In presenting this thesis in partial fulfillment of the requirements for an advanced degree at Idaho State University, I agree that the Library shall make it freely available for inspection. I further state that permission for extensive copying of my thesis for scholarly purposes may be granted by the Dean of the Graduate School, Dean of my academic division, or by the University Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature _____

Date _____

Redux: An Interactive, Dynamic Tool for Learning NP-completeness and Mapping Reductions

by

Kaden Marchetti

A thesis
submitted in partial fulfillment
of the requirements for the degree of
Master of Science in the Department of Computer Science
Idaho State University

To the Graduate Faculty:

The members of the committee appointed to examine the thesis of Kaden Marchetti find it satisfactory and recommend that it be accepted.

Paul Bodily,
Major Advisor

Farjana Eishita,
Committee Member

Irene van Woerden,
Graduate Faculty Representative

Table of Contents

Redux: An Interactive, Dynamic Tool for Learning NP-completeness and Mapping Reductions

Thesis Abstract – Idaho State University (2023)

Whereas interactive dynamic visualization tools have been successfully developed and used for teaching some topics in computational theory (CT), there remains a noticeable lack of such tools for teaching NP-completeness which continues to be widely taught using paper-and-pencil methods. Despite its important theoretical and practical value, NP-completeness—and mapping reductions in NP-completeness in particular—tends to be a challenging concept for CT students to understand. This thesis represents work on an open-source web app called Redux that provides a dynamic interactive user interface atop a practical knowledgebase of NP-complete problems, reductions, and solution algorithms. A key feature of the interface is the visualization of arbitrary problem instances, mapping reductions, solutions, and gadgets—including those reachable via transitivity. The web app is designed to make the knowledgebase extensible, allowing students to contribute and compare their own reductions and solutions to those already available. We describe Redux's development and highlight the first prototype of the tool. Two surveys were administered, with respondents overwhelmingly indicating that Redux helped them to better understand mapping reductions; that they would prefer using Redux to solving similar problems manually; and that Redux makes learning NP-complete reductions more enjoyable. Finally, we discuss how the principles of computational creativity and NP-completeness can be unified in extending many-one reducibility to a popular flood-fill game called KAMI. By rendering NP-complete problems as puzzles, we can unlock the KAMI player-base to crowd-source and solve real-world NP-complete problems. Such a concept serves as a reminder of the importance of NP-completeness in education. Redux is accessible online via `https://redux.portneuf.cose.isu.edu/`.

Chapter 1

Introduction

Computational theory (CT) in computer science (CS) education is important in helping students to understand the capabilities and restrictions of modern-day computers. Concepts such as automata theory, grammars, and NP-completeness play important practical roles in fields ranging from compilers, artificial intelligence, and optimization. The theoretical field is generally separated into two categories: the computability and complexity of problems in computer science. Computability teaches students the frameworks of computational devices such as automata (NFA, DFA, PDA) and Turing machines, showcasing the bounds of what modern day computers are and are not capable of doing. Meanwhile, complexity addresses the subset of problems that are determined to be computable and segregates them into complexity classes dependent on how quickly computers are able to find optimal solutions at scale. In particular, there are four complexity classes discussed throughout this thesis. They can be defined as such:

1. The Complexity Class P

   A language L is in P if and only if there exists a deterministic Turing machine M such that:

   - M runs for polynomial time on all inputs
   - For all x in L, M outputs 1
   - For all x not in L, M outputs 0

   Generally speaking, this is the set of all problems that can be **solved** in polynomial time by a Turing machine. A Turing machine is a mathematical model of computation that manipulates symbols on a strip of tape according to a table of rules. Despite the model's simplicity, it is capable of implementing any modern computer algorithm and

is computationally equivalent [70].

2. The Complexity Class NP

   A language L is in NP if and only if there exist polynomials p and q, and a deterministic Turing machine M such that:

   - For all x and y, the machine M runs in polynomial time on input (x,y).
   - For all x in L, there exists a string y of length q(x) such that M(x,y) = 1.
   - For all x not in L, M outputs 0

   Generally speaking, this is the set of all decision problems for which the answer, if yes, can be **verified** in polynomial time.

3. The Complexity Class NP-Hard

   A decision problem H is NP-Hard when for every problem L in NP:

   - There is a polynomial-time many-one reduction from L to H

   Generally speaking, this is the set of all problems for which there exists a polynomial many-one reduction from every NP problem. You can think of a mapping reduction, more simply, as a function f that turns one problem into another, such that their solution maps.

4. The Complexity Class NP-complete

   A decision problem H is NP-complete if and only if:

   - H is in NP
   - H is in NP-Hard

Generally speaking, this is the set of all decision problems that are verifiable in polynomial time (are in NP), and contain a polynomial many-one reduction from every NP problem (are in NP-Hard). You can think of a mapping reduction, more simply, as a function f that turns one problem into another, such that their solution maps [72].

Fig. 1.1 showcases a map of the aforementioned complexity classes under either assumption that P $\neq$ NP or P = NP, neither of which have been proven. Of these complexity classes, NP-complete, remains a unique and important concept in computer science and the focus of this thesis. Along with its formal definition, these are problems for which no known algorithm exists for optimally solving the problem in a tractable time frame. Many of the the complex problems facing the world today have been formally classified as NP-complete.

NP-complete problems play a significant role in optimization and decision-making across a spectrum of disciplines: genome assembly; scheduling problems; nuclear core fuel reload optimization [32]; route planning [75]; and the list grows ever longer. As a result, there exists a widespread need for computer scientists equipped with the knowledge and skills needed to address these problems. This includes the ability to recognize (i.e., prove) a problem as NP-complete; leverage existing heuristic solutions to solve novel instances of NP-complete problems; and develop novel heuristic solutions that are both time-efficient and that provide quasi-optimal results.

While NP-completeness and theoretical computer science is the backbone of problems we solve in our field, it also happens to be incredibly difficult to teach and instruct. Significant shortcomings exist in the effectiveness with which the subject is taught in CS curricula with most instructors persisting in teaching the topic using rote paper-and-pencil methods and others not covering the topic at all [43]. Paper-and-pencil methods limit instruction to simple instances of these NP-complete problems and make more complex reductions very time-consuming if not outright impractical. Indeed, research suggests that a minority of

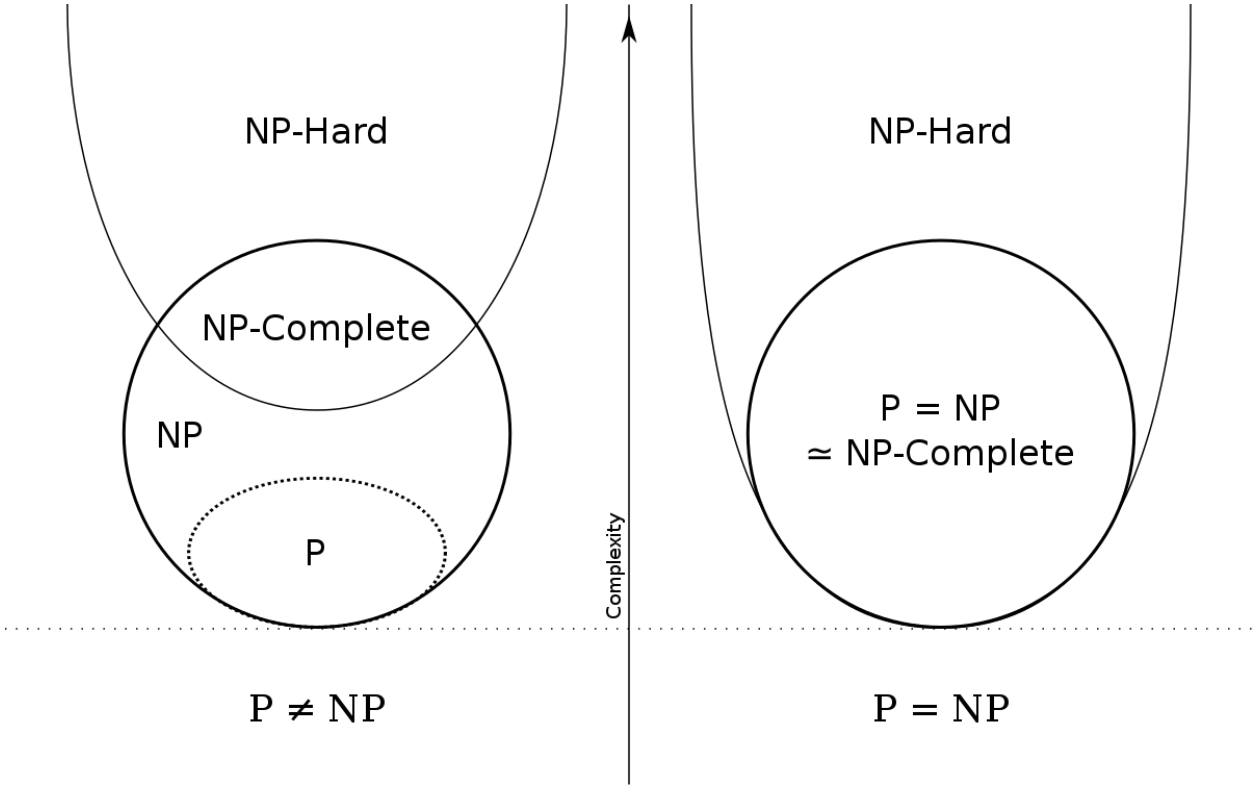


Figure 1.1: An Euler diagram of complexity classes for P, NP, and NP-complete. The right side is valid under the assumption that P equals NP while the left relies on the assumption that P does not equal NP. The vertically of each class correlates to its complexity. Most computer scientists believe P $\neq$ NP, and much of the theory and practical application in academia assume they are not equal. In any case, many significant real-world problems are demonstrably NP-complete, and more effective pedagogical methods are needed to equip students with practical skills for recognizing and addressing these real-world NP-complete problems. Figure is taken from Wikipedia.

undergraduates advance to their careers with the necessary skills to be able to apply this theory in their field [7].

In particular, students struggle with reduction theory, a concept fundamental to NP-completeness. In layperson's terms, reduction theory deals with polynomial mapping functions that can convert an instance of problem A into an instance of problem B. These are also called many-one reductions and are defined mathematically as follows:

- Suppose A and B are formal languages over the alphabets Σ and Γ, respectively. A many-one reduction from A to B is a total computable function $f\colon \Sigma \to \Gamma$ that has the property that each word w is in A if and only if *f(w)* is in B. If such a function f exists, we say that A is many-one reducible (or mapping reducible) to B. [57]

Every NP-complete problem, also being NP-Hard, is many-one reducible to every other NP-complete problem. Indeed, this feature is what separates NP-complete problems, from their parent class, NP. Consider for example, the infamous NP-complete traveling salesperson problem (i.e., "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"). This problem can be reduced to instances of graph coloring problems, boolean satisfiability problems, set cover problems and virtually an infinite number of other problems that routinely crop up in real-world contexts. An efficient, polynomial solution to any of these problems stands to benefit societies and organizations in untold ways [61].

In order to address the existing gap in theoretical CS education, additional teaching methods need to be explored. Several successful dynamic interactive visualization tools have been developed to teach other topics in computational theory. Such tools have a proven track record for helping students develop theoretical and practical mastery [18]. The most notable of these tools is JFLAP [59], an automata visualization simulator that student surveys indicate makes learning such concepts easier and more enjoyable [60]. Despite the apparent success of these tools in other areas, a systematic literature review (described in Chapter 2) suggests significantly less attention has been devoted to developing tools to teach the CT topic of NP-completeness. There exists a demonstrated need for dynamic interactive visualization tools for teaching NP-completeness.

The work presented in this thesis outlines two years of work in developing a novel web-based visualization tool called Redux. In particular, the tool provides examples of several canonical NP-complete problems; allows the user to define custom instances of these problems;

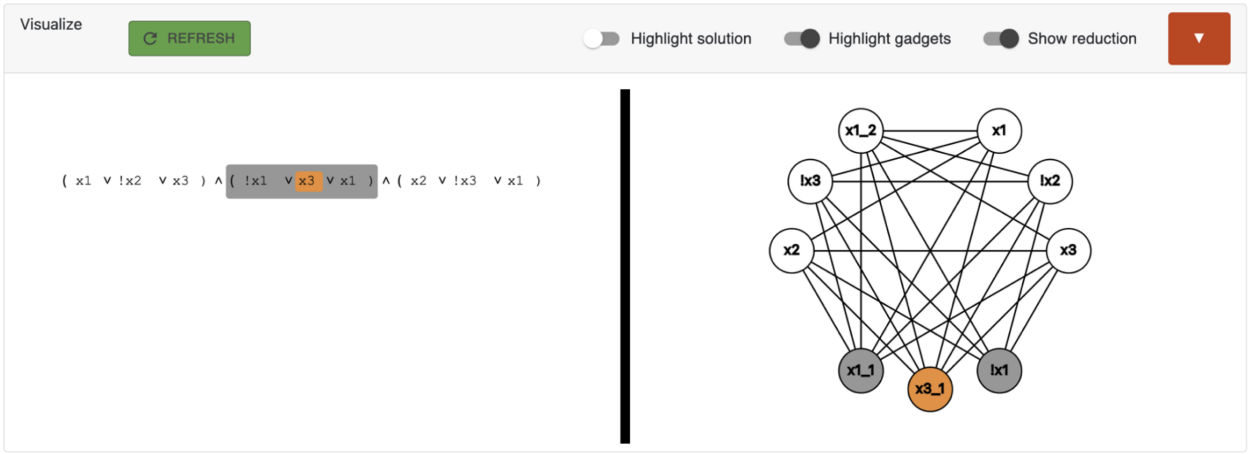


Figure 1.2: *Dynamic, interactive visualization of NP-complete mapping reductions.* One of the key features of Redux is visualization of mapping reductions. Such mapping reductions prove polynomial-time equivalence between problems and allow solutions to one problem to solve the other. Shown here is the result of reducing an instance of *3SAT* (left) to an equivalent *CLIQUE* instance (right). Redux allows users to use their mouse to highlight elements of one instance (e.g., the Boolean literal in orange on the left) and see its analogous element highlighted in the other instance (the orange node on the right). These analogous elements are called *gadgets*, and seeing how they relate is key to understanding how to identify and implement reductions.

and visualizes these instances, their mapping reductions to other NP-complete problems, and their possible solutions (see Figure 1.2). The interactive aspect of the tool enables the user to examine how particular reduction algorithms work by highlighting the analogous substructures, or *gadgets*, between the visualized problem instance and the reduced problem instance. The interface lies atop an extensible knowledgebase and provides students the educational opportunity of comparing their own reduction implementations and heuristic solution algorithms against those already present.

To justify the need for such a tool, Chapter 2 of this thesis outlines the results of a systematic literature review of existing visualization tools designed for teaching computational theory. This review not only outlines existing tools designed to teach NP-completeness, but also other fields in CT such as automata, Turing machines, and finite grammars. Of the 2492 papers examined, we found many tools that have been effective in helping students gain

practical mastery through simulation of state automata [52, 59, 65]; context-free grammars [1, 59]; pushdown automata [24, 59]; pumping lemmas [13, 59]; and Turing machines [26, 59]. Yet there remains an apparent lack of tools for dynamic and/or interactive visualization of topics related to NP-completeness, including mapping reductions between NP-complete problems and the application of heuristic solution algorithms to NP-complete problem instances. The most notable existing work on visualizing concepts relating to NP-completeness is demonstrated in the OpenDSA framework [45]. OpenDSA offers an effective introduction to NP-complete problems and mapping reducibility. The problems demonstrated in this framework are highly limited, however, and are not designed to allow for students to interact with or modify the example instances. Other contributions of theoretical NP-complete visualization tools appear to have been abandoned or have shifted to no longer center on NP-completeness [17]. For example, AlViE is an algorithm visualization environment dedicated to helping students better conceptualize concepts in typical algorithms courses; however, the application appears to have been moved and any mention of it appears to be confined to algorithmic visualization [17]. This conclusion justifies the need for a more comprehensive and extendable tool that educators can use to teach these complex subjects.

In light of the findings of the systematic literature review, we designed and implemented a prototype to visualize NP-complete problems and their reductions. Chapter 3 focuses on the implementation of a dynamic visualization of the reduction from *3SAT* to *CLIQUE* as defined by Michael Sipser, in his book "Introduction to the Theory of Computation" [66]. Included is a detailed discussion of *3SAT*, *CLIQUE*, and a polynomial mapping function f that maps the problems. This chapter was published in the proceedings of the 2022 Intermountain Engineering, Technology and Computing (i-ETC) conference and features screenshots of this early prototype (e.g., see Figure 1.3).

