Z_i left to C' has a literal y_n (where $n \in \{1, j-2\}$) which evaluates to True

The first term of all the clauses in Z_i right to C' has a literal $\overline{y_n}$ (where $n \in \{j-1, k-3\}$) which evaluates to True

Z_i has a satisfying assignment

Figure 5.5: Screenshots of SAT to 3-SAT reduction visualization.

5.3.3 Reducing 3-SAT to Clique

Formal statement of claim: Any boolean formula ϕ in 3-CNF with k clauses can be reduced in polynomial time to a graph G such that the ϕ is satisfiable if and only if G contains a k -Clique.

Description: This reduction involves constructing a graph from the 3-CNF formula. We provide a thorough explanation of this construction process. We use color codes to express the association between clauses in the formula and nodes in the graph. We use step-by-step animation to show how the nodes are to be connected. The example used for this visualization comes from Cormen, et al. [14]. This example graph is also used to augment our textual explanation for how the presence of a k -clique in the graph determines a presence or absence of a satisfying assignment for the 3-CNF formula, with a graphical illustration. Figure 5.6 provide some screenshots of slides.

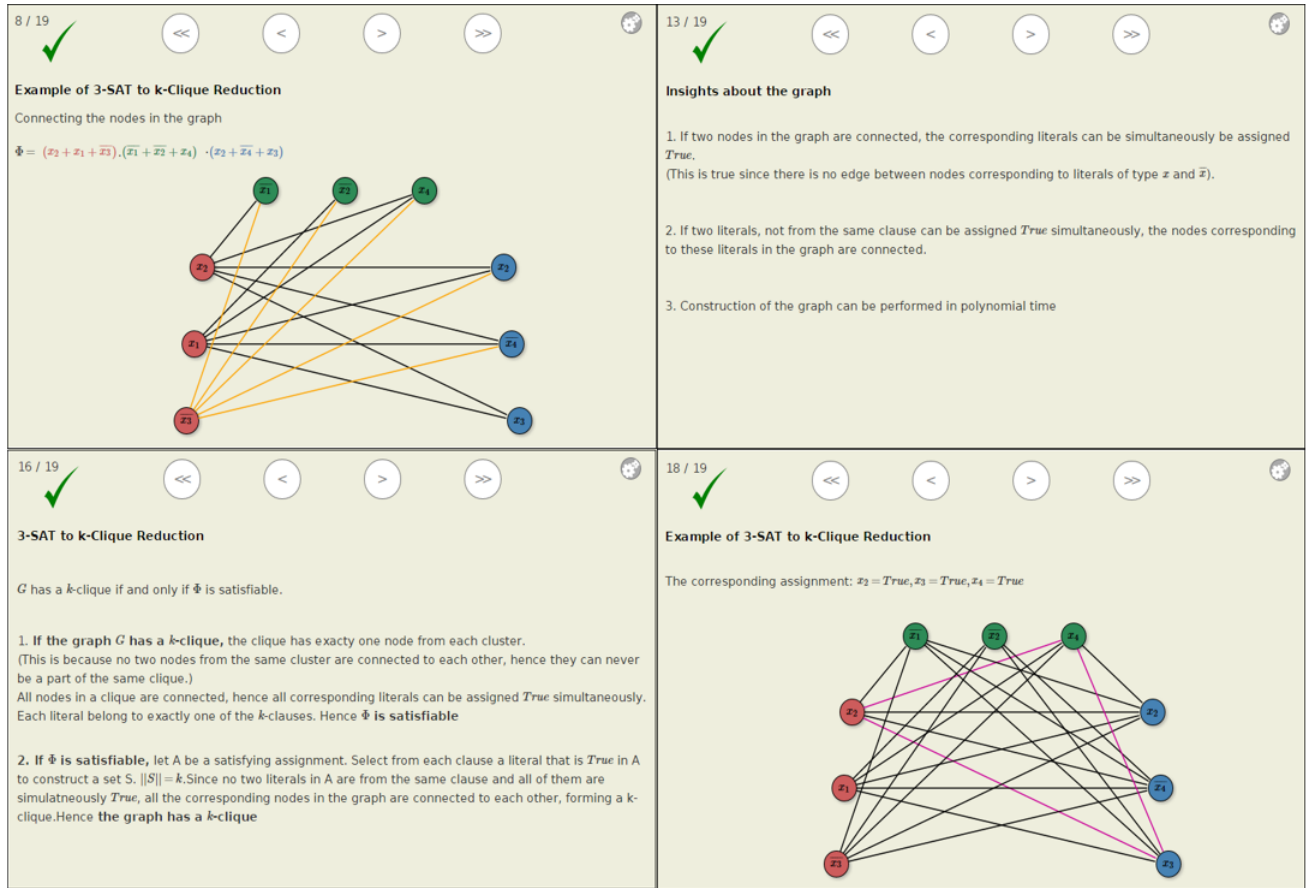


Figure 5.6: Screenshots of 3-SAT to Clique reduction visualization.

5.3.4 Reducing Clique to Independent Set

Formal statement of claim: An instance of the clique problem G can be reduced in polynomial time to a graph G' such that the G contains a clique of size $\geq k$ if and only if G' contains an independent set of size $\geq k$.

Description: The input to the clique problem is a graph. This graph is transformed to its inverse to serve as an instance to the Independent Set problem. We explain this transformation using visual aids. The construction of the reverse graph from the original is first explained textually and then illustrated with an example, using slides with intermediate visuals of the two graphs being superimposed onto one. Figure 5.7 shows some screenshots. This example graph is used to portray how an independent set in the reduced graph forms a clique in the original. The last few slides in this visualization use this fact to show that the presence or absence of an independent set of size k in the reduced graph determines the presence or absence of a clique of size k in the original.

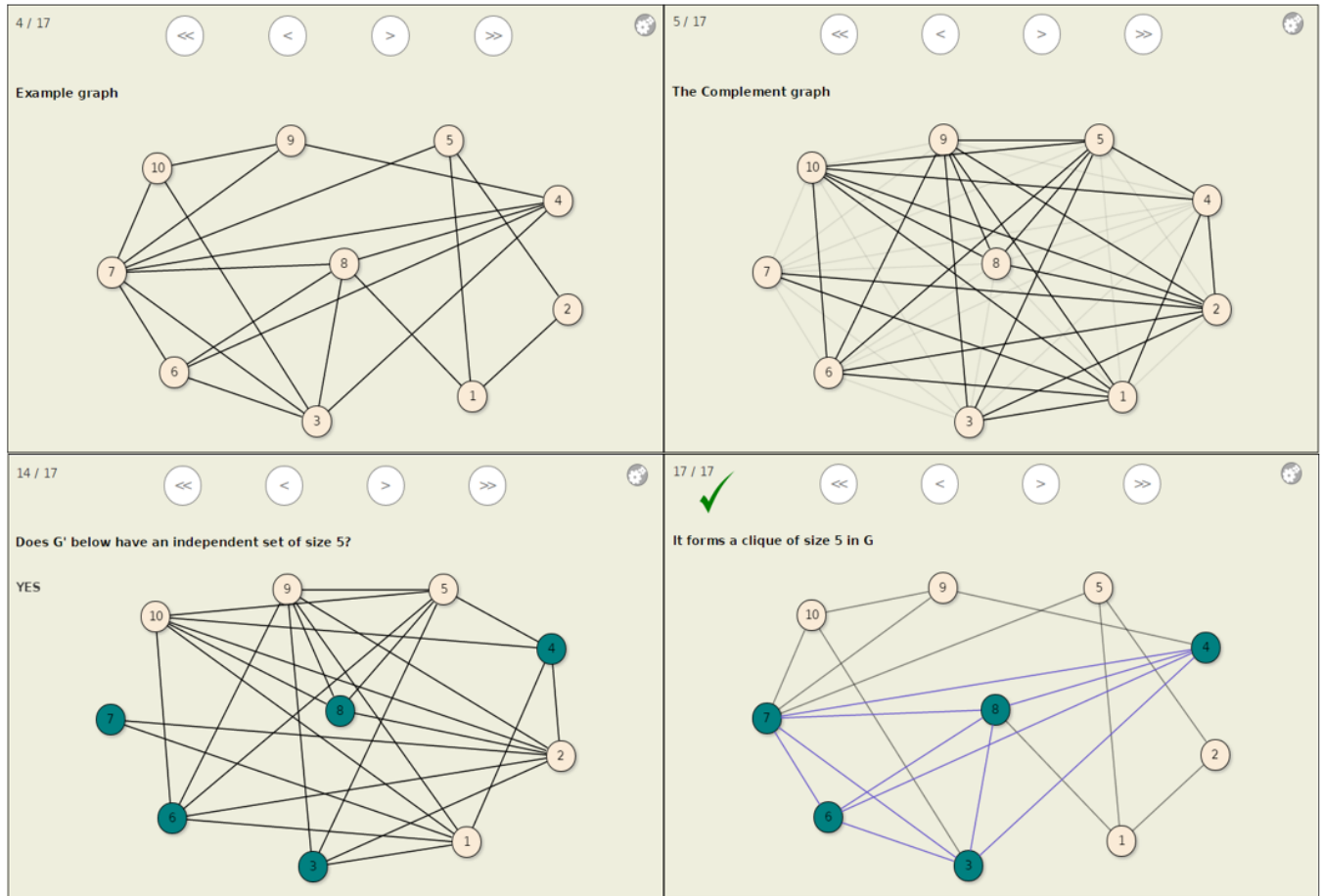


Figure 5.7: Screenshots of Clique to Independent Set reduction visualization.