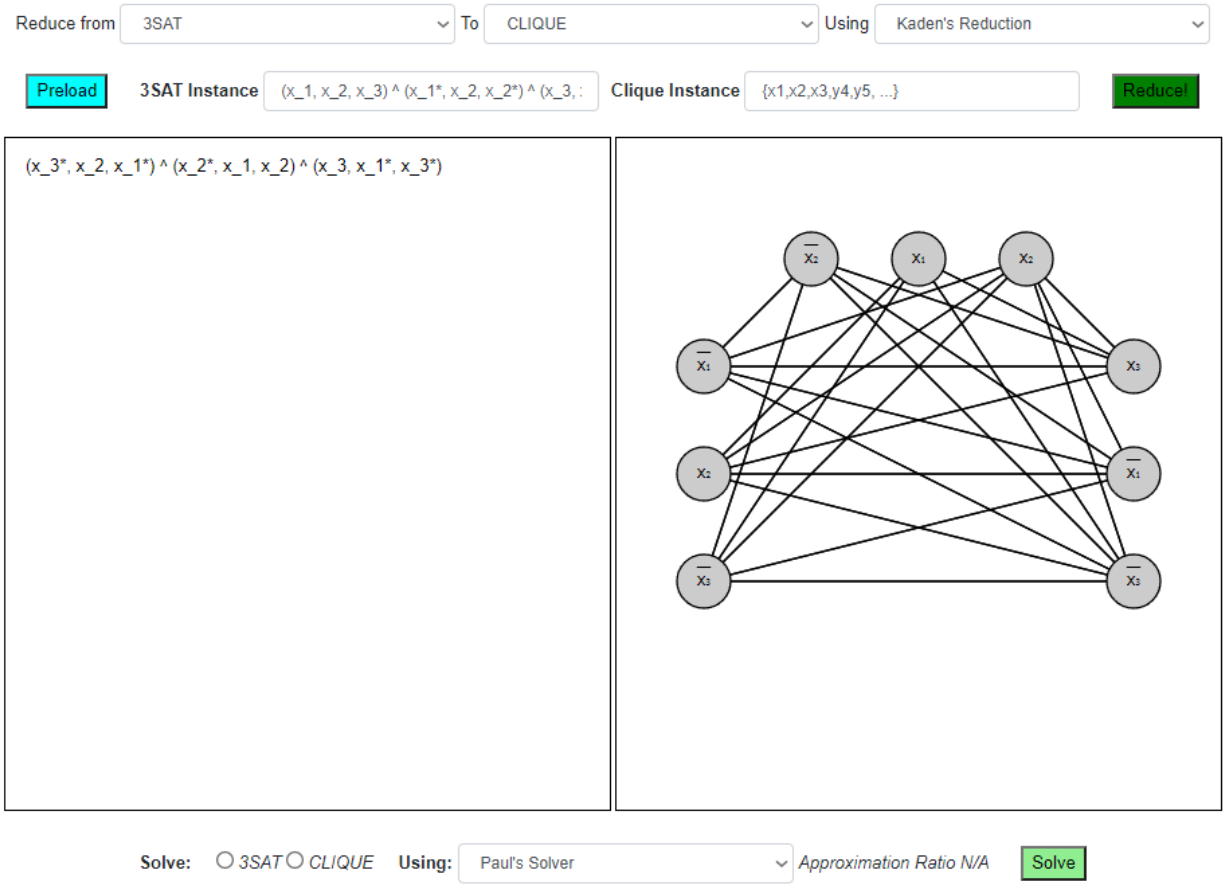


Figure 1.3: A Sample output of the visualization application detailed in Chapter 3. At the top, the user can select a problem to reduce from, reduce to, and a polynomial mapping function that facilitates the reduction. Users can manually enter a problem instance or pre-load an existing configuration before reducing. The CLIQUE graph on the right is dynamically generated from our polynomial mapping function $f(\phi)$ where $\phi = (\overline{x_3} \vee x_2 \vee \overline{x_1}) \wedge (\overline{x_2} \vee x_1 \vee x_2) \wedge (x_3 \vee \overline{x_1} \vee \overline{x_3})$. The 3SAT instance is editable and results in a dynamic, interactive visualization of CLIQUE. At the bottom, the user has access to multiple algorithms to solve either the original 3SAT instance or the dynamically-generated CLIQUE instance.

A subsequent version of Redux is defined in Chapter 4. We discuss the RESTful API framework used to build Redux, allowing it to function as a dual-use application where

students can utilize the website for pedagogical visualizations while researchers can utilize the API for meta-level research. The chapter outlines many novel features that Redux provides including dynamic and interactive gadget highlighting (see Figure 1.2), solution visualizations, and the inclusion of transitive reductions. The inclusion of transitive reductions, in particular, has the potential to be a powerful pedagogical tool.

Allowing students to use the solvers of problems many reductions away to solve the original problem is a fundamental concept in many-one reductions that is often overlooked and forgotten due to its complexity.

Included in the chapter are two surveys designed to assess Redux as a lecture supplement. While an additional, much larger survey still needs to be conducted, these preliminary assessments point to Redux being potentially valuable in teaching students NP-completeness.

Finally, Chapter 5 introduces a popular flood-fill game called KAMI, A one-player combinatorial puzzle game. As an NP-complete problem itself [11], KAMI is reducible to (and from) other NP-complete problems; but unlike many NP-complete problem representations, thousands of mobile game players willingly devote countless hours and money in the pursuit of designing and solving KAMI puzzles. The chapter discusses how the principles of computational creativity (CC) can be applied in the context of rendering arbitrary NP-complete problems as KAMI puzzles in order to leverage the massive KAMI crowd-sourcing platform to effectively solve real-world problems. Along with being the earliest publication related to Redux, and winning "Best Research Paper: Computing" at i-ETC, this chapter showcases the importance of teaching, and understanding, NP-completeness.

Chapter 2

Related Work - A Systematic Literature Review

This paper will be submitted to ACM SIGCSE 2024 for publication

2.1 Introduction

Computational theory (CT) in computer science (CS) education is important in helping students to understand the capabilities and restrictions of modern-day computers. Concepts such as automata theory, grammars, and NP-completeness play important practical roles in fields ranging from compilers, to artificial intelligence, to optimization. Despite this importance, however, CT remains one of the most difficult subjects for students to learn and teach [18, 23, 43]. Perhaps owing to the theoretical nature of the subject matter, instructors frequently fall back on traditional paper-and-pencil methods to teach CT subjects with the result that many students struggle to grasp the concepts well enough to develop practical mastery [60].

Recent decades have demonstrated the power of using visualization tools for pedagogical purposes (e.g., see Figure 2.1). The term algorithm visualization (AV) tools has evolved to refer specifically to digital pedagogical resources designed to help students understand algorithms and other concepts in CS [51], with some calling for there to be more effort made to apply these same strategies in teaching and learning CT [23].

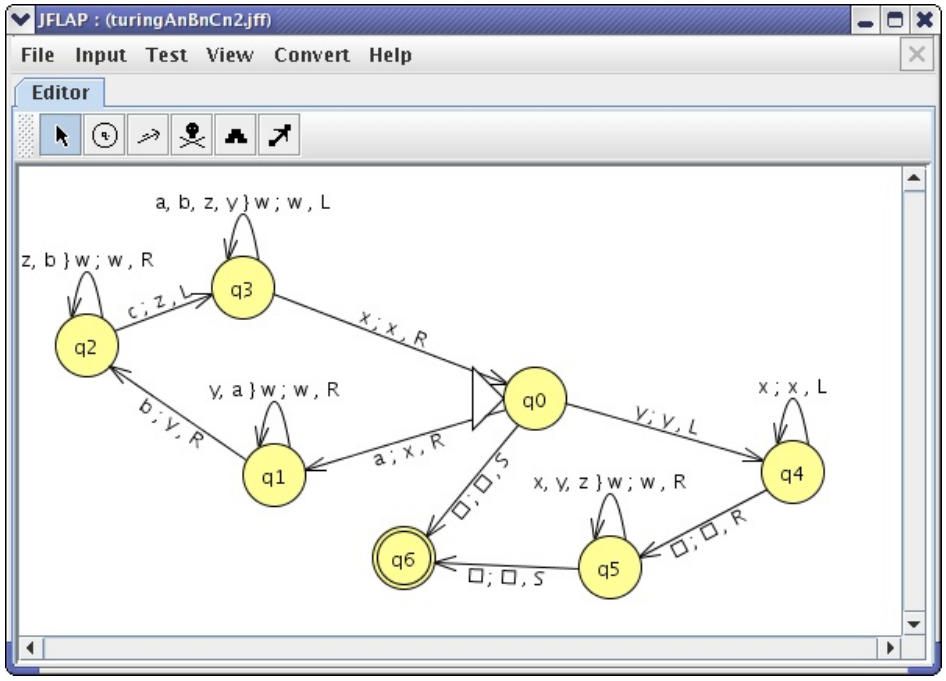


Figure 2.1: *Algorithm visualization.* A screenshot of a Turing machine simulation in JFLAP, a popular software package for visualization of computational theory concepts. Interactive dynamic tools like JFLAP help to make difficult concepts more accessible to computer science students, particularly in the domain of computational theory.

In the interest of identifying what AV tools exist for CT and what areas of CT may yet be lacking in AV, we carried out a systematic literature review of 2492 publications following the method described by Petersen et al. [56]. Our findings demonstrate that while some CT topics (e.g., automata, grammars) have been heavily featured, there are other CT topics (e.g., NP-completeness, proofs) for which AV solutions are virtually no presence. We present these findings with the hope that identifying, compiling, and categorizing existing AV solutions in computational theory will help instructors who utilize these tools to gain confidence and success in teaching students to understand and apply concepts in CT. Furthermore, identifying which CT concepts are underrepresented in the domain of AV will serve to provide direction for future research and development.

Table 2.1: Research Questions and Rationale

	Research Question	Rationale
Q1	What AV solutions exist in the literature that focus on computational theory education?	We want to identify and isolate the body of AV literature that pertains to computational theory and define what visualizations they use.
Q2	Of the AV literature that pertains to computational theory, what conferences and journals are represented?	We want to inform researchers where these papers are being published to direct future research and development.
Q3	What subjects in computational theory see the most publications and are there underrepresented subjects?	We want to help direct new researchers and developers towards areas of need and establish what already exists in the space.
Q4	Of the visualization publications that exist, which are dynamic and interactive?	We want to establish a distinction between studies that have interactive and dynamic visualization tools rather than static instances.

2.2 Methods

A systemic literature review is designed to provide a holistic view of available literature pertaining to a topic of interest. Our review follows the popular framework defined by Petersen et al. [56]. Planning of the literature review began with the formulation of research questions and their rationale which are shown in Table 2.1. From these research questions were derived a set of search terms: "Visualizing reductions in computer science", "Visualizing NP-completeness", "AVs in computational theory", "Computational theory education through visualization", "Animations in computational theory education", "Visualization in education of theoretical computer science". In the absence of a formal database for computational theory literature, we followed Wohlin's suggestion to use Google Scholar to find papers from multiple publishers [78].

As a starting set, the first fifty results from each search string were collected for evaluation. We manually screened the set using the following inclusion and exclusion criteria:

- **Inclusion Criteria**
  - The paper must be peer-reviewed or be published by an accredited journal.

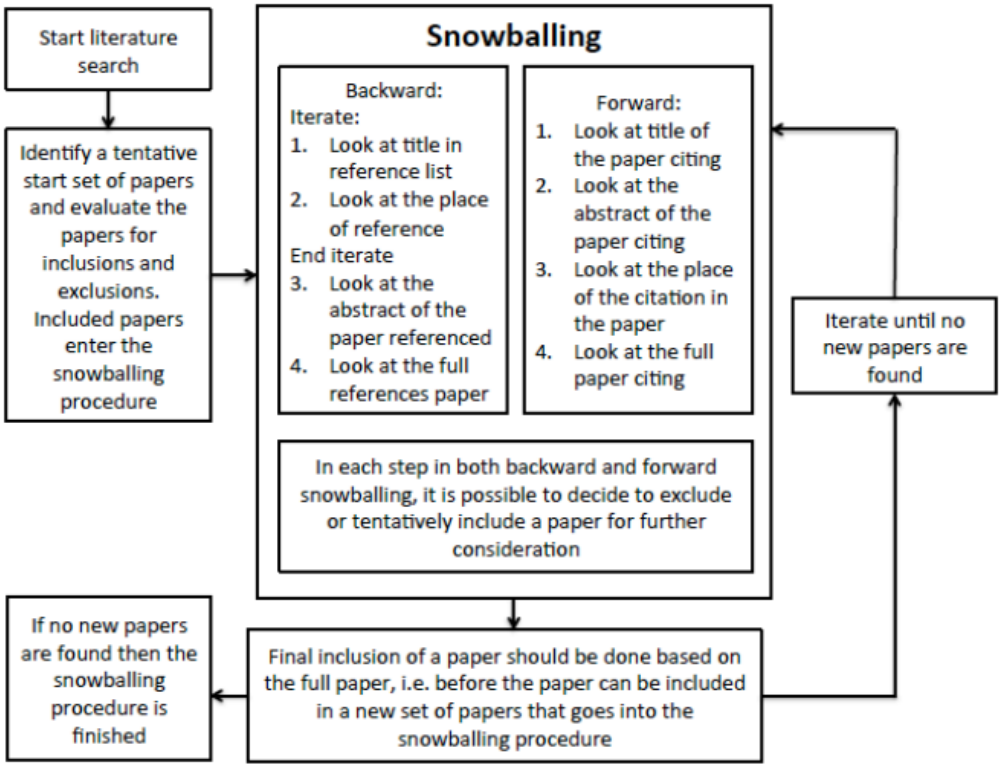


Figure 2.2: *Snowballing in literature reviews.* Beginning from a start set of papers, backward snowballing considers papers cited by those in the set. Conversely, forward snowballing considers papers that cite those in the set. The process is iterative. Figure from Wohlin [78].

  - The paper must have an English translation available.
  - The paper must focus on computational theory education.
  - The paper must mention a keyword identified as relating to computational theory education through algorithm visualization.
  - The paper must be published between 1996 and 2021.

- **Exclusion Criteria**

  - The paper does not mention AVs or other visualizations as a method of computational theory education.
  - The paper does not establish education as the primary purpose of the AV or visualization.

These criteria are first applied to a study's title and abstract; then (for those that survive) to the introduction, conclusion, and methods section. Papers that survive the screening process are included as *primary studies*.

Once an initial set of primary studies is obtained, we expanded this set using forward and backward snowballing. Figure 2.2 summarizes the snowballing process as set out by Wohlin [78]. Four snowballing iterations were performed in this review. A complete summary of our literature review can be seen in Figure 2.3.

2.3 Results

A total of 2492 papers were examined in this study yielding 57 primary studies. Within the set of primary studies, several dozen AV platforms are represented. A few AV platforms stood out (in terms of publication counts and external references), namely White [77] and Rodger [59]. We discuss these more in depth below. Figure 2.4 provides a high-level summary of the systematic literature review results categorizing all 57 primary studies by AV platform, CT topic, and interactibility.