5.3.5 Reducing Independent Set to Vertex Cover

Formal statement of claim: Any instance of an Independent Set Problem $G = (V, E)$ can be reduced in polynomial time to a graph G' such that an independent set of size $\geq k$ is present in the G if and only if the G' contains a Vertex Cover of size $\leq |V| - k$.

Description: This is a relatively simple reduction because of the fact that the transformation function is the Identity function. We use example graphs to show how the vertices not included in an Independent Set in a graph forms a vertex cover in the same graph. We use color coding to associate an edge to a vertex to illustrate the fact that all edges are indeed covered by the set of vertices that are not a part of the independent set. We use examples of both types (i.e. with a “yes” and a “no” answer since it is a decision problem), to graphically demonstrate that the given reduction works. Figure 5.8 shows key slides.

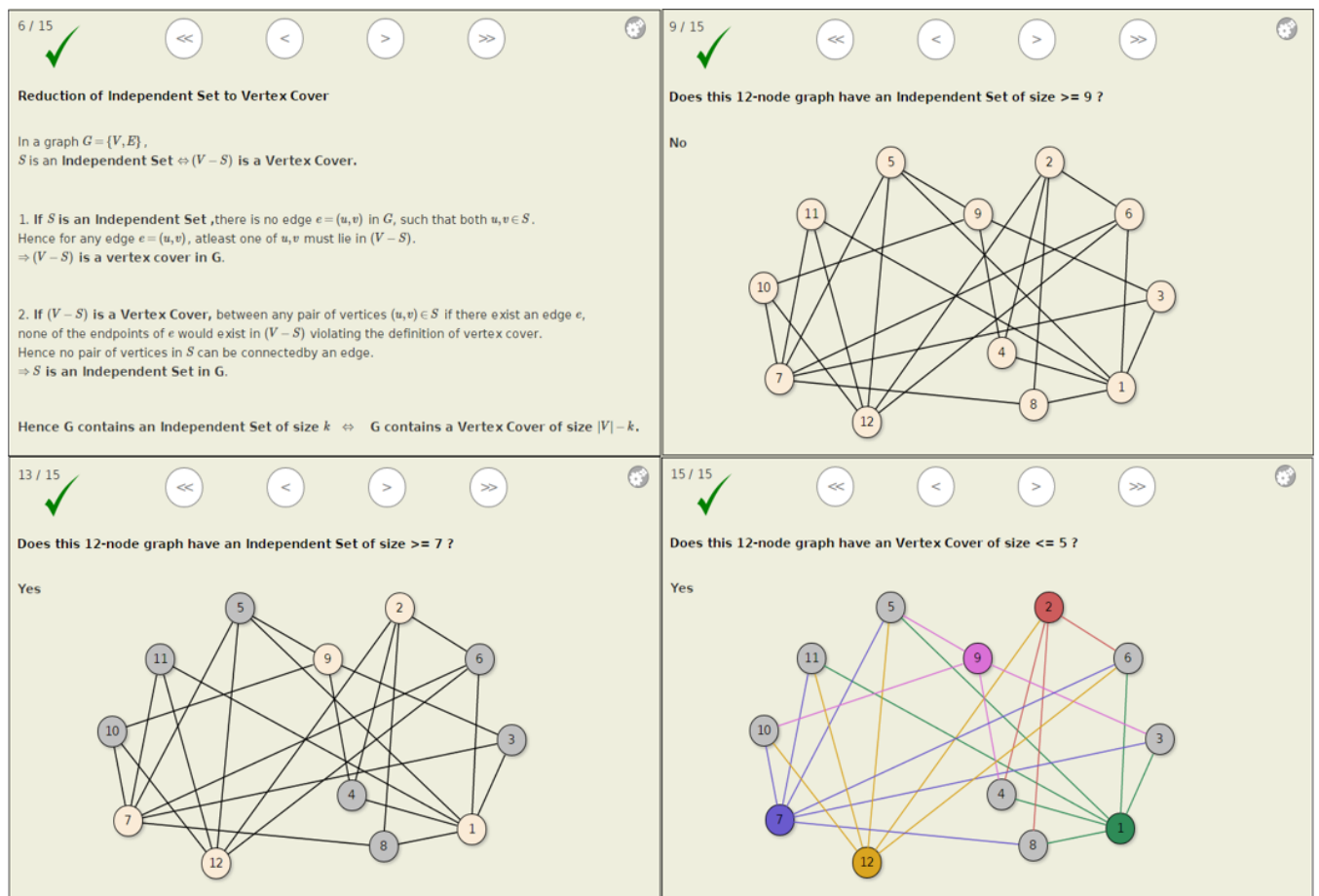


Figure 5.8: Screenshots of Independent Set to Vertex Cover reduction visualization.

5.3.6 Reducing 3-SAT to Hamiltonian Cycle

Formal statement of claim: Any boolean formula ϕ in 3-CNF with k clauses can be reduced in polynomial time to a directed graph G' such that the ϕ is satisfiable if and only if G contains a Hamiltonian Cycle.

Description: This reduction includes a construction of a graph from the given 3-CNF formula. This construction is complex in nature and hence difficult to understand. Hence this visualization takes care to explain every step of the construction in detail. First the nodes are mapped to boolean variables in the 3-CNF. The animation zooms in to each variable, one by one, and draws the nodes on screen. For each clause, it draws the corresponding node on the screen using color-coding to associate the node with the corresponding clause. For each edge added to the graph, an explanation is provided about the interpretation of the direction of the edges. Once the graph is constructed, the visualization provides a textual description of how the presence of a Hamiltonian Cycle in this graph determines the satisfiability of the 3-CNF formula. This is then demonstrated visually by highlighting the Hamiltonian Cycle in the same graph and using the cycle to find a satisfying assignment for the 3-CNF formula. Figure 5.9 shows key slides.

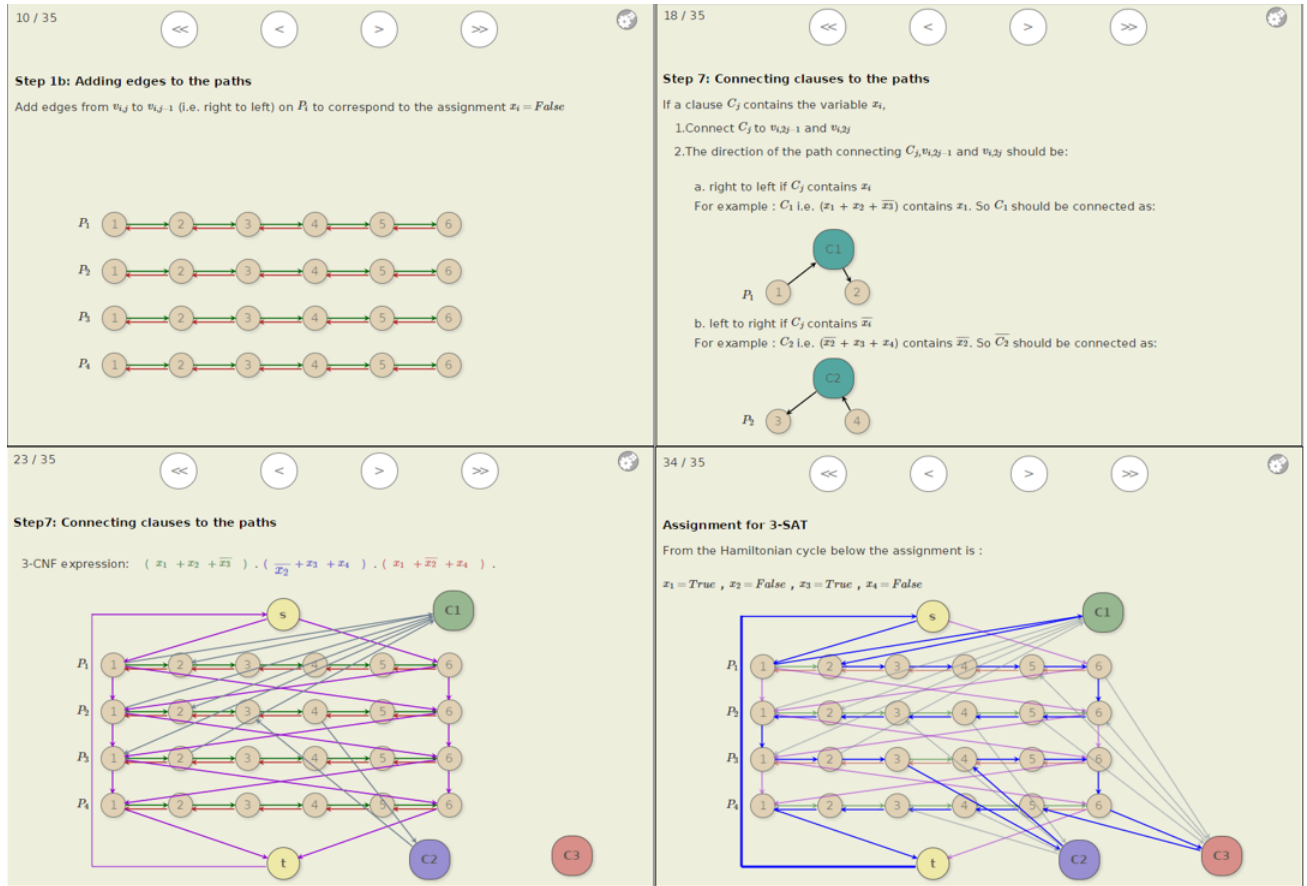


Figure 5.9: Screenshots of 3-SAT to Hamiltonian Cycle reduction visualization.

5.3.7 Reducing the Hamiltonian Cycle Problem to Traveling Salesman

Formal statement of claim: An instance G of a Hamiltonian Cycle problem can be reduced in polynomial time to a weighted graph G' such that G has an Hamiltonian Cycle if and only if the G' has a solution for Traveling Salesman with cost ≤ 0 .

Description: In this reduction the input is a non-weighted graph. The reduction transforms it into a weighted graph that can act as an instance for a Traveling Salesman problem. An example is used to portray this reduction with differently colored edges to distinguish between the original and transformed graph. Then the example graph is used to graphically demonstrate how the presence of a solution for Traveling Salesman Problem in the transformed graph can be used to determine the presence of a Hamiltonian Cycle in the original. Figure 5.10 shows some key slides.

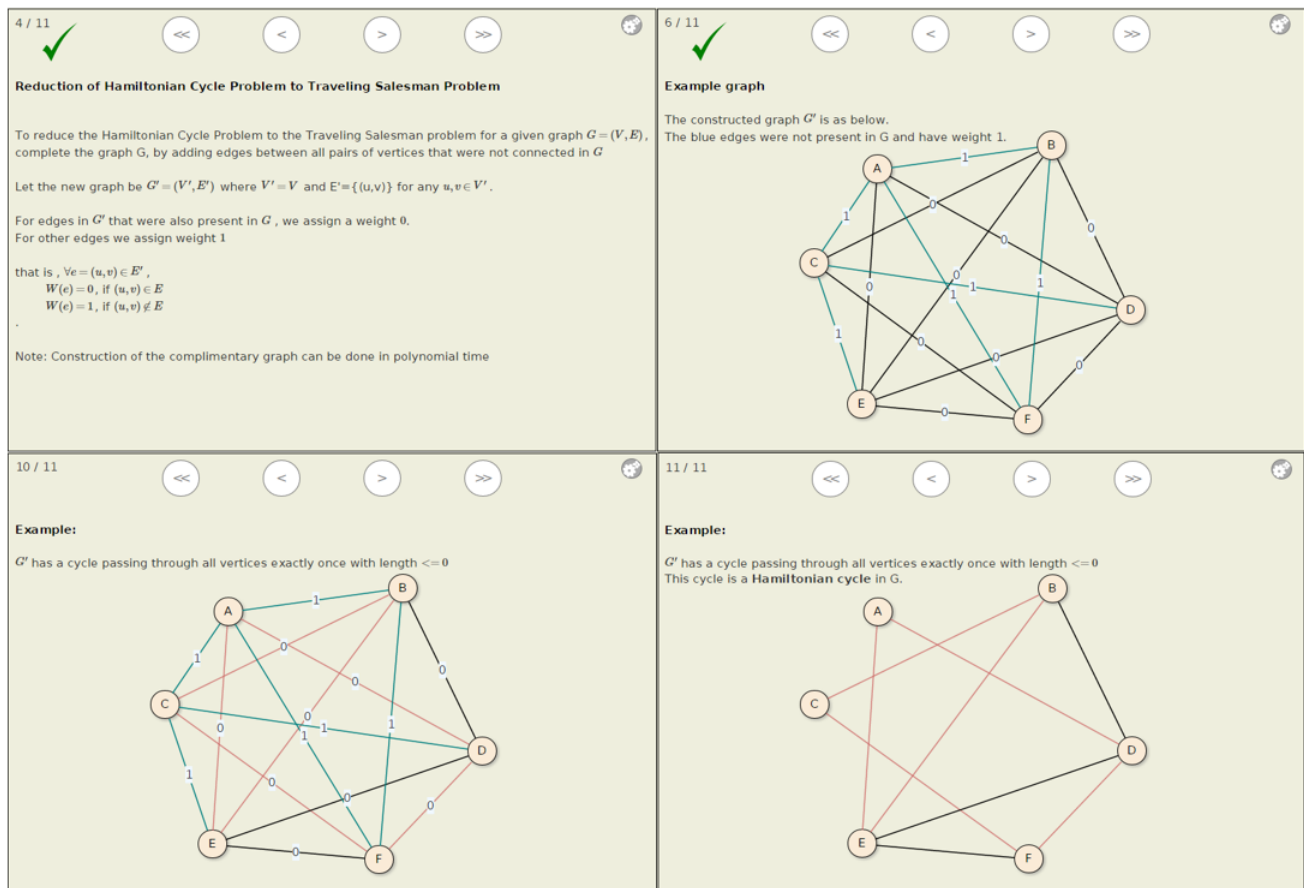


Figure 5.10: Screenshots of Hamiltonian Cycle to TSP reduction visualization.

Chapter 6

Evaluation of Usage Logs and Survey Feedback

In this chapter we report on our efforts to evaluate the use of the tutorial by students. We first describe our research population for this evaluation. We then describe in detail our methods of analysis and the corresponding results for each method.

6.1 Research Population

Our tutorial for NP-Completeness was introduced as supplementary material in two courses, (CS 5114 and CS 4104) that were taught in Spring, 2015 at Virginia Tech. CS 5114 is a graduate level course on Theory of Algorithms. CS 4104 is a senior level undergraduate course on Data and Algorithm Analysis. Both courses include NP-Completeness in their syllabus, but with different levels of coverage. CS 5114 covers the topic in greater detail than does CS 4104. We factor this difference into our analysis, and report our results separately for each course. Our evaluation is based on the tracking of online events (such as clicks) logged by our system, and on the feedback collected from students about their experience with the material. The size of the class for CS 5114 was 34, and that of CS 4104 was 62. Although the content of the tutorial for both courses was the same, they were released as two different book instances for the two classes.

We want to emphasize the fact that the use of the tutorial by the students was completely voluntary. None of the exercises were graded. No extra points or incentive of any sort was provided to the students for adopting the tutorial.

6.2 Evaluation of Usage Based on Event Logs

The OpenDSA framework has a logging system (documentation available at [3]) that stores event logs in a MySQL database. The events stored records of interactions that occur in the tutorial, for example, load a page, gain or lose focus on a window, click on a slideshow, etc. We analyze these events to estimate the amount of time spent by students on a particular module or exercise, or on the whole tutorial. We plot these

time values to understand the use of the tutorial with respect to when the topics are taught in class or when the assignments are due. First, we present the analysis the event log over a period of 30 days for CS 5114 and 24 days for CS 4104. This is the period of time during which NP-Completeness was taught in these classes. Then, we also present an analysis of overall usage extending to the last day of the semester to figure out the usage patterns before the final examination.

One of our major challenges for analyzing the logs was differentiating the usage by one student versus another. Although the students were encouraged to create a user id of their own, it was not a requirement. When a student accesses the tutorial without having a user id, the events get logged as a guest user. This means that the interactions logged under the guest user correspond to multiple students, making it hard to distinguish. For our purposes, for events logged under the guest user, we treat interactions from distinct IP-addresses as separate users. We acknowledge that the IP addresses do not exhibit a one-to-one mapping with actual users. Multiple users may share the same public IP in a shared network, or the same user may connect using different addresses. However, in our analysis we distinguish between IP addresses only to isolate one session from another while calculating our timing estimates. We report only the total time spent by all students on a particular module or exercise. Hence we expect the range of error generated by treating each IP address as a separate user to be reasonably small. For users who log in using their own user ids, we do not make this distinction.

Another challenge that we faced was in distinguishing idle time (time when the tutorial is open, but the student is not looking at it) from productive time (the student is actually working through the tutorial). We calculate the time of usage based on the timestamp of logged events. This means that if a student opens a HTML page, and then leaves the browser window open, it is possible that our calculations may end up including the entire time interval until the window either loses focus or is unloaded. There is no way for us to differentiate between the student actually looking at the material and just leaving the browser open. To minimize the problem, we cap each interval to a maximum of 10 minutes. That is, if the gap between two events is more than 10 minutes, we assume the extra time to be idle time and include only 10 minutes for our calculations. Hence our timing estimates are shorter than actual recorded times. Our estimates may not be accurate, but this does not alter the distribution and variance, and hence does not prevent us from studying the trends of use over the time period.

We report our observations from the event logs separately for CS 5114 and CS 4104 in the following subsections.

6.2.1 Analysis for CS 5114

Curriculum

NP-Completeness was introduced to the class on 02-24-2015. The tutorial was made available to the students on 03-12-2015. Table 6.1 lists the topics and the dates on which they were taught.

Categorized and per-module Analysis of use

The objective of this analysis is to get an idea of the amount of time spent by students in reading the textual content, doing slideshows, or looking at proofs. We divide our tutorial into four categories for this analysis.

Table 6.1: Curriculum for CS 5114.

Date	Topic
03-17-2015	NP and Reducibility
03-19-2015	Circuit-SAT, NP-Complete Problems
03-24-2015	SAT, Proof of NP-Completeness
03-26-2015	3-SAT
03-31-2015	Clique to Vertex Cover Reduction, Hamiltonian Cycle and TSP
04-02-2015	Subset Sum
04-10-2015	Homework 7 (on NP-Completeness) due.