There are some specific conclusions that we can draw from this study before diving into the research questions. First, while there is a respectable amount of interactive, visualization tools for computational theory, these are primarily focused on automata, grammars, and Turing machine visualizations. Indeed, there are only three papers in the primary studies that focus on NP-completeness or reducibility and none of them are interactive and allow for dynamic visualization generation. Furthermore, many of these visualization tools are no longer supported. For example, Crescenzi et al's. [17] paper outline an addition to a popular AV visualization software called AlViE; however, this software has not been updated for over a decade and both the website and application download are no longer operable. While applications such as JFlap are still active, they have not been extended to include reduction

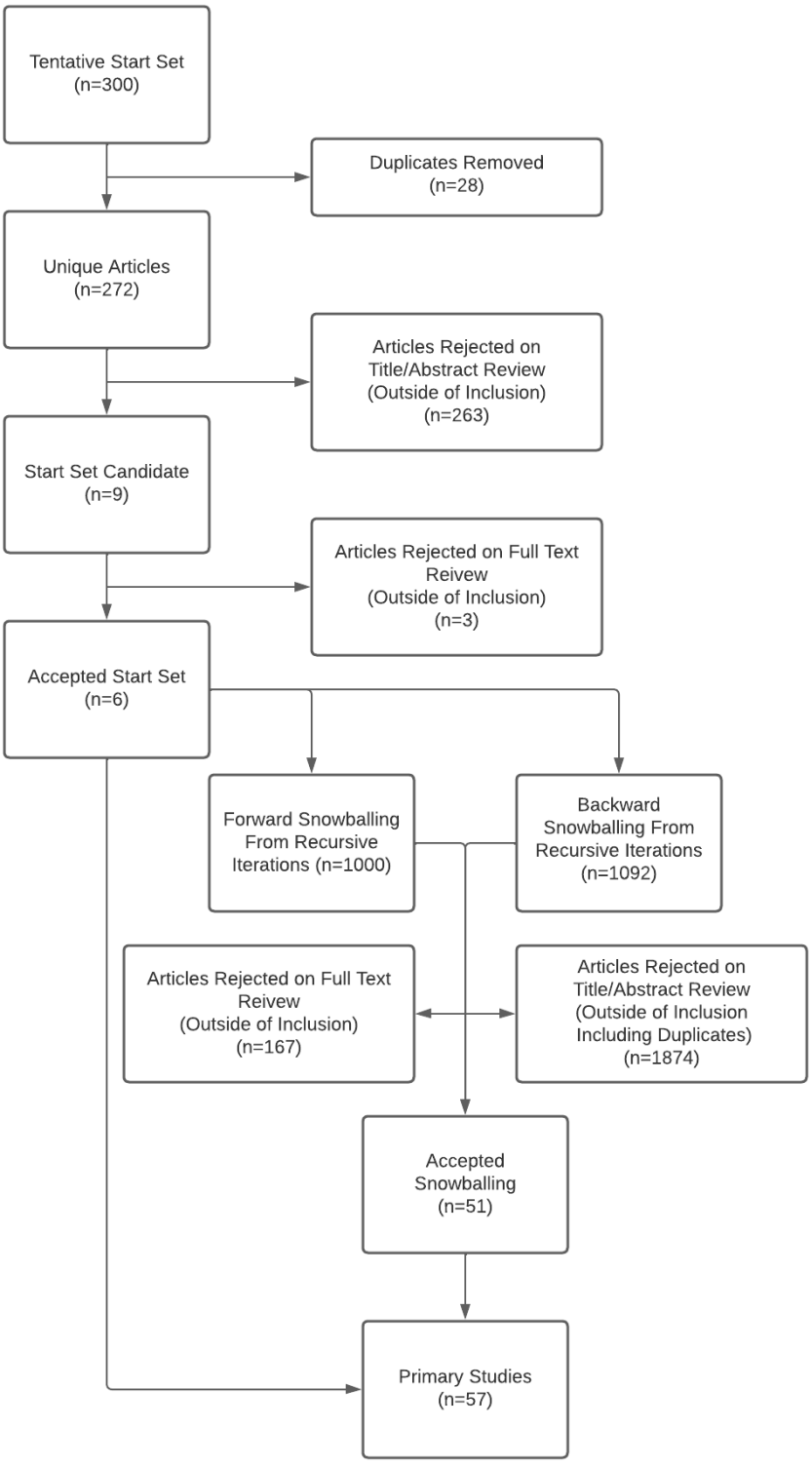


Figure 2.3: *Systematic literature review overview.* An overview of our systematic literature review performed on interactive visualization technology for computational theory. A total of 2492 papers are examined using rules prescribed by Petersen et al. [56].

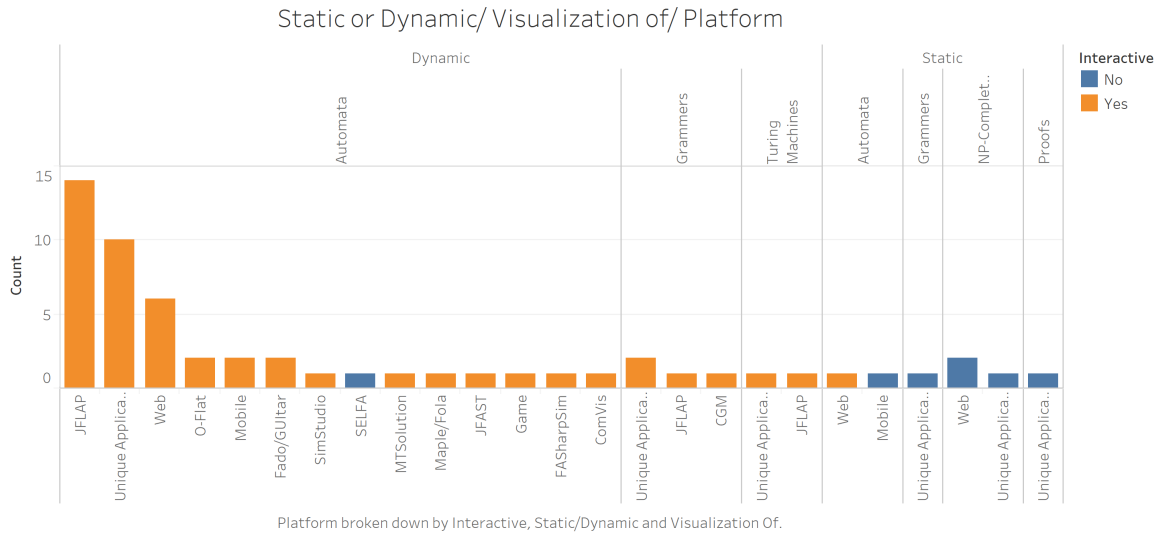


Figure 2.4: *Primary studies by platform and topic.* Categorization of primary studies resulting from the systematic literature review by AV platform, CT topic, and interactibility.

theory and NP-completeness but offer fantastic educational visualization tools for automata, grammars, and Turing machines.

Indeed, there is a plethora of visualization tools dedicated to particular subjects in computational theory. Furthermore, these tools appear to replicate the pedagogical success algorithms courses have seen with visualization technology. Students that used JFLAP, for example, felt they understood theoretical concepts better than their peers [60].

Despite this success, our results showcase a problematic trend. Publications pertaining to visualization technology for computational theory education have been in decline, particularly in the last five years. Figure 2.5 showcases this slowdown and demonstrates a 50 percent reduction in publications in the last decade. Given the success of visualization technology in teaching particular theoretical subjects, this slowdown represents opportunities for future research to reverse course and create tools in under-represented subjects in theory education.

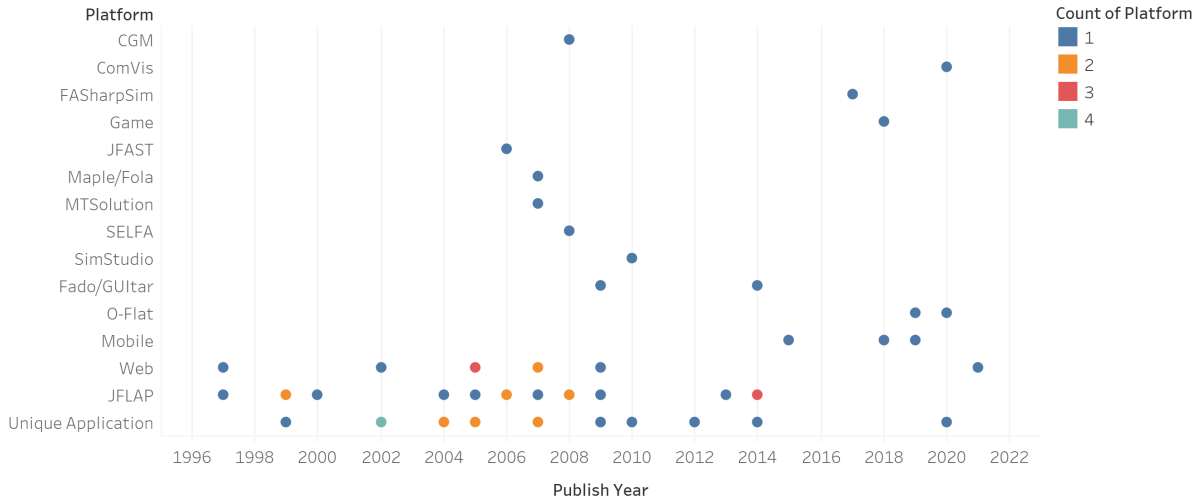


Figure 2.5: *Publication years for primary studies.* Primary studies are grouped by publication year and platform. The plot suggests a possible downward trend in the number of CT-related AV publications starting around 2010.

2.3.1 What AV solutions exist in the literature that focus on computational theory education?

As previously mentioned, the most notable contributions in computational theory visualizations are part of tools such as JFlap [59], JFast [77], and AlViE [17]. JFlap offers educators and students a Java widget for creating, manipulating, and experimenting with theoretical concepts, namely automata and Turing machines. Many researchers have contributed to its development and many of our primary studies outline contributions to JFlap's automata simulator [28, 60, 62, 63]. Despite JFlap's focus on automata, there are contributions that add other theoretical concepts to the application such as parsing algorithms, multi-tape Turing machines, and grammar transformations [2, 8, 36]. JFast, while offering an independent automata simulator, has not seen additional publications since its inception. AlViE has also not been updated for over a decade and both the website and application download are no longer operable. Though it was primarily developed for algorithm visualization, it also contained a number of NP-completeness proofs [17]. Overall, the vast majority of the

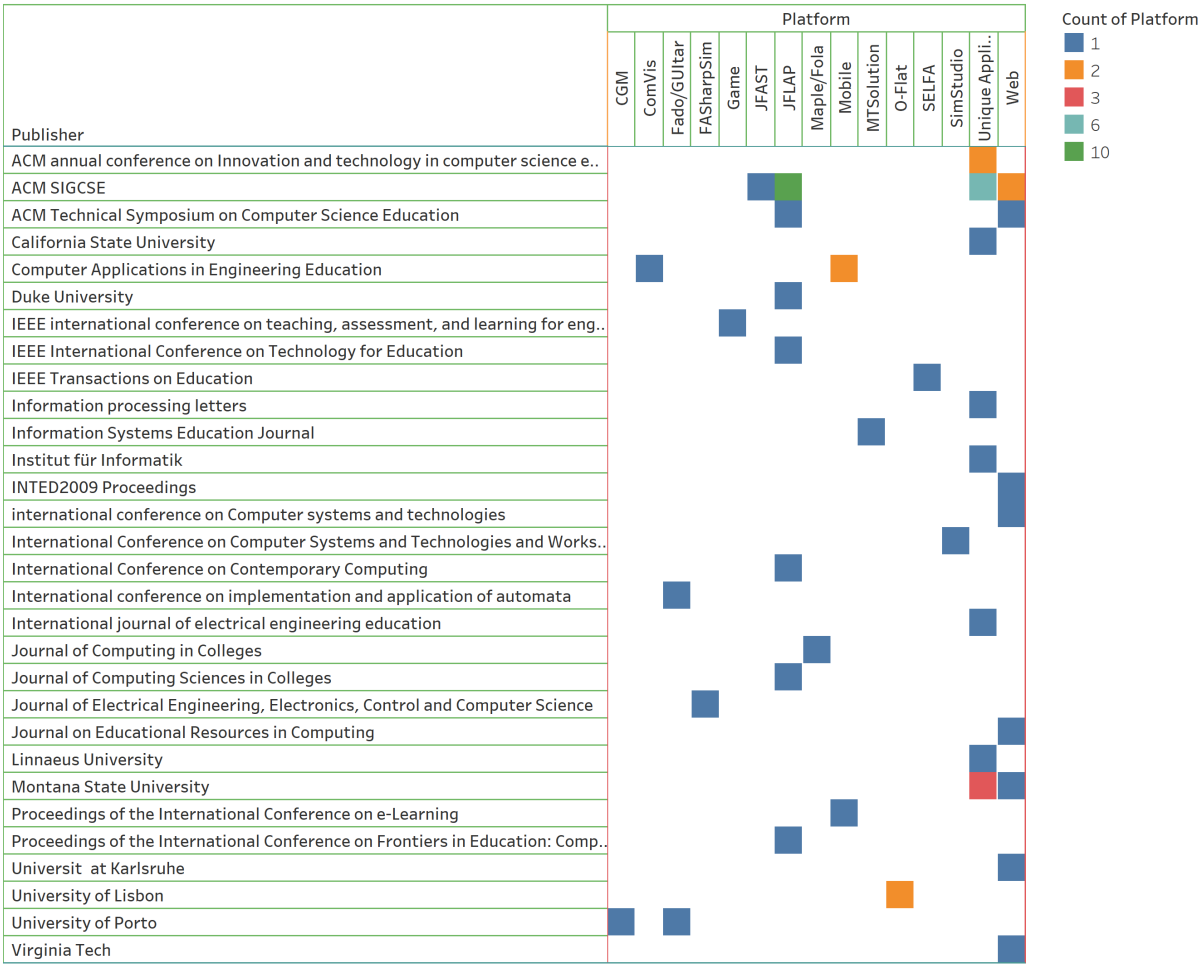
literature presents dynamic, interactive visualization tools for automata, grammars, and Turing machines. Figure 2.4 visualizes the categorization of systems by CT subject.

The study also suggests some notable trends in AV development. Some authors have attempted to bridge accessibility gaps by moving to mobile [10, 65, 68] and web-based [9, 12, 19, 30, 45, 48, 49, 73] platforms. While this shift continues to feature primarily automaton simulators, some novel approaches have emerged. One study, for example, incorporates educational content in a virtual reality game designed to teach students about finite state machines through treasure hunts [20].

Many unique applications exist outside of the most popular platforms. Most of these applications are also specific to automata [24, 29, 31, 41, 46, 58, 80] and/or grammar [13, 50, 74] visualization, and many are no longer being maintained or are not targeted towards CS audiences [13, 25, 38, 52, 64, 69, 74]. This diversity, while offering students many different platforms, may contribute to the range of abandoned applications. In one of the most notable existing literature reviews of current AV solutions, Shaffer et al. [61] note that the concerns over the field of AVs "could be mitigated by building community and improving communication among AV users and developers". In many ways, JFlap represents the best attempt at accomplishing this and demonstrates this with its continued popularity among CT educators.

2.3.2 Of the AV literature that pertains to computational theory, what conferences and journals are represented?

Figure 2.6 outlines which conferences generally publish new tools and methods in educational computational theory visualizations. Several of the AV tools represented were developed and published as theses or dissertations.



Count of Platform (color) broken down by Platform vs. Publisher.

Figure 2.6: *AV publication venues.* A summary of conferences in which primary studies relating to computational theory AV are published. Where the publisher is a university, the primary study was a thesis or dissertation.

The publication venues represented among our primary studies—SIGSCE foremost among them—represent the ideal communities in which conversation and collaboration focused on AV development might find a natural following.

2.3.3 What subjects in computational theory see the most publications and are there underrepresented subjects?

The vast majority of AV solutions for computational theory are designed for automata, finite state machine simulations, and formal languages. Turing machines, both single and multi-tape, are also represented in the most popular applications [60]. This trend has stayed consistent as every publication from the primary set in the last seven years has been about one of these topics [10, 20, 27, 38, 44, 49, 52, 55, 65, 68].

While there are a respectable amount of visualization tools for computational theory, and in particular automata, there is a notable lack of AVs related to NP-completeness and proofs. The most notable studies in the primary set related to NP-completeness include those of Pape [53], Crescenzi et al. [17], and Maji [45]. The work of Pape [53] deals largely with static, proof visualizations rather than mapping reductions. Crescenzi et al. [17] outlines the addition of four new proof visualizations to a popular AV visualization software called AlViE; however, this software has not been updated for over a decade and both the website and application download are no longer operable. Finally, the work by Maji [45] provides a tutorial for understanding NP-completeness using the OpenDSA framework. However, this tutorial has not been updated in many years and is largely static, limiting its ability to effectively engage learners outside of raw introductions. Despite these shortfalls, Maji's work [45] provides one of the best examples of computational theory AV for NP-completeness and mapping reducibility. Still, there is a need for additional contributions in this particular field. Many significant real-world optimization and decision problems fall into the class of NP-complete problems, signaling the ever-present need for students that are well-versed in how to best approach them [40].

2.3.4 Of the visualization publications that exist, which are dynamic and interactive?

The majority of the AV tools included in our review featured dynamic interactivity. We define *dynamic* as the characteristic of the simulation to change its appearance in response to user impetus (e.g., mouse click or hover over). We define *interactivity* as the ability for the user to change the underlying data from which the visualization is derived. Though it is possible to have an AV that is strictly dynamic or strictly interactive, we found that these features usually appear together in an AV tool and that they were more common in platforms that congregate a number of CT visualizations (e.g., JFLAP). Figure 2.4 summarizes which of the platforms reviewed in this study included dynamic interactivity.