They are *Theory*, *Reduction*, *Intro*, and *Proofs*. *Theory* includes modules with just textual content and no visualizations. *Reduction* includes the module that introduces the general concepts of reduction. This module includes both text and visualizations related to Reduction. *Intro* includes the modules that introduce the NP-Complete problems. These modules have no textual content and are comprised of visualizations to introduce the problem, and a practice exercise for a problem instance. We show later that the time spent on exercises was very little, so it is safe to relate the usage reported under this category to the slideshows introducing the problem. *Proof* includes modules that provide visualizations for reductions of NP-Complete problems. They also have no textual content.

Figure 6.1 plots the time spent by students on the corresponding categories on particular dates. Figure 6.2 plots the per-module use of the tutorial over time. This graph is similar to Figure 6.1, with each category broken into individual modules. We observe that on the initial days, March 17 and 19, students spent time looking at the theoretical content and the concept of reduction. This is in tune with the topics taught in the class. We notice a spike in use from the 26th to 30th of March. Important to note here is that March 27 was the due date for homework 6 (the homework before the NP-Completeness homework). So arguably, students start looking into the homework on NP-Completeness (i.e. homework 9) around this time. Another important observation is that during this period, students spend time looking at visualizations that introduce the NP-Complete problems as well as reductions and the textual content about how to prove NP-Completeness. From Figure 6.2, we can see that usage is not restricted to one particular problem. The students seem to be looking at multiple problems at this point. We assume from this observation that students used the tutorial to study for their homework. Our belief is reinforced by the spikes in the graph around April 9 and 10, which is close to the homework deadline. During this time students seem to be spending time on the modules on reduction of NP-Complete Problems.

The total time spent on the tutorial over this period was calculated to be 50.66 hours, which when averaged over the entire strength of the class, results to about 1.5 hours per student. As mentioned earlier, this is a stricter estimate where we attempt to remove the idle time.

Analysis of slideshows based on clicks

This analysis is strictly constrained to the use of slideshows, when a student actually clicks on them. We calculate the time between these clicks. Figure 6.3 plots the time spent over the period of 30 days. An

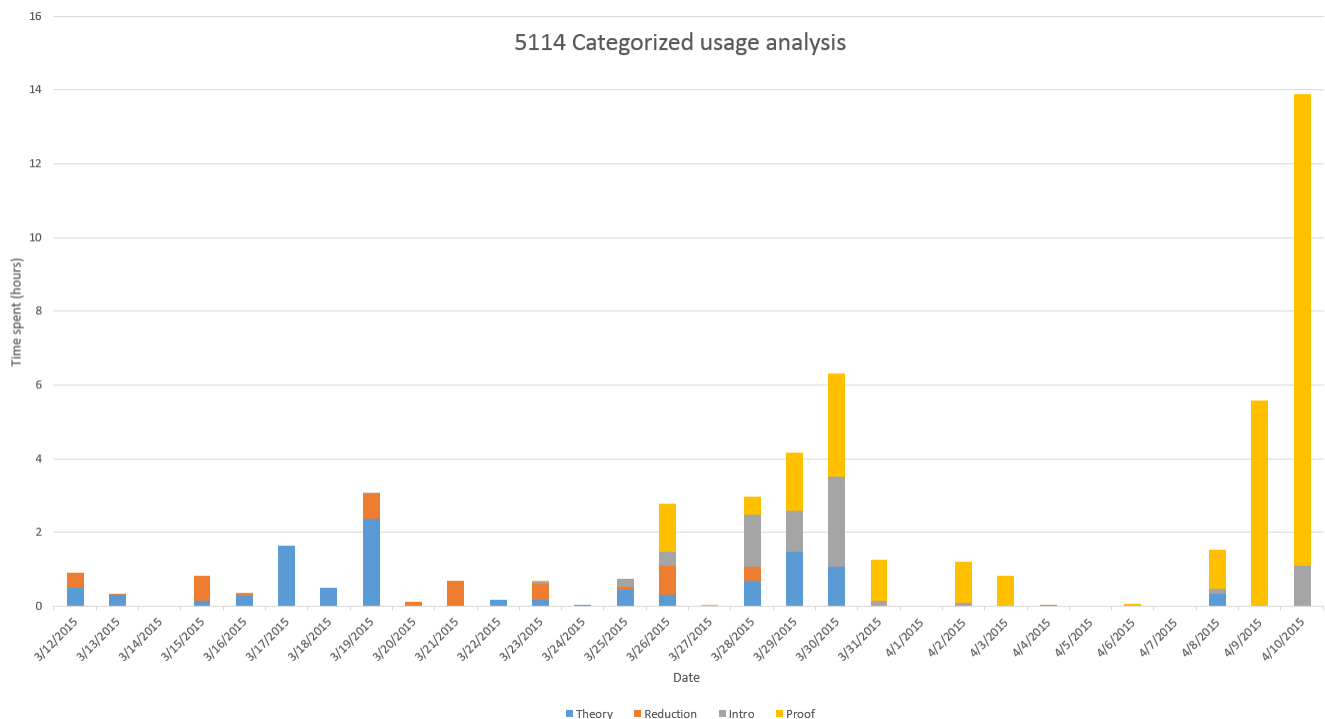


Figure 6.1: Categorized usage in CS 5114 by the entire class.

important observation here is the interest in specific visualizations around April 9-10. This is close to the deadline for submission of homework 7. Students spent a lot of time looking at the visualization of reduction of 3-SAT to Hamiltonian Cycle and a reasonable amount of time looking at the reduction from 3-SAT to Clique. Both of these are examples of reduction from 3-SAT to a graph problem, similar to one of their homework questions. The reason behind the prominent spike in the graph corresponding to the 3-SAT to Hamiltonian Cycle reduction was due to the fact that the homework question required students to construct a complex gadget for their solution. We infer that this particular visualization having similar construction proved helpful in solving the particular problem in their homework.

The total time spent as shown in this analysis is close to 12.17 hours which is about 21 minutes per student when averaged over all students in the class. This excludes the time spent reading the textual content. As we mentioned earlier, this is a stricter estimate.

6.2.2 Analysis for CS 4104

Curriculum

NP-Completeness was introduced to the class on 03-31-2015. The tutorial was made available to the students on 04-01-2015. Table 6.2 lists the topics and the dates on which they were taught.

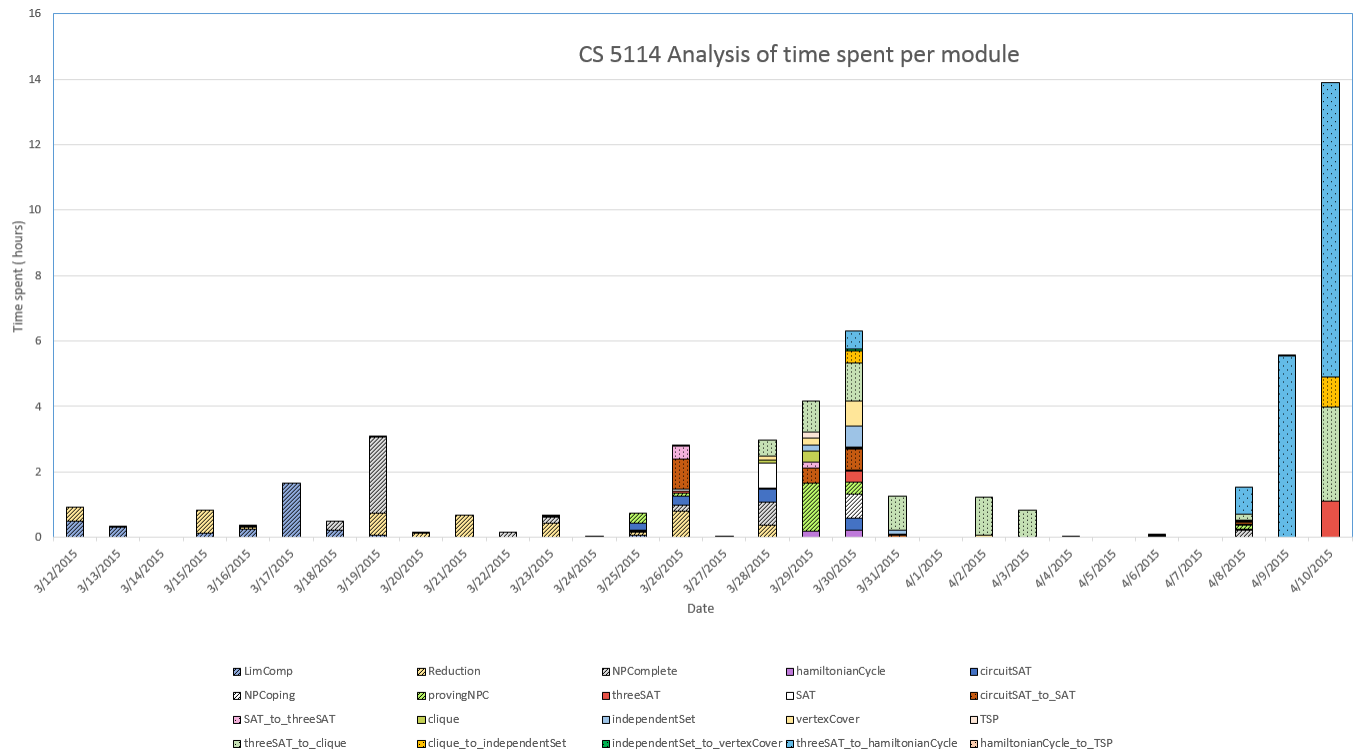


Figure 6.2: Usage per module in CS 5114 by the entire class.