2.4 Conclusion

Visualization tools for computational theory have seen some attention in the last two decades. In particular, applications such as JFlap [59] provide a rich, interactive visual experience for students eager to learn about automata, grammars, and Turing machines via interactive dynamic visuals. Despite there being some applications which have had success pedagogically, most have been discontinued and are no longer supported in development. In addition, studies detailing new applications and AV solutions are rare as there seems to be a decline in published studies in this field starting around 2010 compared to the previous decade. Finally, while some branches of computational theory have many applications supporting them, others lack similar attention. In particular, NP-completeness, reducibility, and proof construction represent some of the most powerful concepts in a computational theorist's knowledgebase but have received little attention in pedagogical visualization. Instructors looking to build and reinforce learning in these particular topics may find that they lack opportunities to expand outside traditional instruction techniques because of the lack of alternative teaching tools. We propose that a course change may be needed to craft new resources to serve this community. Learning through interactive dynamic visualization tools can benefit theoretical

discussions by alleviating traditional difficulties, particularly as these tools carry the added potential of expanding access and educational opportunities to new communities.

Chapter 3

Visualizing the 3SAT to CLIQUE Reduction Process

Submitted and Accepted to the 2022 Intermountain Engineering, Technology and Computing (i-ETC)

3.1 Introduction

Many of the complex problems facing the world today have been formally classified as what computational theorists call NP-complete problems, meaning problems for which no known algorithm exists for optimally solving the problem in a tractable time frame. Meanwhile, NP-completeness consistently rank among the most difficult concepts in computer science education. This is due to the abstract nature of the material and the perceived lack of applicability to industry work [22]. This puts computer science professors and instructors in a tough position; while NP-completeness and theoretical computer science is the backbone of problems we solve in our field, it also happens to be very difficult to teach and instruct. In particular, students struggle with reduction theory. In layperson's terms, reduction theory deals with polynomial mapping functions that can convert an instance of problem A into an instance of problem B. Every NP-complete problem has this reduction property stating that any problem in the complexity class NP can be reduced to any NP-complete problem. Fig. 3.1 showcases a map of the aforementioned complexity classes under either assumption that P $\neq$ NP or P = NP. Much of the theoretical and practical application of this space assumes that they are not equal. Nevertheless, more effective pedagogical methods are needed to equip students with practical skills for recognizing and addressing NP-complete problems.

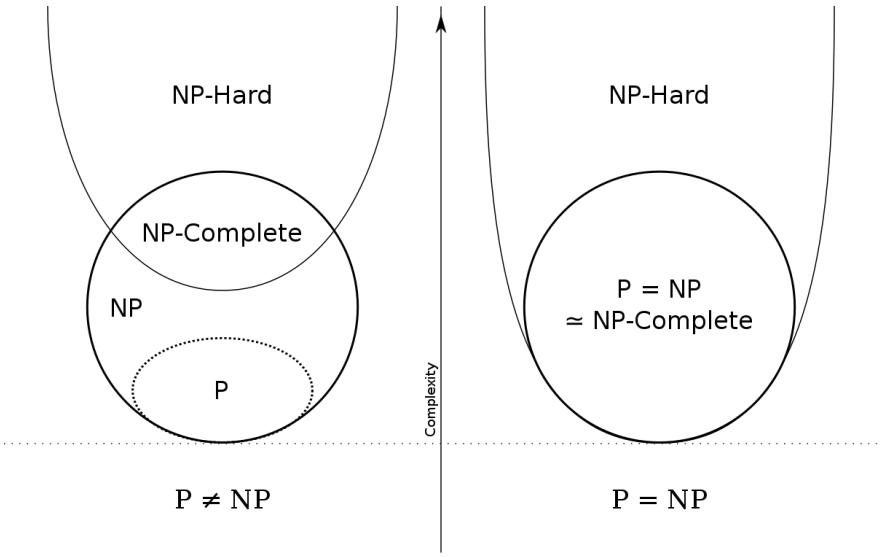


Figure 3.1: An Euler diagram of complexity classes for P, NP, and NP-complete. The right side is valid under the assumption that P equals NP while the left relies on the assumption that P does not equal NP. The vertically of each class correlates to its complexity. Most computer scientists believe P $\neq$ NP, and much of the theory and practical application in academia assume they are not equal. In any case, many significant real-world problems are demonstrably NP-complete, and more effective pedagogical methods are needed to equip students with practical skills for recognizing and addressing these real-world NP-complete problems. Figure is taken from Wikipedia.

Consider for example, the infamous traveling salesperson problem (i.e., "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"). This problem can (in tractable time) be reduced to instances of graph coloring problems, boolean satisfiability problems, set cover problems and virtually an infinite number of other problems that routinely crop up in real-world contexts. An efficient, polynomial solution to any of these problems stands to benefit societies and organizations in untold ways [61]. Thus, it is fundamentally important that we teach NP-completeness and reduction theory in a way that students can comprehend and absorb.

Enter, visualization. Applying visual learning to theoretical concepts such as NP-completeness and reduction theory creates opportunities for students who may not initially comprehend heavily abstract concepts. Using Two.js and a standard web framework stack, we have created a interactive visualization tool to allow students to visualize the reduction process. While still in its infancy, the results in this paper showcase a strong foundation of growth.

3.2 Related Works

Algorithm Visualization Tools (AVs) have been examined as an educational tool for traditional computer science education. For example, Stasko et al. finds that AVs are generally ineffective without a host of characteristics [67] while T.L. Naps finds that they provide an effective pedagogical tool for computer science educators [51]. While the effectiveness of AVs without specific characteristics is still a debate, the potential benefits of such tools warrants motivations to discover and review the bounds of existing literature. Furthermore, a comprehensive collection of AV research when applied specifically to computational theory does not yet exist. Before beginning this project, a systematic mapping study was performed as described by Peterson et al. [56]. This study is in the process of being professionally published; however, we still find it important to showcase some of its findings to highlight related works.

A total of 2492 papers were examined in this study. Of these, the most notable in relation with this project come from White [77] and Rodger [59] who develop Java widgets to simulate other computational theory concepts. Most notable, however, is that we find an extreme lack of visual technologies focused on reduction theory and NP-completeness.

3.3 Problem Definitions

Before examining the visualization tool, we define the two NP-complete problems involved in the reduction to be visualized. Michael Sipser, in his Introduction to the Theory of Computation [66], includes a reduction from *3SAT* to *CLIQUE* for the purposes of demonstrating the

CLIQUE problem NP-complete. While still complex, this reduction serves as an introduction to problem equivalence. In designing an educational application for visualizing this reduction process, we expect this particular reduction to be among those most well-known to beginning computational theory students given that is frequently referenced in beginning computational theory textbooks.

Along with the definitions of both these problems, we will also define the polynomial mapping function that converts an instance of problem A into an instance of problem B.

3.3.1 3SAT

A *literal* is a Boolean variable as in x or $\overline{x}$. A *clause* is several literals connected with disjunctions such as in $(x_1 \vee x_2 \vee \overline{x_3})$. *CNF form* is defined as a Boolean formula that is comprised of k clauses connected with $\wedge$'s, such as $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1}, x_3, x_4)$. k-CNF form defines that each clause has k literals. *3SAT* is defined as the problem of determining if a satisfying assignment exists for a given Boolean formula ϕ of k clauses in 3-CNF form. An example of a formula ϕ can be seen in Fig. 3.3.

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$$

Figure 3.3: Example of a valid 3SAT formula ϕ with three literals and three clauses.

3.3.2 CLIQUE

CLIQUE is defined as the problem of determining if an undirected graph G = (V,E) has a subset of k nodes such that every two distinct vertices between the nodes in the subset are adjacent. An example can be seen in Fig. 3.4.

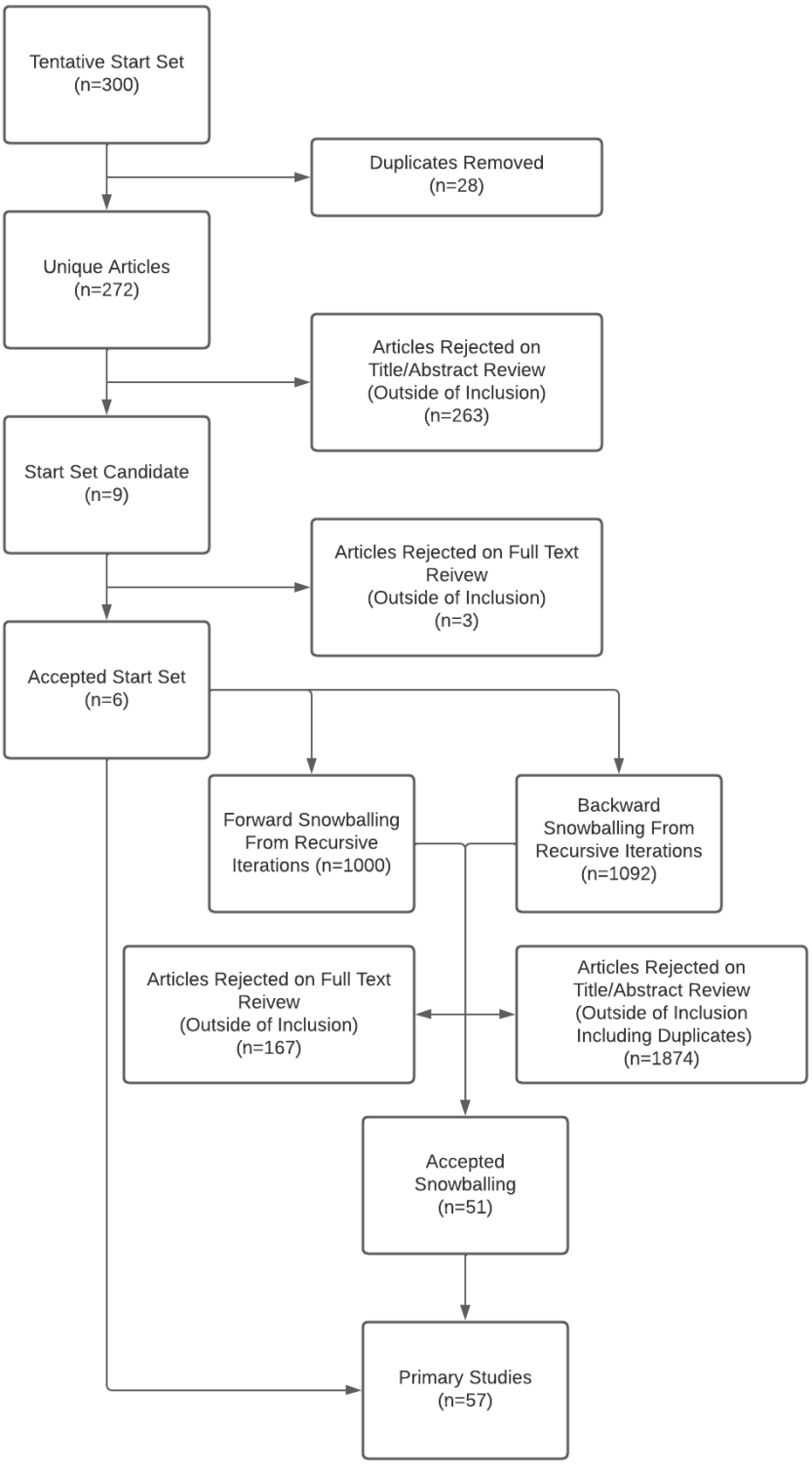


Figure 3.2: A breakdown of a systematic mapping study performed on interactive, visualization technology for computational theory. A total of 2492 papers are examined using rules prescribed by Peterson et al. We find that there is an extreme lack of visual technology focused on reduction theory and NP-completeness.

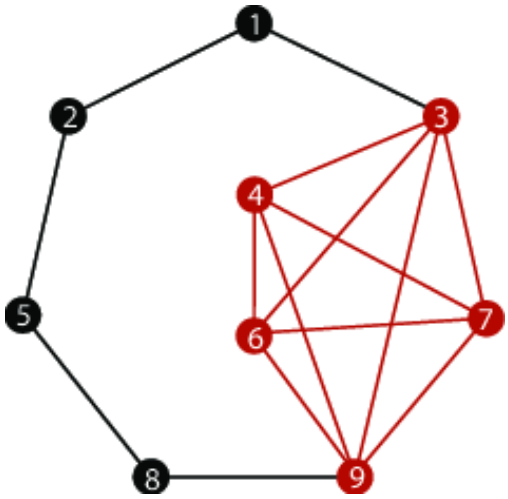


Figure 3.4: Example of a 5-CLIQUE in a graph as shown in red between nodes 3, 4, 6, 7, and 9. [3]

3.3.3 Polynomial Mapping Function f

Given an instance ϕ of 3SAT, we compute $f(\phi)$ as follows:

- Construct a graph $G = (V, E)$ of k clusters where k is equal to the number of clauses in ϕ. 3 nodes should be in each cluster as we are reducing from 3SAT. Each cluster corresponds to one of the clauses in ϕ and each node corresponds to a literal in the associated clause.
- Label each node of G with its corresponding literal in ϕ.
- Create an edge between all pairs of nodes in different clusters except pairs that have one, or both, of the following characteristics:
  - Pairs where one is the compliment of the other (e.g., x_1 and $\overline{x_1}$)
  - Pairs that belong to the same cluster.

$f(\phi)$ runs in polynomial time. Both the construction of the nodes along with their respective labeling runs in constant time. Creating edges between all pairs of nodes takes $O(n^2)$ time.

The conditions check requires another $O(n^2)$ operation as we check each pair of nodes. This is the exact formula that the algorithm relies on to construct a CLIQUE instance from 3SAT.

An example of a valid construction can be seen in Fig. 3.5.

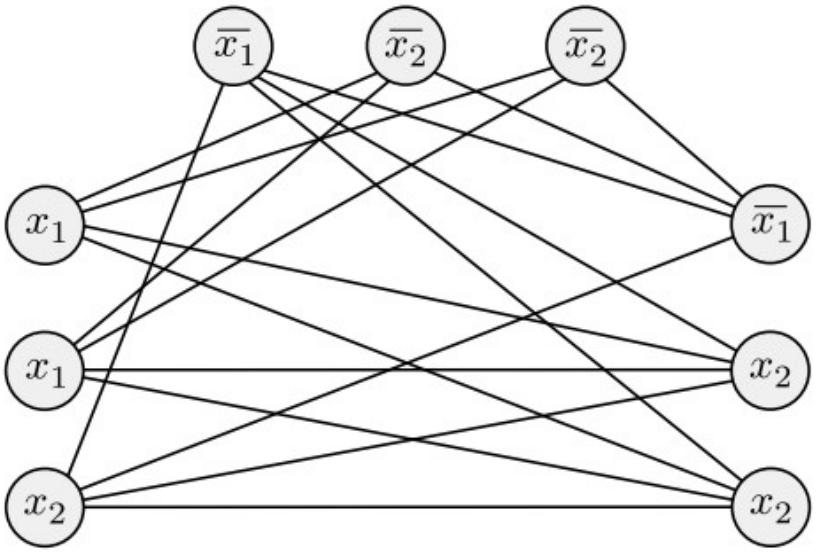


Figure 3.5: A valid construction from $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$. The nodes are edges are formulated via $f(\phi)$ [66].

3.4 Results

Using the mapping reduction function defined in section 3.3, a user can use our tool to input any arbitrary *3SAT* or *CLIQUE* instance and visualize its respective counterpart.

The visualization relies on two.js to create an svg with the nodes, edges, and appropriate intractability. We have separated out the three sections that highlight how our visualization is created.

3.4.1 Data Wrangling

The GUI for the visualization sends a request to a public API published on Idaho State Universities servers. This performs a full reduction on the back-end and returns an instance

of the CLIQUE problem for graphical reconstruction. This back-end function breaks down the user input and separates the string into its clauses, then literals. Each clause becomes an array in the triples matrix and the name of each literal is recorded along with a Boolean identifier determining if it is an inverse (ex,. $\overline{x_1}$).

The objective is to give the user as much flexibility in naming their literals, but also maintain some special characters for parsing.

3.4.2 Creating the Nodes

In order to facilitate the creation of the nodes. A custom class was created to store the necessary information for each object. Furthermore, a number of custom methods were incorporated to help orient the graph when it is dynamically created.

Each literal passes through a block function that translates it into a node object. The nodes maintain their cluster relationship and are categorized as being the top node, middle node, and bottom node. Once each of the clusters are created, then are placed around the center of the svg. This process is important as the location of each node and associated cluster mimic most academic textbooks teaching this reduction.

These node objects are then passed into a matrix where edge creation can then take place.

3.4.3 Creating the Edges

The creation of the edges takes the longest in terms of complexity. The time it takes to check conditions and create edges is, at worst, O(n^2) where n is the number of nodes.

Using iterative loops, we check each node in correlation with every other node to determine if there needs to be an edge. To determine if two nodes belongs to the same cluster, we simply use the iterative index's and see if they are checking two nodes that belong to the same triple in our matrix. To determine if two nodes are the inverse of one-another, we check the name of each node along with a custom Boolean identifier attached to the node class.

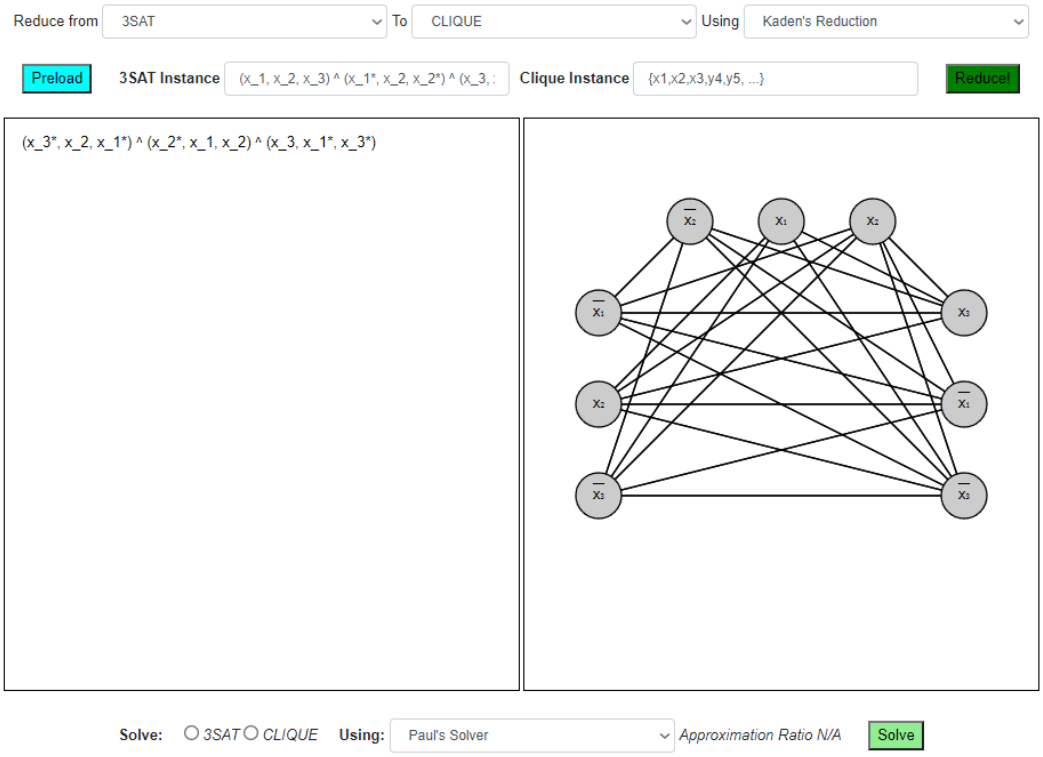


Figure 3.6: A Sample output of the visualization application. At the top, the user can select a problem to reduce from, reduce to, and a polynomial mapping function that facilitates the reduction. Users can manually enter a problem instance or pre-load an existing configuration before reducing. The CLIQUE graph on the right is dynamically generated from our polynomial mapping function $f(\phi)$ where $\phi = (\overline{x_3} \vee x_2 \vee \overline{x_1}) \wedge (\overline{x_2} \vee x_1 \vee x_2) \wedge (x_3 \vee \overline{x_1} \vee \overline{x_3})$. The 3SAT instance is editable and results in a dynamic, interactive visualization of CLIQUE. At the bottom, the user has access to multiple algorithms to solve either the original 3SAT instance or the dynamically-generated CLIQUE instance.

If both conditions return false, then we build an edge between the two nodes in question and continue iterating.

Fig. 3.6 showcases a graph that has gone through every stage of our reduction function and represents a 3SAT problem.

3.5 Discussion

There were a number of challenges we encountered during developing. These challenges include difficulties arranging the nodes in the correct position, incorporating the function

from my theoretical research into a practical application, and allowing a user to dynamically create a graph of their choice via a unique 3SAT instance. In considering these and other challenges, there may be other ways to make even more complex visualizations that are better able to visualize 3SAT as a CLIQUE representation. For example, in instances where the resulting CLIQUE graph overwhelms the applicability of a 2d graph, we could resort to multi-dimensional visualizations.

3.6 Future Work

There still is a number of features that need to implemented for the *3SAT* to *CLIQUE* reduction. For example, we would like to implement a verifier and solver button that will highlight possible solutions for the given *3SAT* instance and its respective *CLIQUE* solution. Furthermore, work needs to be done on the educational aspect of the tool. For example, we have plans in motion to add a "teach me" button to the GUI that will explain the reduction at each step of the visualization. Each of these features will be implemented next semester as a part of a thesis course.

Critically, work has been done to facilitate a open-source repository of NP-complete problems and their reductions. This repository has seen immense growth in the coming months and will soon be released to allow scientists and engineers easy access to problems and their reductions via public API.

This project creates a strong foundation for which an entire team is being created for. We have recently been awarded a number of research grants given to help us build a team of students that will add additional NP-complete problems to this database. Within 2 years, we are confident that we will have a back-end public API with access to hundreds of NP-complete reductions and visualizations. Developed in parallel will be a GUI visualization tool that will showcase each of those visualizations dynamically, as shown in this paper with *3SAT* to *CLIQUE*.

Chapter 4

Redux: An Interactive, Dynamic Tool for Learning NP-completeness and Mapping Reductions

This paper will be submitted to ACM SIGCSE 2024 for publication

4.1 Introduction

The ability to understand, recognize, and reduce nondeterministic polynomial-time (NP) complete[1] problems remains an important skill for students of computer science (CS) [40]. Despite this importance, significant shortcomings exist in the effectiveness with which the subject is taught in CS curricula with most instructors persisting in teaching the topic using rote paper-and-pencil methods and others not covering the topic at all [43]. Paper-and-pencil methods limit instruction to simple instances of these NP-complete problems and make more complex reductions very time-consuming if not outright impractical. Several successful dynamic interactive visualization tools have been developed to teach other topics in computational theory (CT) to address this same concern. Such tools have a proven track record for helping students develop theoretical and practical mastery [18]. The most notable of these tools is JFLAP [59], an automata visualization simulator that student surveys indicate makes learning such concepts easier and more enjoyable [60]. Despite the apparent success of these tools in other areas, a survey of the literature suggests significantly less attention has been devoted to developing tools to teach the CT topic of NP-completeness. There exists a demonstrated need for dynamic interactive visualization tools for teaching NP-completeness.

NP-complete problems play a significant role in optimization and decision-making across a spectrum of disciplines: genome assembly; scheduling problems; nuclear core fuel reload optimization; route planning; and the list grows ever longer. As a result, there exists

[1]Currently Redux only considers decision problems. However, minimal modifications envisioned as future work will enable the framework to accommodate NP-hard optimization variants of the NP-complete problems we consider.

a widespread need for computer scientists equipped with the knowledge and skills needed to address these problems. This includes the ability to recognize (i.e., prove) a problem as NP-complete; leverage existing heuristic solutions to solve novel instances of NP-complete problems; and develop novel heuristic solutions that are both time-efficient and that provide quasi-optimal results. Research suggests that a minority of undergraduates advance to their careers with the necessary skills to be able to apply this theory in their field [7].

A recent literature review of dynamic interactive visualization tools for teaching CT showed that such tools have been effective in helping students gain practical mastery through simulation of state automata [52, 59, 65]; context-free grammars [1, 59]; pushdown automata [24, 59]; pumping lemmas [13, 59]; and Turing machines [26, 59]. Yet there remains an apparent lack of tools for dynamic and/or interactive visualization of topics related to NP-completeness, including mapping reductions between NP-complete problems and the application of heuristic solution algorithms to NP-complete problem instances. The most notable existing literature for visualizing concepts related to NP-completeness exists in the OpenDSA framework [45]. OpenDSA offers an effective introduction to NP-complete problems and mapping reducibility. The problems demonstrated in this framework are highly limited, however, and are not designed to allow for students to interact with or modify the example instances. Other contributions of theoretical NP-complete visualization tools appear to have been abandoned or have shifted no longer centered on NP-completeness [17]. For example, AlViE is an algorithm visualization environment dedicated to helping students better conceptualize concepts in typical algorithms courses; however, the application to appears to have been moved and any mention of it appears to be confined to algorithmic visualization, despite literature claiming to have added NP-complete visualizations.

We present a novel web-based dynamic visualization tool called Redux designed as a pedagogical aid for teaching NP-completeness. In particular, the tool provides examples of several canonical NP-complete problems, allows the user the ability to define custom instances of these problems, and visualizes these instances, their mapping reductions to

other NP-complete problems, and their possible solutions. The interactive aspect of the tool enables the user to examine how particular reduction algorithms work by highlighting the analogous substructures or *gadgets* between the visualized problem instance and the reduced problem instance. The interface lies atop an extensible knowledgebase and provides students the educational opportunity of comparing their own reduction implementations and heuristic solution algorithms against those already present.

4.2 Methods

The development of Redux was largely inspired by the features of existing visualization tools such as JFLAP, with an emphasis on long-term sustainability. In this section, we describe how Redux was designed and how that plays into the needs of education.

4.2.1 Implementation

In order to facilitate the most accessibility, Redux was designed using the web framework React with a RESTful API back-end developed in C#. This allows Redux to function as a "dual-use" application where students can utilize the website for pedagogical visualizations and supplement information about the problems while researchers can utilize strictly the API for meta-level research. For example, if a researcher today wanted to examine how subtle changes to a problem input changes a reduction algorithm, they would have to manually reduce each instance or write their own program to automate the process. Using Redux's back-end API, we can allow this researcher to send thousands of problem inputs directly to our API simplifying their work. At the same time, a student can access our website from anywhere in the world to learn about NP-completeness, reduction theory, and algorithms without preconceived knowledge on how to structure an API call that will satisfy their curiosity.

Before development began, we considered what other solutions exist today and how we can address current problems in the field. Previous attempts to visualize NP-completeness,

Problems	Polynomial-time Reduction(s) to	Heuristic Solution Algorithm(s)	Verifier
0-1 INTEGER PROGRAMMING		Greedy	x
3D MATCHING		Hurkens Schrijver [34]	x
3SAT*	0-1 INTEGER PROGRAMMING, 3D MATCHING, CLIQUE*, GRAPH COLORING	Backtracking [66]	x
CLIQUE*	VERTEX COVER*	Greedy	x
EXACT COVER*		Greedy	x
FEEDBACK ARCSET		Naive approximation (2.0)	x
GRAPH COLORING	SAT	DSatur [6]	x
KNAPSACK		Greedy	x
SAT		Greedy	x
SUBSET SUM	KNAPSACK	Greedy	x
TRAVELING SALESPERSON		Branch and Bound, Greedy	x
VERTEX COVER*	FEEDBACK ARCSET	Greedy	x

Table 4.1: *Current functionality in Redux.* A list of all problems, reductions, solvers, and verifiers currently available on the Redux platform. Visualization of problem instances and of reductions is available for problems/reductions marked with an asterisk. Parentheses indicate the approximation ratio where applicable.

most notably the AlViE framework [17], are no longer available and showcase the need for so-called "future-proofing". With sustained longevity in mind, both the API and the website are hosted and supported by Idaho State University to facilitate load-balancing and future support for the project. Furthermore, the application is hosted publicly on Gitbhub to allow the community to reference, fork, and contribute to the project at any time. The Github repositories can be found here:

- API: https://github.com/marckade/Redux
- GUI: https://github.com/marckade/ReduxGUI

The API is structured to allow contribution in a way that is fully compatible with the front-end. Any change on the back-end will result in that contribution becoming available on the front-end immediately. This design pattern allows for students specializing in algorithms

to prioritize development on the back-end and still have their contributions made public without ever touching the front-end.

4.3 Redux Features

When first accessed, Redux displays the visualization of the 3SAT to CLIQUE reduction—a reduction between two canonical NP-complete problems that appears in many CT textbooks. Users can immediately begin to change the 3SAT instance and see how these changes are reflected in the reduced instance. They can highlight gadgets and compare the results of different solution algorithms.

Problem instances are defined using traditional discrete math notations. For example, Figure 4.1 showcases the visualization of a CLIQUE instance and a VERTEX COVER instance drawn from problem definitions in set notation. In this way, each problem is implemented with an example problem instance that demonstrates the expected syntax for defining arbitrary problem instances. This provides students with an opportunity to learn discrete math notations and see resulting visualizations as they modify inputs. This feature was inspired by the many interviews we had with students and professors. Some of the early problem notations were derived from definitions found in computer science literature. This notation was unfamiliar to particular math professors we interviewed that used different notations and language to describe the same problem.

All fields are paired with an information button that displays a dialog box with the problem definition, the originator, the contributor, high-level algorithm descriptions, and hyperlinks to find more information.

As is well-known, the proof that a problem is NP-complete requires proving two defining features of the problem. First is the polynomial-time mapping reduction from an existing NP-complete problem. Second is (typically) a polynomial-time verifier that, given a prospective solution for a problem instance, proves that solution valid or invalid. Redux includes modules to visualize and interact with both features for a selected problem. The

inclusion of these features allows a student to understand why a problem is NP-complete and how this theory interacts with other concepts they are familiar with. Currently, Redux focuses on NP-complete variations of problems (decision variants) rather than their NP-Hard counterparts. We plan to include both in the future and allow students to experiment with either to help educate them on their respective differences.

Table 4.1 outlines what problems, reductions, solvers, and verifiers exist in Redux today. Problems and reductions that have an accompanying star also support visualizations on our GUI. These visualizations are fully interactive and offer instructors and students an opportunity to take problems typically shown in set-notation and turn them into interactive visualizations. The ability to **dynamically** generate visualizations for any problem (or accompanying reduction) is also valuable for professors who want to produce more sophisticated lectures before class, or perform live demonstrations.

Outside of being an accessible repository for NP-complete problems, reductions, solvers, and verifiers, Redux also provides the following novel additions to CT visualization technology:

4.3.1 Dynamic and Interactive Gadget Highlighting

One unique feature of Redux originates in our dynamic gadget highlighting. Since, by definition, NP-complete problems map to one another, we are able to pedagogically show the user how each element and substructure of problem A is converted to problem B. For example, how clauses and literals in a 3SAT instance are reduced to nodes and subgraphs in the analogous CLIQUE instance. Before we showcase this, we define 3SAT and CLIQUE as follows.

3SAT: A *literal* is a Boolean variable as in x or $\overline{x}$. A *clause* is several literals connected with disjunctions such as in $(x_1 \vee x_2 \vee \overline{x_3})$. *CNF form* is defined as a Boolean formula that is comprised of k clauses connected with $\wedge$'s, such as $(x_1 \vee x_2 \vee \overline{x_3}) \wedge (\overline{x_1}, x_3, x_4)$. k-CNF form defines that each clause has k literals. *3SAT* is defined as the problem of determining if a satisfying assignment exists for a given Boolean formula ϕ of k clauses in 3-CNF form.

CLIQUE: A *CLIQUE* is defined as the problem of determining if an undirected graph G = (V,E) has a subset of k nodes such that every two distinct vertices between the nodes in the subset are adjacent.

Using Redux, we can not only visualize the reduction between these problems but also dynamically highlight how each element of *3SAT* relates to the nodes of the *CLIQUE* graph. This feature is entirely dynamic and will work from any unique user input. This concept helps elevate some of the current pitfalls in traditional education as it allows the student to explore how characteristics of one problem, regardless of its input, map to another based on a reduction. Furthermore, it helps clarify how solutions map from one problem to another as students are able to highlight gadgets at the same time as viewing the solution. Figure **??** shows gadget highlighting in the reduction from 3SAT to CLIQUE. This feature works with any reduction on Redux and is not limited purely to *3SAT* and *CLIQUE*.

4.3.2 Visualizing Solutions

Redux also offers instructors and students the ability to visualize solutions using an algorithm of their choice. The ability to generate a solution and map it directly to the visualization helps students understand the original problems (See survey in Results section). This also proves educational when learning the distinctions between approximation and heuristic solving algorithms. The solution visualizer works with any problem input.