Categorized and per-module analysis of usage

The categories and calculations of time estimate for this analysis are the same as described in Section 6.2.1. Figure 6.4 shows the categorized analysis over time and Figure 6.5 breaks it into individual modules. We observe that the majority of the time spent was on the textual content and the module on reduction. For this class also, we notice a spike in use of the tutorial close to the homework deadlines.

However, the total time spent on the material for this class is only about 9.96 hours which is a little less than 10 minutes for each student when averaged over the total strength of the class. This is considerably lower than for CS 5114. However, NP-Completeness is covered in a greater detail in CS 5114. The homework problems in CS 5114 are more difficult and more closely aligned to the content of the tutorial. So this result does not surprise us.

Analysis of slideshows based on clicks

Figure 6.6 shows the time spent on the tutorial when the student clicks on a slideshow, distributed over the period. The total time spent is about 41 minutes. This is similar to our previous analysis, where we conclude that the major portion of time spent by the class on the tutorial is on its textual content.

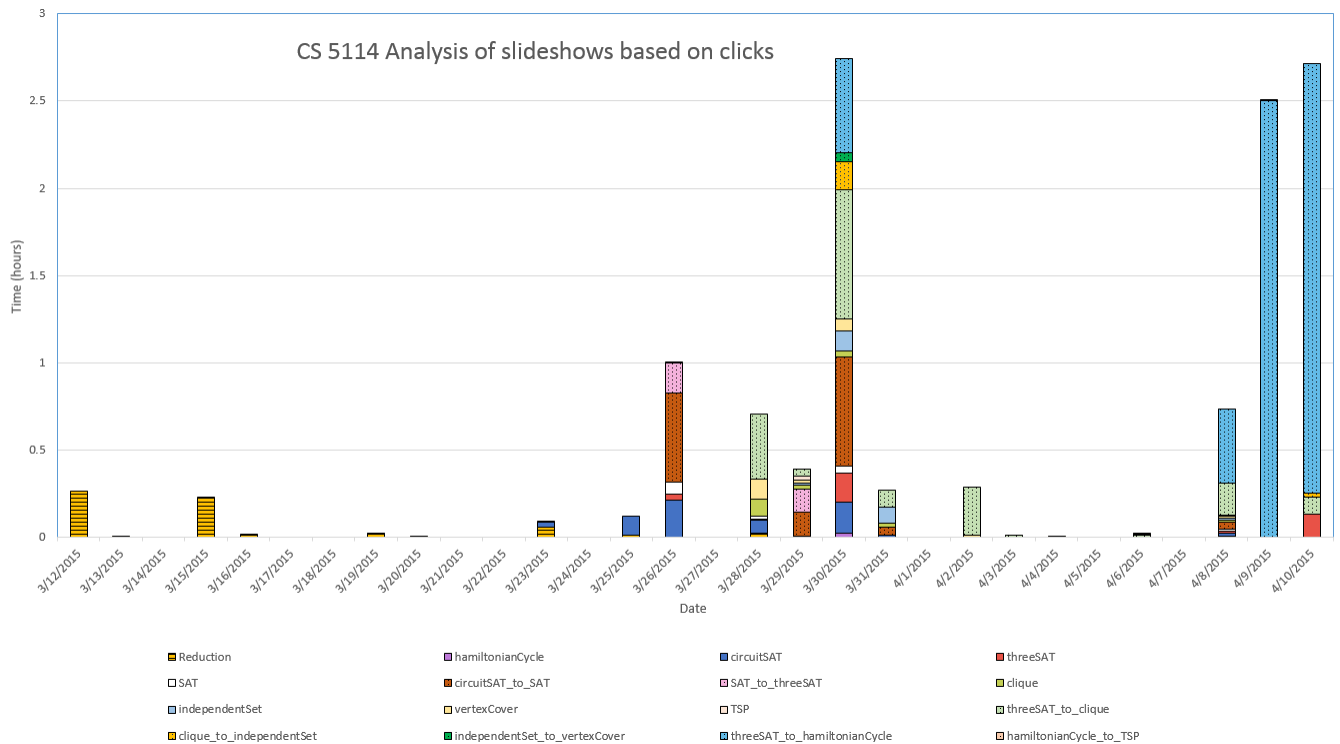


Figure 6.3: Usage of slideshows in CS 5114 for the entire class.

6.2.3 Analysis of Exercises

The practice exercises included in the tutorial do not seem to be used much by the students. Only 4 students in CS 5114 and CS 4104 combined have attempted the exercises, and the total time spent is 1.2 hours. Figure 6.7 shows the usage. We conclude from this that students do not spend time on exercises when they are not mandated or graded.

6.2.4 Usage of the tutorial for the finals

Figure 6.8 shows the use of the tutorial by the students in the CS 5114 class. As we see in the figure, the students seem to use the tutorial a lot before their homeworks, but do not use it much to prepare for their finals. However, the final examination in CS 5114 was a take-home examination. So the students knew the questions they had to answer to pass their finals. But in CS 4104, the final examination was not an open book test, which means the students had to study the entire syllabus for their finals, for which the tutorial came handy. As we see in Figure 6.9, they use the material heavily during the last two days before their final examination. The time spent on the tutorial by the class of CS 4104 in the last two days of the semester is more than 50% of the total time spent by the class in the entire semester.

Table 6.2: Curriculum for CS 4104.

Date	Topic
04-7-2015	Concepts of Reduction
04-9-2015	Cook's theorem, SAT
04-14-2015	Proof of NP-Completeness
04-17-2015	Homework 8
04-24-2015	Homework 9 due

6.3 Feedback from students

We provided the students in both classes with a questionnaire (see Appendix A), in order to obtain their feedback on the materials. The following sections summarize the results.

6.3.1 CS 5114

The survey was taken by 23 out of 34 students (about 67%) in CS 5114. For 17 out of those 23 students (about 74%), this was the first time they were learning about NP-Completeness. 18 students (about 78.2% of the survey respondents) report to have looked into the tutorial and spent some time on it. Among these 18 students, the average time spent on the material as reported in the survey was 2.28 hours. Let us assume that the same proportion of the students i.e. 78.2% of the entire class actually looked at the tutorial. As stated earlier, the calculations based on our event logs shows total usage for the entire class during the 30 day period when NP-Completeness was taught, to be 50.66 hours which averages to 1.9 hours per student who used the tutorial. This is slightly lower than the 2.28 hours calculated from the survey. But, only 2/3rd of the class responded to the survey and as mentioned earlier our analysis provides stricter time estimates than actual.

For Questions 2, 4, 5, and 6 in the survey, we map the responses to weights 1-5. Doing so, the average level of difficulty of NP-Completeness as compared to other topics scores 3.95 out of 5. This supports our basic assumption that this is a difficult topic to learn. In Question 4, when asked to compare how helpful the tutorial is as relative to the textbook, among the 21 students who chose to answer the question, we find the mean to be around 2.65 on a scale of 1-5 which can be interpreted as the tutorial being about equally helpful as the textbook. In Question 5, where the student is asked about the number of slideshows they looked at, the mean is again 2.52, which says they worked through a few slideshows. This is consistent with our analysis of event logs. When asked the same about exercises, the mean is 1.69. This low mean is consistent with the our previous analysis of event logs.

15 students (about 65%) found the tutorial to be helpful to solve their homework. 4 out of these 15 students explicitly mention the visualization of reduction of 3-SAT to Hamiltonian Cycle in their feedback. This is consistent with the trend we see in our analysis of event logs where we see an increased use of this particular visualization around the homework deadline. 19 students (about 82%) report that they will be using this tutorial to prepare for their finals. However as can be seen from Figure 6.8, they do not actually use it much before the finals. The total time spent on the material by the entire class after April 10 is 1.8

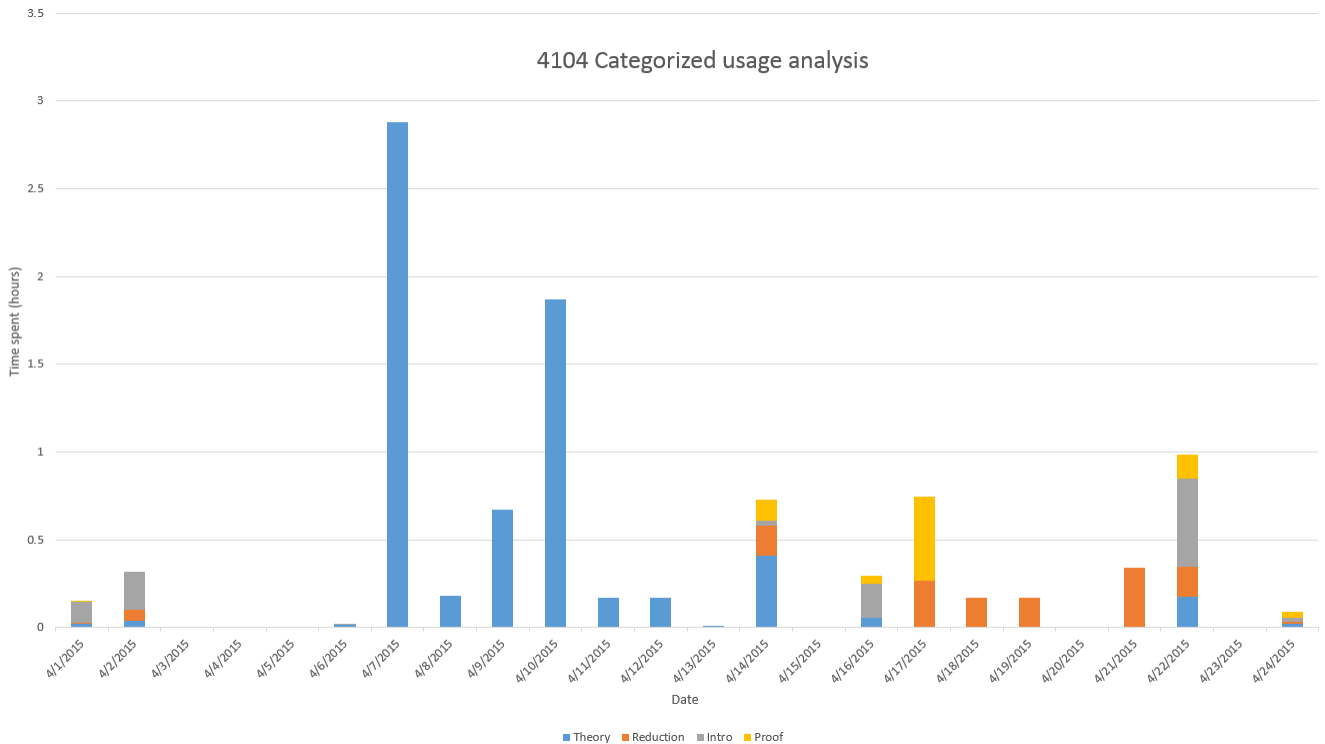


Figure 6.4: Categorized usage in CS 4104 for the entire class.

hours.