4.3.3 Transitive Reductions

The law of transitivity asserts that for some arbitrary problems *A,B,C*, if *A* is related to *B* and *B* is related to *C*, then *A* is related to *C*. This law also applies to mapping reducibility. Despite well-known laws stating that all NP-complete problems have polynomial-time reductions to one another, the vast majority of these reductions have not been discovered yet. Many dismiss the need to derive them as being a purely theoretical exercise. However, reduction

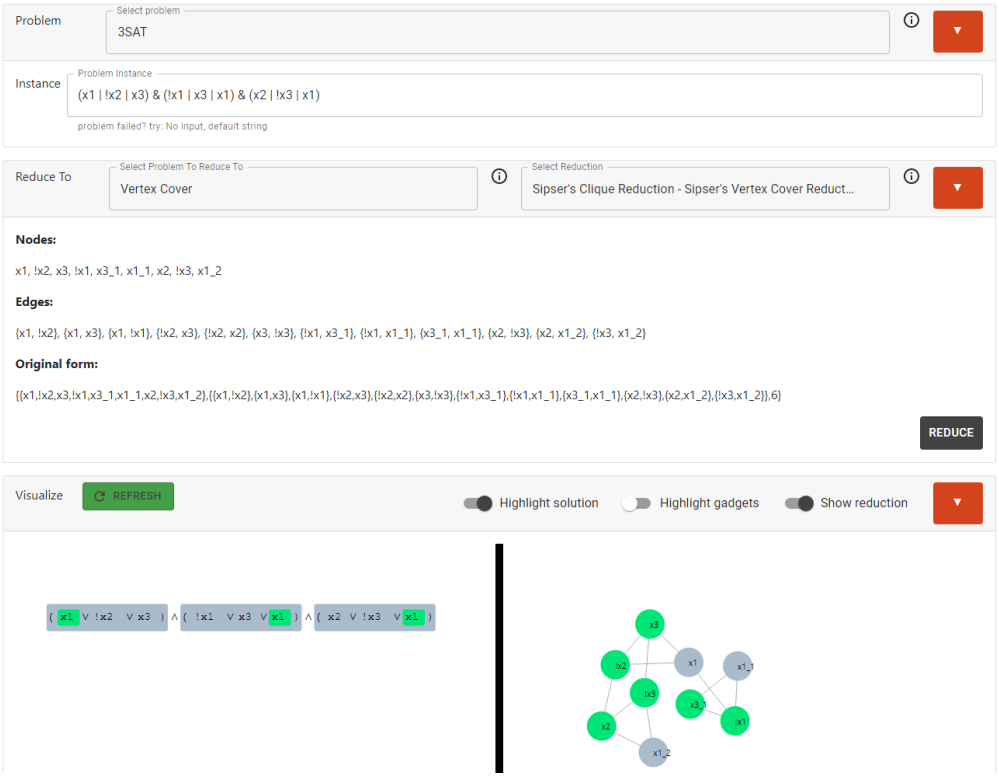


Figure 4.1: *Transitivity of reductions.* Because Redux implements both the 3SAT to CLIQUE and the CLIQUE to VERTEX COVER reductions, a reduction from 3SAT directly to VERTEX COVER can be derived automatically. Besides the pedagogical value of this example, transitivity also allows a particular problem (e.g., 3SAT) access to more solution algorithms (e.g., CLIQUE and VERTEX COVER solvers) by which to search for more optimal solutions.

has practical value, as well, enabling one problem to be solved by algorithms developed for an entirely different problem. Redux has incorporated a feature that provides access to indirect reductions via transitivity. For example, *3SAT* has a valid reduction to *CLIQUE* and *CLIQUE* has a valid reduction to *VERTEX COVER*, we are able to transitively reduce directly from *3SAT* to *VERTEX COVER* by going through two separate reduction algorithms. As the set of problems and reductions implemented in Redux expands, the system is designed to automatically detect and provide user access when new transitive reductions become possible.

This novel feature has three primary benefits. First, the ability for students to experiment with transitive reductions provides yet another angle for students to explore

how the underlying theory applies to known problems. Second, it allows students to use the solvers and verifiers of problems many reductions away to solve the original problem (of which can be valuable when comparing solution algorithms). Finally, it provides students methods of reduction that may not be discovered yet, or added to the Redux application. For example, Redux does not have a direct reduction implemented between *3SAT* and *VERTEX COVER*, however users can still reduce between the two problems as shown in Figure 4.1.

4.3.4 Open-Source Contributing

In order to establish an effective repository of AV solutions, problems, and reductions, Redux functions as an open-source application that encourages contribution. Because of how Redux was designed, instructors and students can contribute to either the algorithmic or visualization aspect of the project and see their results live on our website. This engagement is supported by the web app which provides templated code stubs and instructions on how to contribute your code.

4.4 Survey and Results

Two separate surveys were conducted to evaluate the pedagogical effectiveness of Redux. The first survey was administered to a group of students to assess the tool's effectiveness as a lecture supplement. The second survey was administered to a group that included industry professionals to assess the usability and overall effectiveness of individual features.

4.4.1 Survey 1: Assessment as Lecture Supplement

We conducted a preliminary mixed-mode pilot study of 27 students currently enrolled in "Introduction to Computational Theory" and/or "Advanced Algorithms". Respondents were randomly assigned to either a control group with no access to Redux (12) or a test group with access to Redux (15).

After a 30-minute lecture introducing 3SAT, CLIQUE, and the reduction between them, the groups were separated and asked a series of questions designed to assess their conceptualization of the source material. The survey was split into two sections: **practical application** and **theoretical implication**. The first assessed the ability to solve simple NP-complete problems. As an example, given a simple 3SAT or CLIQUE instance respondents were asked to identify which of several options was a valid solution to the instance. The second assessed ability to extend theoretical concepts to new scenarios. As an example, given a 3SAT instance, respondents were asked to identify the correct CLIQUE instance resulting from a particular mapping reduction. Respondents were also asked conceptual questions to correctly identify how reductions can be applied both in A) proving NP-completeness and B) indirect solutions to NP-complete problems. Overall, the control group performed slightly better (3%) on the first part; the test group performed slightly better (9%) better on the second part (see Table 4.2).

Careful consideration of the results from the pilot study suggested several important changes that needed to be made both to improve Redux' usability and to improve the survey. These included: adding survey instrumentation to assess the user's ability to apply Redux to non-trivial problems, including those involving reduction; adding a brief tutorial to acquaint users with the tool's features; and improving several aspects of the GUI to make basic navigation more intuitive.

Survey section	**Control Group**	**Test Group**
Practical application	63.3 ±29.6%	60 ±16.1%
Theoretical implication	68.3 ±31.9%	77.3 ±23.7%

Table 4.2: *Results of Survey 1 (Pilot Study).* The results of the pilot study were largely inconclusive and led to several important revisions in both the survey instrumentation (e.g., revising questions to more specifically relate to the purposes for which Redux was built: application to non-trivial instances, breadth of application features, and usability) and in the Redux application.

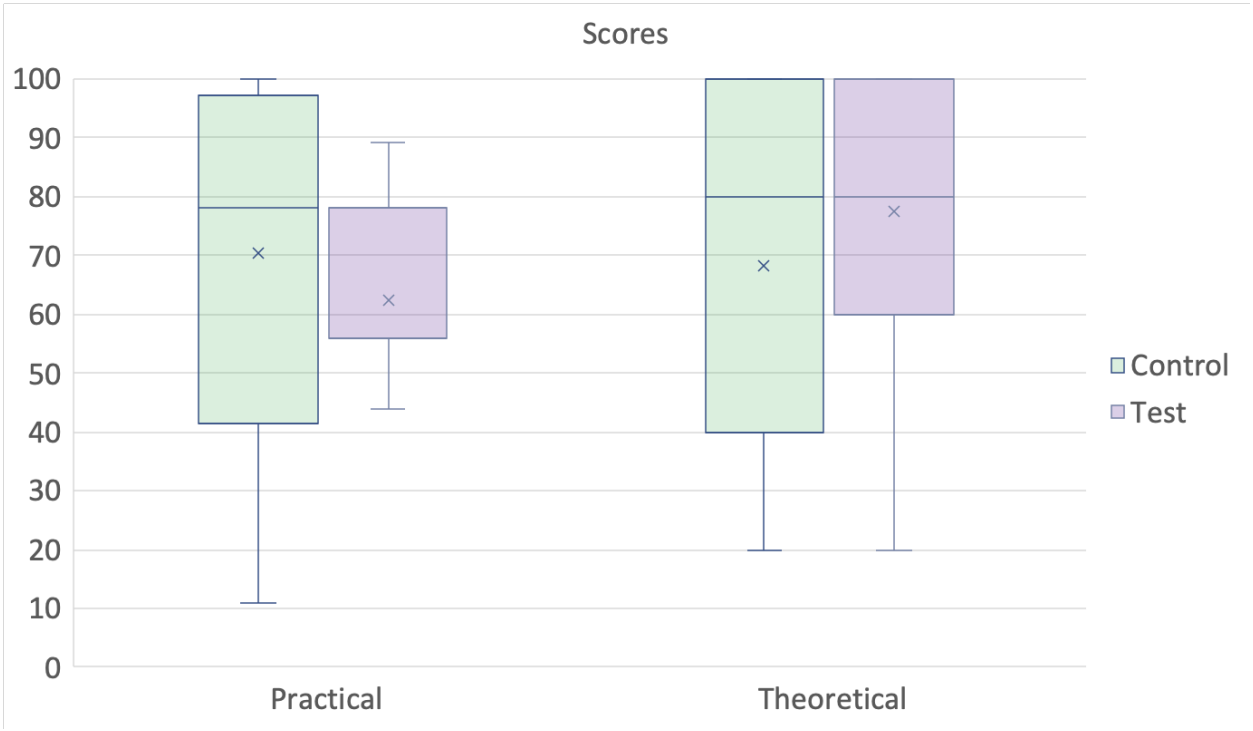


Figure 4.2: Box plots for each group and assessment section. While the control group performed better in the practical application of the theory, the test group performed better on the theoretical material. Furthermore, the standard deviation and outer quartiles indicate a much tighter score line for individuals in the test group. The 'X' indicates the mean for each plot.

4.4.2 Survey 2: Standalone Usability Assessment

The second survey was adjusted to better assess the usability of Redux as a standalone application and examine how each application feature contributed to the user's understanding of the underlying material. Its sample size of 13 was made up of students and industry professionals working in UI/UX, and/or positions related to the theory of computation and algorithms. Participants were given Redux along with a short Redux tutorial before being asked to perform calculations, find solutions, and create visualizations. Along the way, they were asked to evaluate the layout of the application and rank how intuitive accomplishing each task was.

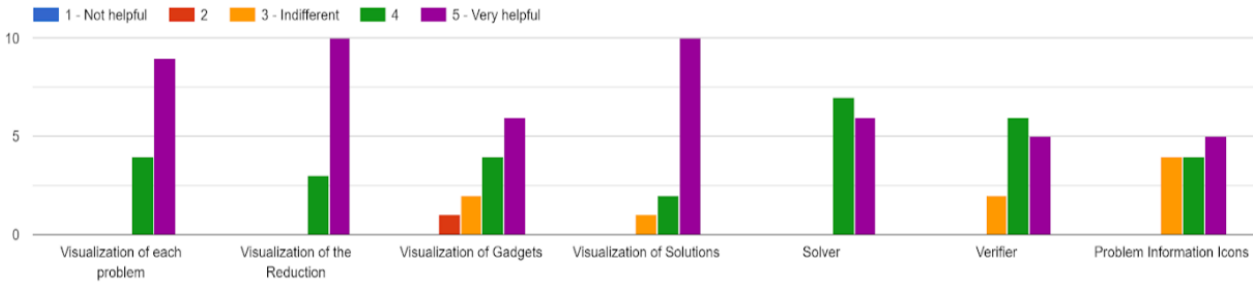


Figure 4.3: *The Most Helpful Redux Features.* Survey two asked participants to rank how helpful each Redux feature was in learning the underlying concepts. The four highest-ranked features all relate to the visualization of problems, reductions, gadgets, and solutions.

The results pointed to high usability (the average Likert score over fourteen questions related to usability was 4.6 out of 5) with higher usability scores for tasks related to data input, accessing problem documentation, and instance visualization. Participants were asked to rank features by how effectively they facilitated learning. As Figure 4.3 outlines, the most helpful features related to visualization.

Participants also indicated that the use of Redux helped make learning NP-complete reductions more enjoyable with 77% of respondents agreeing or strongly agreeing with the statement "Using Redux makes understanding NP-complete reductions more enjoyable."

To evaluate the extent to which visualizing reductions aided respondents in understanding the concept of mapping reductions, users were asked to rank their understanding of specifically the *3SAT* to *CLIQUE* reduction at various steps in the survey. In particular, participants were asked to rank their understanding 1) after seeing the reduction pane (no visualization), 2) after visualizing the reduction and solution, and 3) after enabling and exploring gadget highlighting. Results indicate that participants' understanding of mapping reductions—both the specific 3SAT-to-CLIQUE reduction and reductions generally—was significantly enhanced with exposure to each additional visualization feature. Results from these questions can be seen in Figure 4.4.

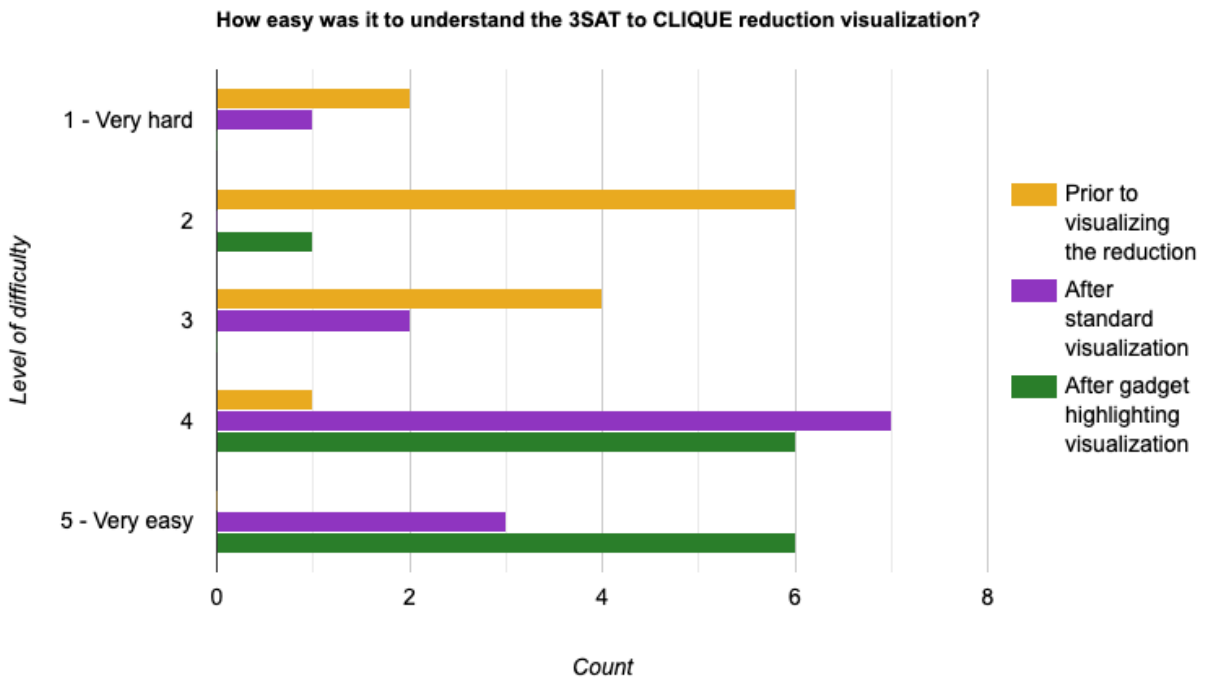


Figure 4.4: *Understanding the Reduction at each Visualiation Step.* Survey two asked participants to rank how easy it was to understand the *3SAT* to *CLIQUE* reduction after seeing the reduction text pane, after visualizing the full reduction, and after enabling gadget highlighting. Results indicate that participants had an easier time understanding the theory the more visualization features they had access to.

4.5 Discussion and Conclusion

More than simply providing instructors and students with an interactive dynamic pedagogical tool for CT algorithm visualization, the vision of Redux is to provide a platform for academic research, for open-source contributions from students and practioners alike, for facilitating improved comparative analysis of algorithmic solutions, and for exploring ways encourage more widespread incorporation of the applied benefits of CT in industry. Students can use the application to explore more complex instances of problems and reductions and to uncover how simple changes to an input can change its resulting solution, all while visualizing the

entire process. Professors can utilize the independent back-end API for meta-level research or can use the website to quickly generate more complex visualizations for lectures.