When asked to comment on their experience of the tutorial in Question 9, 18 students report this tutorial to be “good” and “helpful”. Only 1 student reported of the tutorial to be not useful, and 1 other who thinks it is a good tutorial in general but is just not of much use to the individual because of his prior experience in the subject.

6.3.2 CS 4104

In a class of 62 students, feedback was provided by 36 students (about 58%). 28 out of these 36 students (about 77.77%) report no prior experience with NP-Completeness. 14 out of 36 students (about 38.88% of the survey respondents) report to have spent some time on the tutorial. Among these 14 students, the average time spent on the material as reported by the survey is about 1.39 hours. Let us assume the same proportion of students (i.e. 38.78%) of the entire class actually used the tutorial. As we have reported earlier the total time spent on the tutorial by the entire class as calculated from our event logs, during the 24 days when NP-Completeness was taught is about 9.96 hours. This averages to about 24.78 minutes per student who used the tutorial. This is less than the 1.39 hours as reported by the survey. As stated earlier, our estimate is stricter than actual times recorded. This also indicates that the population of survey respondents includes most of the students who actually used the tutorial.

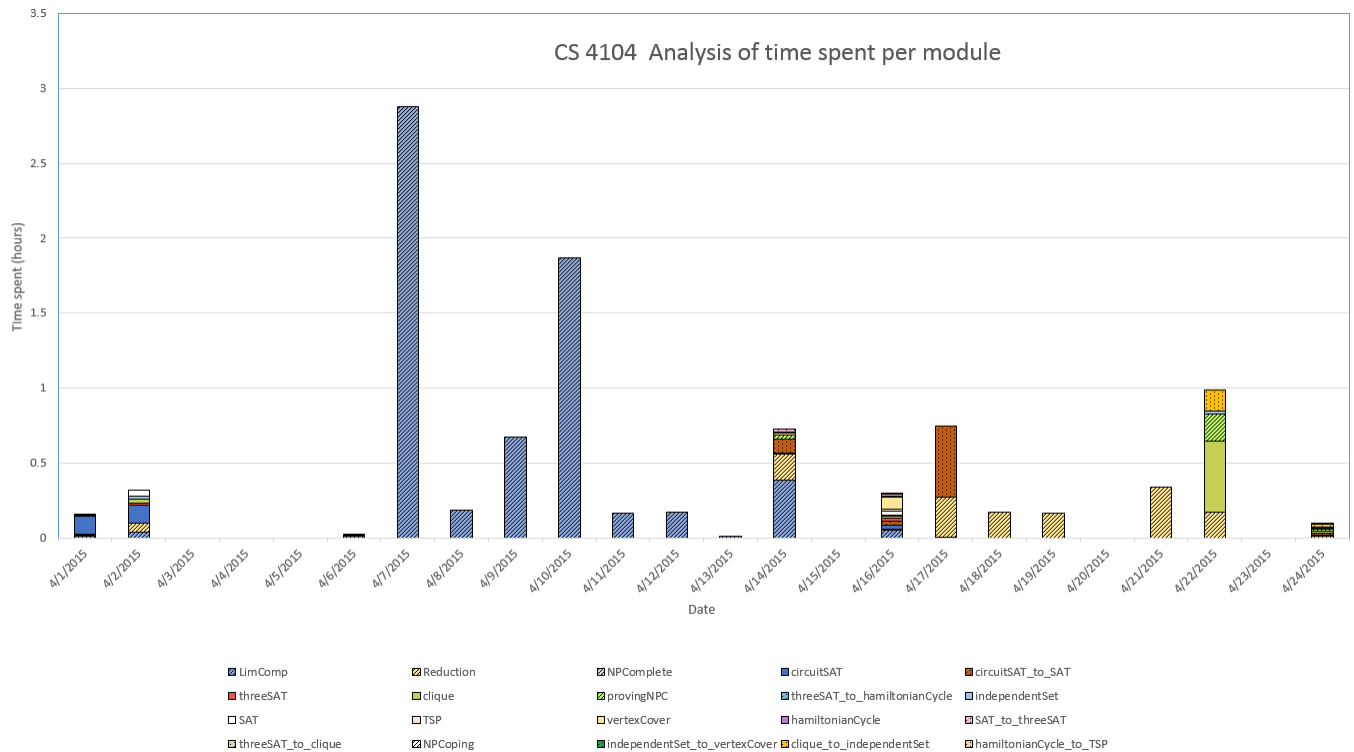


Figure 6.5: Usage per module in CS 4104 for the entire class.

We mapped the students' answers in Questions 2,4,5 and 6 to a scale of 1-5. The level of difficulty of NP-Completeness as compared to other topics had a mean of 3.83 out of 5 which shows that the students find it a somewhat difficult topic to learn. The time spent on the material as reported by the students on the tutorial averages to about 32 minutes. Question 4 asked students to compare the textbook to the tutorial. Only 24 out of 36 students answered the question, whose responses lead to a mean of 3.34. This can be interpreted as students who had an opinion about the relative usefulness of the tutorial as compared to the textbook think the tutorial is about as helpful. In Question 5, where the students were asked about their use of the slideshows, the mean response was around 1.69 which indicates low use. In Question 6 which reports the usage of practice exercises, the mean response is 1.38. Both of these means report low use, which is consistent with our results from the analysis of event logs.

Among the 36 students filling out the survey, only 7 students report that the tutorial was helpful in solving their homework. However 25 students report they will be using the tutorial and 6 others report they might potentially find studying from the tutorial helpful for finals. Interesting to note here, is that about 7 students report the reason for not using the tutorial to be lack of time and 4 students say they did not remember the tutorial existed while doing their homework. As we see from Figure 6.9, their usage of the tutorial spikes up before the finals.

When asked about their experience with the tutorial in Question 9, about 26 students (72%) provide positive feedback.

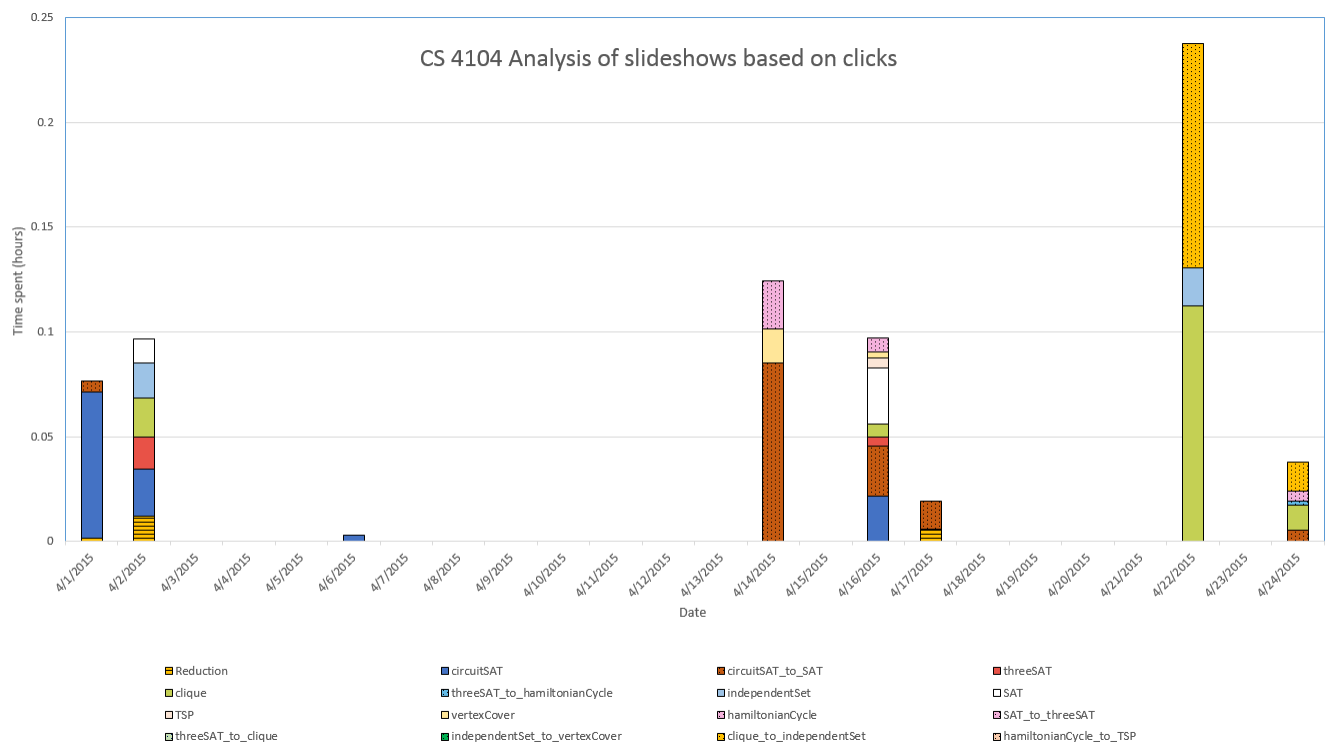


Figure 6.6: Usage of slideshows in CS 4104 by the entire class.

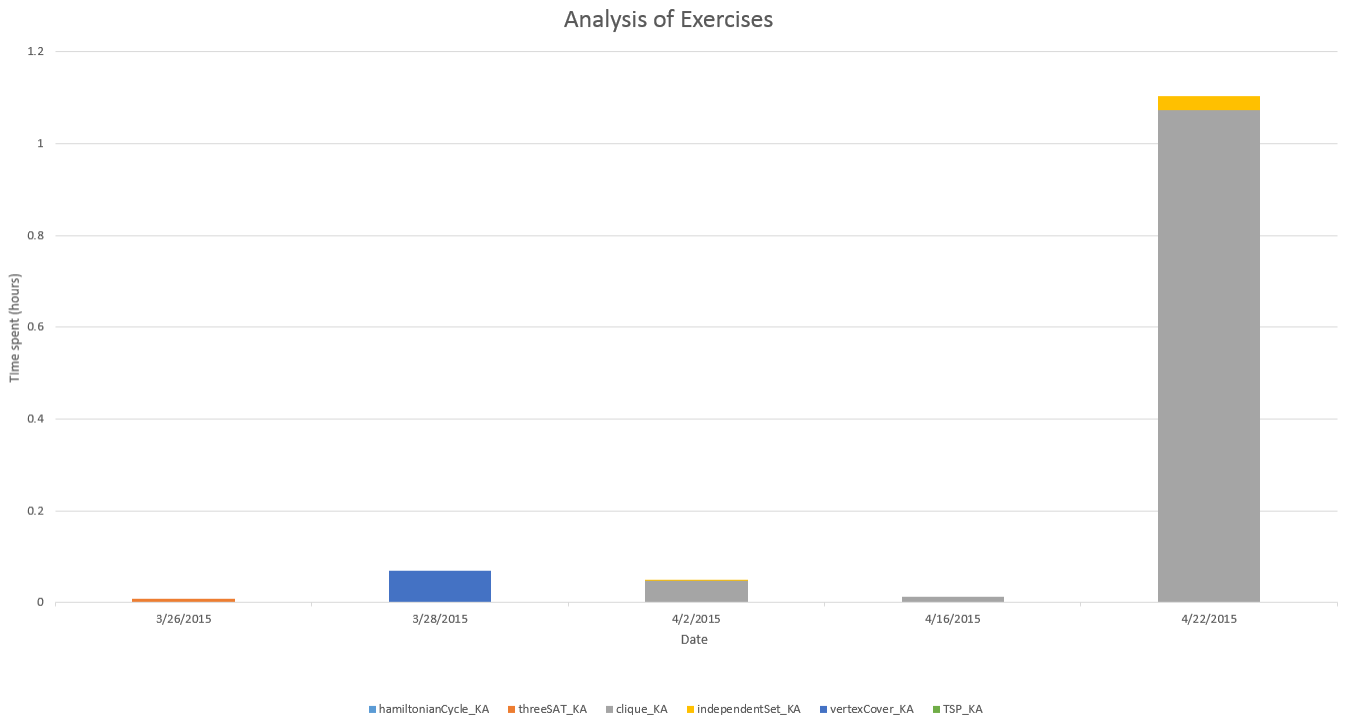


Figure 6.7: Cumulative usage of exercises.

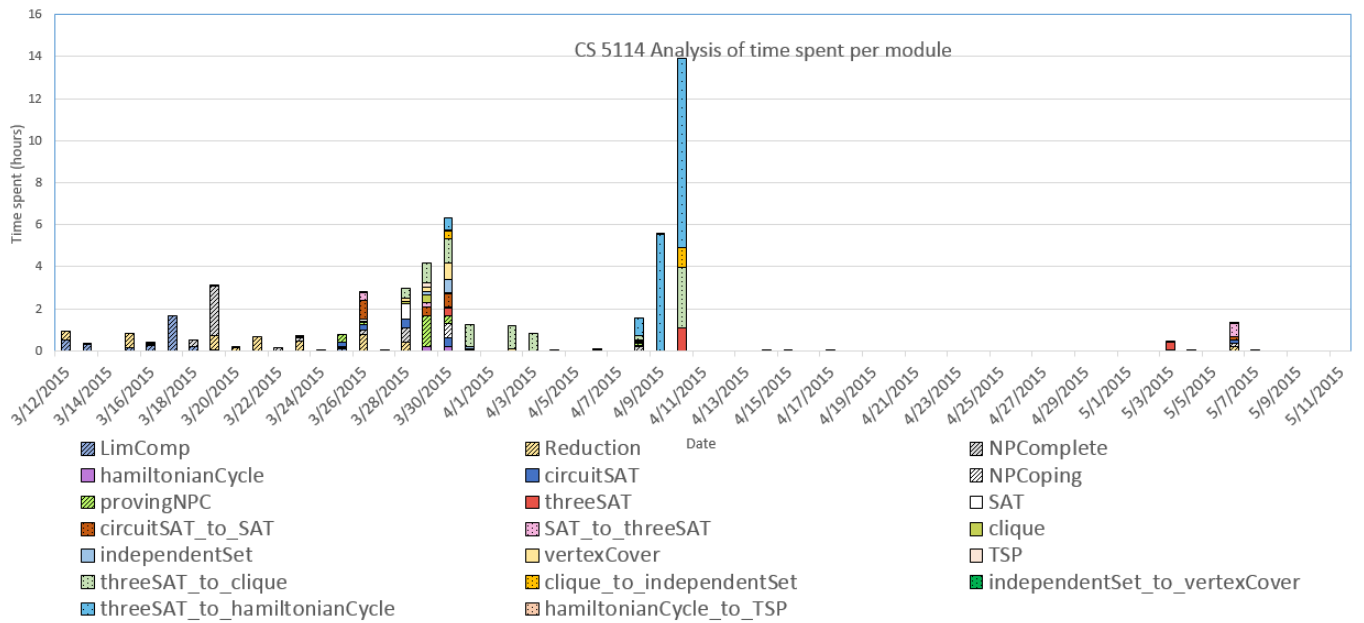


Figure 6.8: Usage of tutorial for CS 5114 over the semester.

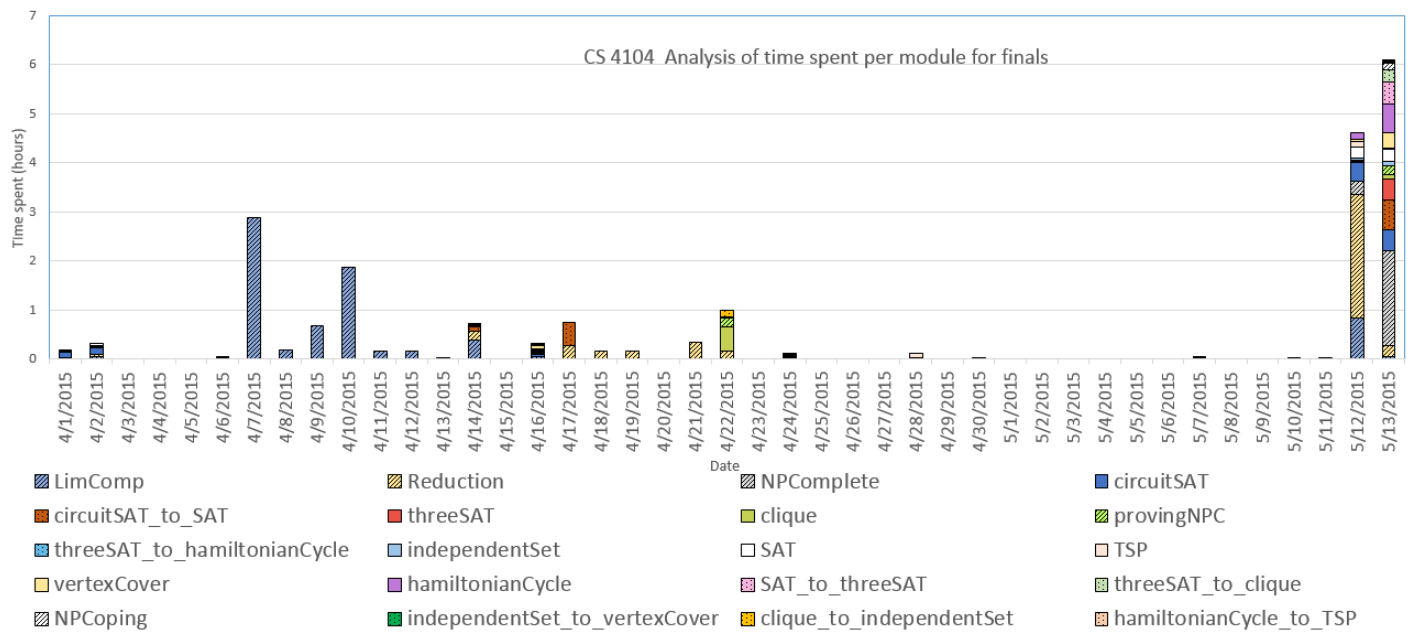


Figure 6.9: Usage of tutorial for CS 4104 over the semester.