Additional work for the Redux application has already been scheduled. This includes a plan for a larger, more sophisticated survey between multiple universities as well as full API documentation for Redux users who want to take full advantage of the algorithms powering the front-end. This work will be completed by the end of 2024 and be made accessible on the main Redux website. Future development also includes the addition of NP-hard (optimization) variants of currently accessible problems, along with additional visualizations for problems such as the traveling salesperson and knapsack. The class P in Redux could also be used to demonstrate P algorithms, for example, Dijkstra's algorithm [21] for the shortest path in a weighted, directed graph. Early results indicate that these additions could increase the opportunity for students to understand these difficult theoretical concepts.

Redux represents a highly extensible framework with many possible directions for future work. As an open-source application, we have opened the application's code-base up to allow for additional collaborators interested in contributing visualizations, algorithms, or any combination of the above. As the needs of industry change, Redux aims to fill the gap that currently exists in visualization technology and allow a greater understanding of NP-completeness, computational theory, and its application.

Chapter 5

KAMI: Leveraging the power of crowd-sourcing to solve complex, real-world problems

Submitted and Accepted to the 2022 Intermountain Engineering, Technology and Computing

Awarded "Best Research Paper: Computing"

5.1 Introduction

As the winner of multiple Editors' Choice awards, KAMI boasts a player base in the tens of thousands. As a one-player combinatorial game with an engaging puzzle experience, KAMI may simultaneously hold a secret that would allow it to function as a powerful tool enabling computer scientists, doctors, and engineers to solve some of the world's most complex problems: NP-complete problems. NP-complete problems are uniquely characterized by the fact that mots NP-complete problems cannot be solved in a tractable time frame by modern day computers. They are also uniquely characterized by the fact that all NP-complete problems can be converted (in a tractable time frame[1]) into the representation of any other NP-complete problem, allowing solutions to one such problem to satisfy instances of other NP-complete problems.

Consider for example, the infamous traveling salesperson problem (i.e., "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"). This problem can (in tractable time) be reduced to instances of graph coloring problems, boolean satisfiability problems, set cover problems and virtually an infinite number of other problems that routinely crop up in real-world contexts. An efficient solution to any of these problems stands to benefit societies and organizations in untold ways [61]. It is important to note that while the solutions to these problems can be verified in polynomial time, the reason they are classified as NP is because the

[1] i.e., in polynomial time

problems themselves cannot be solved in polynomial time (NP stands for "non-polynomial"). NP-complete problems have the same properties that NP problems do with the exception that every existing NP problem must be reducible to it in polynomial time. This is a property that regular NP problems don't have. Many notable examples of NP-complete problems can be observed in Richard Karp's 1972 paper "Reducibility Among Combinatorial Problems" [39]. Practical examples of NP-complete problems are virtually everywhere in the modern world: scheduling problems [71], genome assembly and scaffolding[5], cloud-computing [76], resource allocation [35], powergrid computing [54], water flow planning [81], route planning, dependency planning, organizational behavior, and on and on.

As a flood fill game, a KAMI puzzle itself represents an instance of an NP-complete problem [11]. But unlike many NP-complete problem representations, tens of thousands of mobile game players willingly devote countless hours and money in the pursuit of designing and solving KAMI puzzles. What makes this particular form of NP-complete puzzle so alluring is that puzzles are creatively and artistically designed to be novel, interesting, intentional, and surprising—in short, KAMI puzzles are considered creative artifacts. In this paper we present an initial hypothesis on how the principles of computational creativity (CC) can be applied in the context of rendering arbitrary NP-complete problems as KAMI puzzles in order to leverage the massive KAMI crowd-sourcing platform to effectively solve real-world problems. Critically, this paper will serve as a pilot to future research we intend to conduct which would allow us to evaluate this hypothesis.

The idea of using crowd-sourcing to solve problems too complex for modern computers has been only recently explored, particularly in domains such as bioinformatics and quantum computing. Foldit is an online puzzle video game about protein folding (a notoriously difficult problem in bioinformatics) that leveraged a player base of 57,000 players to obtain useful results that matched or outperformed algorithmically computed solutions [15]. More recently, researchers in the domain of quantum computing introduced a citizen science game, Quantum Moves 2, and found that players were able to optimize solutions roughly on par with the best

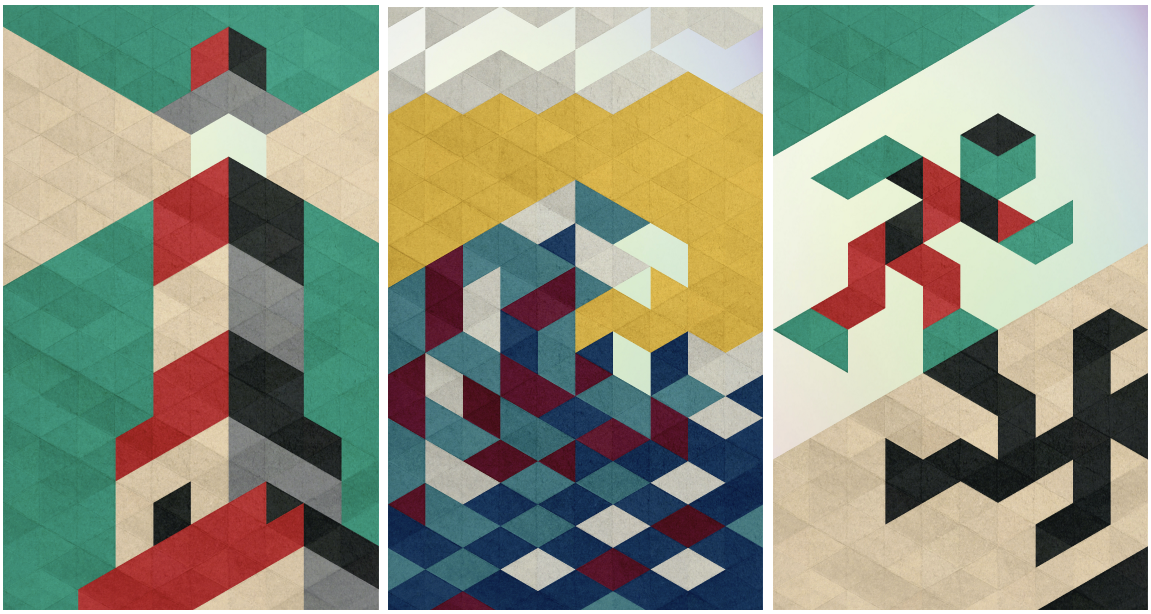


Figure 5.1: KAMI puzzles are characteristically artistic and creative. Finding optimal solutions to a KAMI puzzle via computational means is an NP-complete problem.

of the tested standard optimization methods performed on a computer cluster [37]. This area of study has been called "human computing" and is a topic of study in the field of Human-Computer Interaction (HCI) [42].

5.2 KAMI Puzzle Representation

KAMI is an origami-inspired puzzle game that presents players with creative geometric patterns (see Fig. 5.1). The creativity of the puzzles inspires players to work diligently to find optimal solutions to a puzzle. An optimal solution to a KAMI puzzle is a minimum sequence of moves that would transform the puzzle into a monochromatic plane (see Fig. 5.2). It's creative appeal has garnered a player base in the tens of thousands, something fundamental for crowd-sourcing.

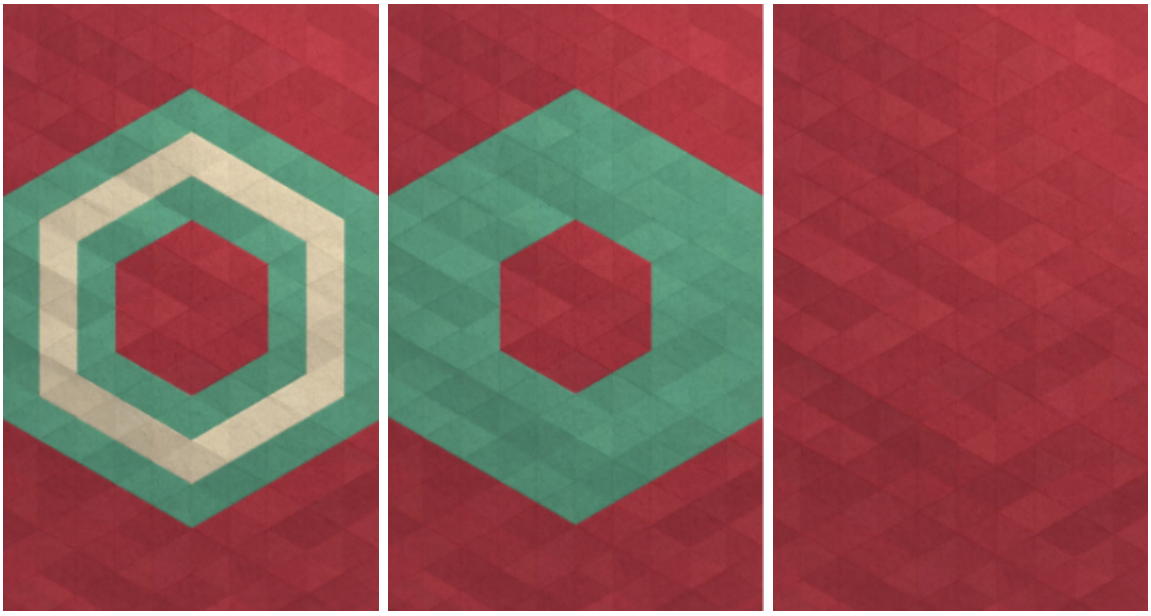


Figure 5.2: Each KAMI puzzle presents itself as a multi-colored plane. The player makes a move by replacing a colored section of the graph with a color of their choice (e.g., replacing white with green). The goal of the game is to make the entire plane monochromatic in the fewest possible moves.

5.2.1 Representing a KAMI puzzle as a Graph

A KAMI puzzle can be reduced to a simplified representation of the same problem as a colored graph (see Fig. 5.3). A KAMI puzzle P can be converted to an identical graph G according to the following algorithm:

- Each continuous monochromatic subregion of color in P, no matter its size, is shown as a single node in G of the same color.
- An edge exists between each pair of nodes in G for which their corresponding colored regions in P are adjacent.

As multiple KAMI puzzles can be simplified to the same graph, there exists a many-to-one relationship between KAMI puzzles and their simplified graphical representations.

Conversely, any colored graph G with the following three characteristics has at least one equivalent KAMI puzzle.

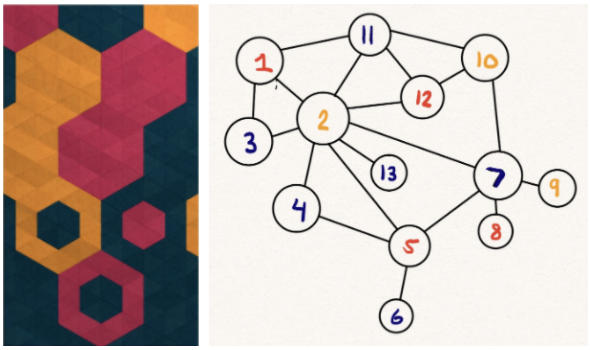


Figure 5.3: We can use the graph on the right to represent a simplified version of the KAMI puzzle on the right. There is a many-to-one relationship between KAMI puzzles and their simplified graphical representations.

- G must be finite
- G must be colored
- G must be undirected

A *move* can be played on a colored graph G by selecting a node $N \in G$ and performing the following:

- Change the color of N to a color of your choice C.
- If any two adjacent nodes N and N' are of the same color, add the neighbors of N' to become neighbors of N and remove N from G.

An example of a valid move, starting from Fig. 5.3 and then changing node 2 from yellow to blue, is shown in Fig. 5.4.

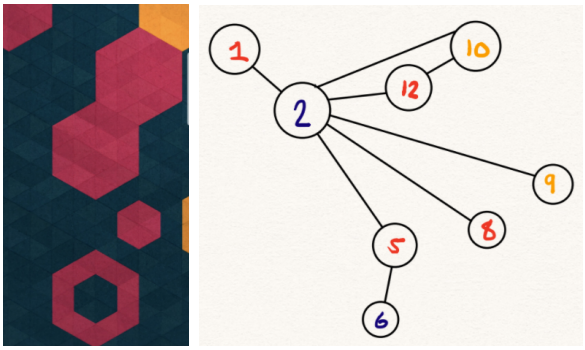


Figure 5.4: The above KAMI puzzle and its associated colored graph are the result of starting from Fig. 5.3 and making the move of changing node 2 from yellow to blue.

A KAMI solution for a given graph can be defined as any sequence of moves that results in a monochromatic graph with exactly one node. Its decision problem variant asks if a k-move sequence exists that turns the graph monochromatic. The optimal solution is the solution with the shortest sequence of moves. Finding the optimal solution is an NP-complete problem [11].

The fact that a single graph can encode the basic problem for many different puzzles creates an interesting challenge: *Given an arbitrary colored graph representing an arbitrary NP-complete problem, what is the most creative (i.e., novel, valuable, surprising, intentional) KAMI puzzle that can be used to represent that graph?*

The process of formally reducing an arbitrary NP-complete problem into an equivalent colored graph is beyond the scope of this initial foray into the domain. Suffice it to say for now that we know that such a reduction exists and its articulation is the focus of ongoing research.

5.3 A Deeper Significance?

The question arrived at in the last section represents an important point of consideration for those in the CC field. In some sense this represents yet another possible domain where CC

stands to make a mark. But there are several things that make this particular domain of peculiar interest.

First, **there is nothing particularly new** about the challenge being posed in this domain. Many CC researchers are currently interested in designing systems that can generate artifacts that would be considered creative in the visual art domain [14, 16, 33]. So much of the groundwork has already been laid, and we have strong reason to hope that we will to some meaningful extent find success in achieving a solution.

Second, **the impact factor is likely to be significant**. If indeed an arbitrary NP-complete problem can be rendered as a colored graph and then as a KAMI puzzle (as indeed we believe it can be), then an automation of that process to any extent puts real-world, NP-complete problems right under tens of thousands of people's itching, anxious fingertips in a format and on a platform that they already love and are familiar with. Besides the impact of finding solutions to some important real-world problems, the impact of empowering *people* to use their collective intuition and knowledge to solve those problems—particularly in a world where AI is continuously threatening to render such intuition and knowledge obselete—could be equally impactful.

Third, building off of the previous point, **this is an excellent example of practical CC**. One of the challenges that has been hypothesized as being a barrier to growth for the field of CC is its practical applicability [4]. Whereas the world sees great practical value in many of the advances of AI in general, the same has not proven as true for many of the advances of CC. The more we can demonstrate the practical value of CC to society, the more we will see greater growth and success as a community.

Fourth, **KAMI is only one of many domains/platforms via which CC could be used to render computationally-intractable, real-world, complex problems in accessible mediums.** We might just as easily ask: *Given an arbitrary data structure representing an arbitrary NP-complete problem, what is the most creative (i.e., novel, valuable, surprising, intentional) musical/literary/culinary/visual art puzzle that can be used to represent*

that structure? More than simply representing an isolated example of applied CC, we are suggesting a possible paradigm shift in motivation. Constraints lie at the heart of so many of the forms of creativity we seek to emulate in our CC systems. Rather than learning these constraints purely from a database of existing artifacts or arbitrarily defining them ourselves, could we not literally derive these constraints via some algorithmic reduction from the constraints we face in real-world problems of interest?

5.4 Challenges and Solutions

As we consider the steps necessary to reduce a real-world NP-complete problem to a colored graph, an immediate challenge that surfaces is: What if the reduction results in a colored graph that is too large, complex, or convoluted to be capable of being rendered as a KAMI puzzle that anyone would want to solve? There will almost surely be real-world problems for which this will be the case. But we might draw some hope from recalling the No Free Lunch Theorems for Optimization [79] which put forth that "for any algorithm, any elevated performance over one class of problems is offset by performance over another class." We might reasonably hope to find real-world classes of problems for which the reduction proves effective.

An equally valid challenge to the hypothesis we have presented would be to argue that even NP-complete problem instances of small sizes can be solved via brute-force algorithms in relatively short order. Given the limited canvas size of a KAMI puzzle, one might reasonably conjecture that any NP-complete problem that could be reduced to a puzzle that size would be trivial to solve in its original form by a brute force algorithm. Such may well be true. However, even with a canvas of the sizes shown in the figures above, the complexity of the underlying graphs grows exponentially with the size of the canvas, suggesting that non-trivial problem instances ought to be capable of being represented as KAMI puzzles.

In considering these and other challenges, there may be other ways to make even more complex puzzles accessible to the power of "human computing". For example, what if rather

than expecting a puzzle to be solved by individuals, the puzzle is framed as a team-based competition? What if individuals are partitioned sections of the puzzle to be solved, with optimal sub-solutions being combined in creative ways (e.g., via dynamic programming or divide-and-conquer algorithms)?