Chapter 7

Conclusion and Future Work

As a part of our tutorial we created 18 visualizations in the form of slideshows and 6 practice exercises, apart from the textual content. Our visualizations cover 8 NP-Complete problems, and proofs of NP-Completeness for 7 problems. We introduced the tutorial as supplementary material in a graduate and an undergraduate class. We analyzed the interaction logs for the tutorials and collected feedback from the students. Our analysis suggests that a fair number of students look at the visualizations voluntarily, when they feel that doing so directly supports completing their assignments. But they do not typically try out exercises voluntarily.

In future, the tutorial needs to be evaluated for pedagogical effectiveness. This can be done by introducing the tutorial to different batches of students over multiple semesters and collecting their feedback and tracking their progress. The content can also be expanded to include more NP-Complete problems and corresponding exercises to solve their problem instances. Different instructors include different NP-Complete problems as a part of their syllabus. Hence, visualization more NP-Complete problems will facilitate us to selectively include visualizations as per the need of instructors and have the content customized for each course. The area of reductions can be further explored to include exercises on reductions. Our current module includes some textual content on coping with NP-Complete problems, but no visualizations. This can be extended to include approximation algorithms for NP-Complete problems and visualizations to portray them.

Bibliography

- [1] AlviE: An Algorithm Visualization JAVA Environment. [<http://alvie.algoritmica.org>].
- [2] Documentation for JSAV: JavaScript AV library. [<http://jsav.io/>].
- [3] Documentation for OpenDSA. [<http://opensda.readthedocs.org>].
- [4] GraphBench: Learning environment for NP-Completeness. [<http://www.swisseduc.ch/compscience/graphbench>].
- [5] Khan Academy. [<https://www.khanacademy.org>].
- [6] Source code for Khan academy exercises. [<https://github.com/Khan/khan-exercises>].
- [7] Swan: A Data Structure Visualization System. [<http://research.cs.vt.edu/AVresearch/Swan>].
- [8] Sorting out Sorting Video. [<https://www.youtube.com/watch?v=SJwEwA5gOkM>], 1981.
- [9] C. Armstrong, K. Asanovic, and R. F. Babiceanu. Computer Science Curricula, 2013.
- [10] R. Baecker. Sorting out sorting: A case study of software visualization for teaching computer science. *Software Visualization: Programming as a Multimedia Experience*, 1:369–381, 1998.
- [11] A. L. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. *Neural Networks*, 5(1):117–127, 1992.
- [12] M. A. Brändle. *GraphBench*. PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 16392, 2006.
- [13] M. H. Brown and R. Sedgewick. *A system for algorithm animation*, volume 18. ACM, 1984.
- [14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.
- [15] P. Crescenzi. Using AVs to explain NP-completeness. In *Proceedings of the fifteenth annual conference on Innovation and Technology in Computer Science Education*, pages 299–299, 2010.

- [16] J. T. Eyck and G. Sampath. Incorporating an interactive visualization of NP-Completeness proofs into a web-based learning environment. *Journal of Computing Sciences in Colleges*; 14, 4; 110-119, Annual CCSC northeastern conference, 1999.
- [17] V. Karavirta and C. A. Shaffer. JSAV: the JavaScript algorithm visualization library. In *Proceedings of the 18th ACM conference on Innovation and Technology in Computer Science Education*, pages 159–164, 2013.
- [18] A. J. Marrow. *The Practical Theorist: The Life and Work of Kurt Lewin*. Basic Books, New York, 1969.
- [19] C. H. Papadimitriou. NP-completeness: A retrospective. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, pages 2–6, 1997.
- [20] C. Pape. Teaching the reduction technique with interactive visualizations: report on a simple automatic tutor. *Interner Bericht. Universität Karlsruhe, Fakultät für Informatik*; 1998, 12.
- [21] C. Pape. Using Interactive Visualization for Teaching the Theory of NP-completeness. In *Proc. ED-MEDIA/ED-TELECOM*, pages 1070–1075, 1998.
- [22] C. A. Shaffer, M. L. Cooper, A. J. D. Alon, M. Akbar, M. Stewart, S. Ponce, and S. H. Edwards. Algorithm visualization: The state of the field. *ACM Transactions on Computing Education (TOCE)*, 10(3):9, 2010.
- [23] C. A. Shaffer, L. S. Heath, and J. Yang. Using the Swan data structure visualization system for computer science education. In *ACM SIGCSE Bulletin*, volume 28, pages 140–144, 1996.
- [24] C. A. Shaffer, V. Karavirta, A. Korhonen, and T. L. Naps. OpenDSA: beginning a community active-eBook project. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, pages 112–117, 2011.
- [25] C. Thompson. How Khan Academy is changing the rules of education. *Wired Magazine*, 126, 2011.
- [26] S. R. Vegdahl. Visualizing NP-completeness through circuit-based widgets. *Journal of Computing Sciences in Colleges*, 30(1):190–198, 2014.
- [27] B. J. Wadsworth. *Piaget’s theory of cognitive and affective development: Foundations of constructivism*. Longman Publishing, 1996.

Appendix A

Student Survey

1. Is this the first course that you have taken that presents the theory of NP-Completeness and NP-Completeness proofs? [Yes/No]
 - a. If "No", please briefly describe your prior exposure to NP-completeness theory.
2. How difficult do you find the topic of NP-Completeness in terms of understanding the concepts?
 - a. Not at all difficult compared to other topics in this course.
 - b. Somewhat less difficult compared to other topics in this course
 - c. It is of average difficulty compared to other topics in this course.
 - d. Somewhat more difficult compared to other topics in this course.
 - e. Much more difficult compared to other topics in this course.
3. How much time approximately did you spend on the online OpenDSA tutorial on reductions and NP-Completeness (in hours)?
4. Compared to the textbook that you used for this course, how helpful did you find the OpenDSA online tutorial for learning about NP-completeness?
 - a. The textbook was far more useful than the OpenDSA tutorial.
 - b. The textbook was somewhat more useful than the OpenDSA tutorial.
 - c. The textbook and the OpenDSA tutorial were about equally useful.
 - d. The textbook was somewhat less useful than the OpenDSA tutorial.
 - e. The textbook was far less useful than the OpenDSA tutorial.
5. Consider in particular the various slideshows that were presented as part of the online tutorial. Which best describes your use of these?
 - a. I did not look at any OpenDSA tutorial slideshows.
 - b. I briefly looked at a couple of slideshows, but did not really work through them.
 - c. I worked through a few of the slideshows.
 - d. I worked through half or more of the slideshows.
 - e. I worked through all or nearly all of the slideshows.

6. The online tutorial included a number of exercises that were meant to help you gain understanding of the material. Which best describes your use of these?
- a. I did not look at any OpenDSA tutorial exercises.
 - b. I briefly looked at a couple of exercises, but did not really work through them.
 - c. I worked through a few of the exercises.
 - d. I worked through half or more of the exercises.
 - e. I worked through all or nearly all of the exercises.
7. Do you think the OpenDSA tutorial helped you to do your homework on NP-Completeness? Please explain.
8. Do you think you will use the OpenDSA tutorial to prepare for your finals?
9. Please describe your overall opinion of the OpenDSA online tutorial for reduction and NP-completeness.

Appendix B

Distribution in problem instances generated for exercises

In order to inspect the kind of problem instances that get generated for each exercise in the tutorial, we load each exercise 1000 times. A new problem instance is generated every time an exercise is loaded. The data collected in this experiment has been tabularized below for each individual exercise.

For example, Table B.1 shows that the problem instance contained a 3-clique at 52 out of 1000 times.

Table B.1: The distribution in problem instances generated for the exercise on Clique Problem.

Size of clique	3	4	5	6
Number of occurrences	52	654	284	10

Table B.2: The distribution in problem instances generated for the exercise on Vertex Cover Problem.

Size of Vertex Cover	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
Number of occurrences	9	71	201	213	153	165	111	58	6	2	1	1	5	3	1

Table B.3: The distribution in problem instances generated for the exercise on Independent Set Problem.

Size of Independent Set	3	4	5	6	7	8	9	10	11	12	13
Number of occurrences	1	48	132	157	177	176	169	106	28	5	1

Table B.4: The distribution in problem instances generated for the exercise on Traveling Salesman Problem.

Length of shortest Hamiltonian Cycle	Number of occurrences
100, 104, 108, 112, 21, 22, 23, 90, 93, 95, 96, 98	1
101, 16, 20, 24, 31, 97	2
26, 28, 87, 94	3
29, 32, 85, 86, 89	4
25, 91	5
30, 84, 88	6
34, 79, 80	8
74, 75, 81, 83	9
69, 82	10
27, 73, 76, 78	11
36, 72	12
33, 54	13
35, 77	15
38, 62, 67	16
37, 41, 49, 65, 71	17
48, 66	18
39, 42	19
43, 46, 70	20
52, 53, 59, 60	21
40, 45, 58, 63, 68	23
51, 57	24
50	15
44, 64	26
55	29
56	30
47,61	31