As stated in the introduction, as an initial exploration into the domain, our purpose has been primarily to create a blueprint for a future research agenda that will undoubtedly reveal a host of challenges not yet considered. But we hope that the work undertaken thus far will suffice to facilitate meaningful conversation with the community, conversation that may yield additional insights, solutions, and collaborations as we take next steps. In some sense, we rely on the research community itself to be its own crowdsourcing platform.

We have presented the hypothesis that arbitrary NP-complete problems can be reduced to a KAMI puzzle. The KAMI platform, with its tens of thousands of users, represents an ideal means by which to crowdsource solutions to optimization and decision problems of real-world significance. The success of the KAMI platform in this endeavor depends on the effective application of CC principles to render the underlying problem as a puzzle whose creativity will be sufficiently engaging so as to entice the masses to seek its solution. As such it represents a practical and impactful application of CC. In a broader sense, the approach suggests a possible paradigm shift towards using CC in general to represent complex real-world problems in more accessible ways to large groups of people for the purposes of leveraging "human computing" power. We eagerly look forward to advancing the research agenda we have laid out and to addressing the challenges that will arise in the process.

Chapter 6

Conclusion

Computational theory, and in particular NP-completeness, remains an important topic in computer science education. Despite this importance, students consistently rank these theoretical courses as being the most difficult. This is largely due to the abstract nature of the material and the perceived lack of industry applicability. While other subjects in computer science enjoy a plethora of visualization tools designed to alleviate these concerns, those specific to theoretical computer science are condensed to automata; context-free grammars; pushdown automata; pumping lemmas; and Turing machines. Chapter 2 discussed this body of CT visualization literature in more detail and concludes there is a lack of attention dedicated toward teaching NP-completeness and mapping reducibility. In order to remedy this, Chapters 3 and 4 discuss the creation of a tool designed to dynamically visualize canonical NP-complete problems, their solutions, and reductions to other NP-complete problems; we call this tool Redux.

As it exists today, Redux offers students and educators a visualization prototype for NP-completeness, reductions, and solutions. With its novel gadget and solution highlighting, it allows students to experiment with unique problem instances and validate the curiosity that static text images cannot. At the same time, instructors and CT researchers can utilize our back-end API to do meta-level research and generate unique visualizations for their lectures. Transitive reductions, also a novel feature of Redux, provide users with the ability to solve problems using the solution algorithms of other NP-complete problems many reductions away. Surveys shown in Chapter 4 indicate strong potential for Redux as it exists today. With high usability scores, we believe the current user interface offers a strong foundation to build in many additional features.

There are numerous opportunities for future work within Redux's framework. Additional problems need visualizations created and published to the front-end website. Open-

source documentation could be expanded to allow for the community to contribute directly to the project more effectively. There are numerous visualization enhancements that could also be developed. For example, a step-by-step visualization could be integrated for particular problems to show how each gadget is created and connected to the rest of the data structure. Such an addition would particularly benefit students brand new to the theory; indeed, a full tutorial could also be developed to introduce students to the website, the concept of NP-completeness, and many-one reductions.

Redux could also be expanded to facilitate cooperation with industry. The assembly sequence planning (ASP) [32], for example, is a known NP-complete problem in the field of nuclear engineering. Industry professionals could theoretically add such a problem to Redux and reduce it to an existing problem within the Redux database. From there, ASP could be solved using the solvers of other NP-complete problems. While ASP does have existing heuristic solvers, this example could be extended to problems without solvers today.

More than simply providing instructors and students with an interactive dynamic pedagogical tool for CT algorithm visualization, Redux provides a platform for a global audience to engage with, contribute to, discuss, compare, and probe the possibilities and limitations of computational theory. With an evergrowing knowledgebase and userbase, Redux has the potential to be transformative well beyond the classroom, providing a platform by which industry and research collaborators can seek out improved solutions to their problems, fueled by a new generation of students impassioned and imbued with confidence in their ability to apply computational theory to improve the world around them.

References

[1] André Almeida, José Alves, Nelma Moreira, and Rogério Reis. Cgm: A context-free grammar manipulator. *CoRTA'2008*, page 44, 2008.

[2] Priyanka Bansal and Viraj Kumar. A jflap extension for checking context-free grammars. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2013.

[3] Peter Bloem and Steven de Rooij. A tutorial on mdl hypothesis testing for graph analysis. *arXiv preprint arXiv:1810.13163*, 2018.

[4] Paul Bodily and Dan Ventura. What happens when a computer joins the group? In *Proceedings of the 11th International Conference on Computational Creativity*, 2020.

[5] Paul M Bodily, M Stanley Fujimoto, Quinn Snell, Dan Ventura, and Mark J Clement. Scaffoldscaffolder: solving contig orientation via bidirected to directed graph reduction. *Bioinformatics*, 32(1):17–24, 2016.

[6] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.

[7] WC Brookes. Computing theory with relevance. In *Australasian Conference on Computer Science Education*. Australian Computer Society Inc, 2004.

[8] Ryan Cavalcante, Thomas Finley, and Susan H Rodger. A visual and interactive automata theory course with jflap 4.0. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 140–144, 2004.

[9] Daniela Chudá. Visualization in education of theoretical computer science. In *Proceedings of the 2007 international conference on Computer systems and technologies*, pages 1–6, 2007.

[10] Daniela Chuda, Jakub Trizna, and Peter Kratky. Android automata simulator. In *International Conference on e-Learning*, volume 15, page 80, 2015.

[11] Raphaël Clifford, Markus Jalsenius, Ashley Montanaro, and Benjamin Sach. The complexity of flood filling games. *Theory of Computing Systems*, 50(1):72–92, 2012.

[12] Joshua J Cogliati, Frances W Goosey, Michael T Grinder, Bradley A Pascoe, Rockford J Ross, and Cheston J Williams. Realizing the promise of visualization in the theory of computing. *Journal on Educational Resources in Computing (JERIC)*, 5(2):5–es, 2005.

[13] Joshua Joseph Cogliati et al. *Visualizing the pumping lemma for regular languages*. PhD thesis, Montana State University-Bozeman, College of Engineering, 2004.

[14] Simon Colton. The painting fool: Stories from building an automated painter. In *Computers and Creativity*, pages 3–38. Springer, 2012.

[15] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew Leaver-Fay, David Baker, Zoran Popović, et al. Predicting protein structures with a multiplayer online game. *Nature*, 466(7307):756–760, 2010.

[16] Joao Correia, Penousal Machado, Juan Romero, and Adrian Carballal. Evolving figurative images using expression-based evolutionary art. In *Proceedings of the 3rd International Conference on Computational Creativity*, pages 24–31, 2013.

[17] Pierluigi 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.

[18] Pierluigi Crescenzi, Emma Enström, and Viggo Kann. From theory to practice: Np-completeness for every cs student. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, pages 16–21, 2013.

[19] E del Acebo, I Boada, J Poch, F Prados, and J Soler. A web-based e-learning tool for theoretical computer science. In *INTED2009 Proceedings*, pages 572–579. IATED, 2009.

[20] Andreas Dengel. Seeking the treasures of theoretical computer science education: Towards educational virtual reality for the visualization of finite state machines. In *2018 IEEE international conference on teaching, assessment, and learning for engineering (TALE)*, pages 1107–1112. IEEE, 2018.

[21] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[22] Emma Enström. *On difficult topics in theoretical computer science education*. PhD thesis, KTH Royal Institute of Technology, 2014.

[23] Emma Enström and Viggo Kann. Iteratively intervening with the "most difficult" topics of an algorithms and complexity course. *ACM Transactions on Computing Education (TOCE)*, 17(1):1–38, 2017.

[24] Felix Erlacher et al. Pushdown automata simulator. *PhD, Institut für Informatik, Universität Innsbruck*, 2009.

[25] Agustín Esmoris, Carlos Iván Chesñevar, and Maráa Paula González. Tags: a software tool for simulating transducer automata. *International journal of electrical engineering education*, 42(4):338–349, 2005.

[26] Barry Fagin and Dino Schweitzer. Myturingtable: a teaching tool to accompany turing's original paper on computability. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*, pages 333–338, 2012.

[27] João Miguel Gago Gonçalves. *OCaml-Flat-An OCaml Toolkit for experimenting with formal languages theory.* PhD thesis, Faculty of Sciences and Technology, NOVA University of Lisbon, 2019.

[28] Eric Gramond and Susan H Rodger. Using jflap to interact with theorems in automata theory. In *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 336–340, 1999.

[29] Michael T Grinder. Animating automata: A cross-platform program for teaching finite automata. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 63–67, 2002.

[30] Michael T Grinder, Seong B Kim, Teresa L Lutey, Rockford J Ross, and Kathleen F Walsh. Loving to learn theory: Active learning modules for the theory of computing. *ACM SIGCSE Bulletin*, 34(1):371–375, 2002.

[31] Michael Thomas Grinder. *Active learning animations for the theory of computation.* Montana State University, 2002.

[32] Jianwen Guo, Hong Tang, Zhenzhong Sun, Song Wang, Xuejun Jia, Haibin Chen, and Zhicong Zhang. An improved shuffled frog leaping algorithm for assembly sequence planning of remote handling maintenance in radioactive environment. *Science and Technology of Nuclear Installations*, 2015, 2015.

[33] Derrall Heath and Dan Ventura. Before a computer can draw, it must first learn to see. In *Proceedings of the 7th International Conference on Computational Creativity*, pages 172–179. Citeseer, 2016.

[34] Cor A. J. Hurkens and Alexander Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.

[35] Anisse Ismaili, Tomoaki Yamaguchi, and Makoto Yokoo. Student-project-resource allocation: Complexity of the symmetric case. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 226–241. Springer, 2018.

[36] Jonathan Jarvis and Joan M Lucas. Understanding the universal turing machine: an implementation in jflap. *Journal of Computing Sciences in Colleges*, 23(5):180–188, 2008.

[37] Jesper Hasseriis Mohr Jensen, Miroslav Gajdacz, Shaeema Zaman Ahmed, Jakub Herman Czarkowski, Carrie Weidner, Janet Rafner, Jens Jakob Sørensen, Klaus Mølmer, and Jacob Friis Sherson. Crowdsourcing human common sense for quantum control. *Physical Review Research*, 3(1):013057, 2021.

[38] Nenad Jovanović, Dragiša Miljković, Srećko Stamenković, Zoran Jovanović, and Pinaki Chakraborty. Teaching concepts related to finite automata using comvis. *Computer Applications in Engineering Education*, 29(5):994–1006, 2021.

[39] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[40] Seungyon Kim and Seongbin Park. Teaching np completeness in secondary schools. In *Informatics in Schools: Local Proceedings of the 6th International Conference ISSEP 2013–Selected Papers*, page 35, 2013.

[41] Rajini Singh Kshatriya Jagannath et al. *Visualizing the minimization of a deterministic finite state automaton*. PhD thesis, Montana State University-Bozeman, College of Engineering, 2007.

[42] Anand Kulkarni. The complexity of crowdsourcing: Theoretical problems in human computation. In *CHI Workshop on Crowdsourcing and Human Computation*, 2011.

[43] James Lockwood and Aidan Mooney. Computational thinking in education: Where does it fit? a systematic literary review. *arXiv preprint arXiv:1703.07659*, 2017.

[44] Rita Pedroso Macedo. *OCaml-Flat on the Ocsigen framework*. PhD thesis, 2020.

[45] Nabanita Maji. *An interactive tutorial for NP-completeness*. PhD thesis, Virginia Tech, 2015.

[46] Jennifer McDonald. Interactive pushdown automata animation. In *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 376–380, 2002.

[47] Me. This is not a real paper. In *Proceedings of My Life (ML)*, 2020.

[48] William Patrick Merryman et al. *Animating the conversion of nondeterministic finite state automata to deterministic finite state automata*. PhD thesis, Montana State University-Bozeman, College of Engineering, 2007.

[49] Mostafa Mohammed, Clifford A Shaffer, and Susan H Rodger. Teaching formal languages with visualizations and auto-graded exercises. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, pages 569–575, 2021.

[50] Nelma Moreira and Rogério Reis. Interactive manipulation of regular objects with fado. *ACM SIGCSE Bulletin*, 37(3):335–339, 2005.

[51] Thomas L Naps. Jhavé: Supporting algorithm visualization. *IEEE Computer Graphics and Applications*, 25(5):49–55, 2005.

[52] Georgiana Mihaela NiÈ et al. Fasharpsim: A software simulator for deterministic and nondeterministic finite automata. *Journal of Electrical Engineering, Electronics, Control and Computer Science*, 2(2):13–20, 2017.

[53] Christian Pape. *Using interactive visualization for teaching the theory of NP-completeness*. Citeseer, 1997.

[54] Marzieh Parandehgheibi and Eytan Modiano. Robustness of interdependent networks: The case of communication networks and the power grid. In *2013 IEEE Global Communications Conference (GLOBECOM)*, pages 2164–2169. IEEE, 2013.

[55] Timothy Park. *Automata Simulators*. PhD thesis, California State University, Northridge, 2020.

[56] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, pages 1–10, 2008.

[57] Emil L Post. Recursively enumerable sets of positive integers and their decision problems. 1944.

[58] Matthew B Robinson, Jason A Hamshar, Jorge E Novillo, and Andrew T Duchowski. A java-based tool for reasoning about models of computation through simulating finite automata and turing machines. In *The proceedings of the thirtieth SIGCSE technical symposium on Computer science education*, pages 105–109, 1999.

[59] Susan H Rodger and Thomas W Finley. *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[60] Susan H Rodger, Eric Wiebe, Kyung Min Lee, Chris Morgan, Kareem Omar, and Jonathan Su. Increasing engagement in automata theory with jflap. In *Proceedings of the 40th ACM technical symposium on Computer science education*, pages 403–407, 2009.

[61] Clifford A Shaffer, Ville Karavirta, Ari Korhonen, and Thomas 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.

[62] Vinay Shekhar, Akshata Prabhu, Kavitha Puranik, Lakshmi Antin, and Viraj Kumar. Jflap extensions for instructors and students. In *2014 IEEE Sixth International Conference on Technology for Education*, pages 140–143. IEEE, 2014.

[63] Vinay S Shekhar, Anant Agarwalla, Akshay Agarwal, B Nitish, and Viraj Kumar. Enhancing jflap with automata construction problems and automated feedback. In *2014 Seventh International Conference on Contemporary Computing (IC3)*, pages 19–23. IEEE, 2014.

[64] Kshatriya Jagannath Rajini Singh. *VISUALIZING THE MINIMIZATION OF A DETERMINISTIC*. PhD thesis, Citeseer, 2007.

[65] Tuhina Singh, Simra Afreen, Pinaki Chakraborty, Rashmi Raj, Savita Yadav, and Dipika Jain. Automata simulator: A mobile app to teach theory of computation. *Computer Applications in Engineering Education*, 27(5):1064–1072, 2019.

[66] Michael Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1): 27–29, 1996.

[67] John Stasko, Albert Badre, and Clayton Lewis. Do algorithm animations assist learning? An empirical study and analysis. In *Proceedings of the INTERACT'93 and CHI'93 conference on human factors in computing systems*, pages 61–66, 1993.

[68] Maulana Muhamad Sulaiman, Romi Andrianto, and Muhamad Arief Yulianto. Mobile learning application for language and automata theory using android-based. *Jurnal Online Informatika*, 5(2):176–184, 2020.

[69] Quoc-Nam Tran. Interactive symbolic software for teaching formal languages, automata and beyond. *Journal of Computing in Colleges*, 22(4):130, 2007.

[70] Alan Turing. Turing machine. *Proc London Math Soc*, 242:230–265, 1936.

[71] Jeffrey D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10(3):384–393, 1975.

[72] Jan Van Leeuwen. *Handbook of theoretical computer science (vol. A) algorithms and complexity*. Mit Press, 1991.

[73] Rakesh M Verma. A visual and interactive automata theory course emphasizing breadth of automata. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*, pages 325–329, 2005.

[74] Luiz Filipe M Vieira, Marcos Augusto M Vieira, and Newton J Vieira. Language emulator, a helpful toolkit in the learning process of computer theory. *ACM Sigcse Bulletin*, 36(1):135–139, 2004.

[75] Marcel Walter, Robert Wille, Daniel Große, Frank Sill Torres, and Rolf Drechsler. Placement and routing for tile-based field-coupled nanocomputing circuits is np-complete (research note). *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 15(3):1–10, 2019.

[76] Joe Weinman. Cloud computing is NP-complete. In *Proc. Tech. Symp. ITU Telecom World*, pages 75–81, 2011.

[77] Timothy M White and Thomas P Way. jFast: A Java finite automata simulator. In *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, pages 384–388, 2006.

[78] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10, 2014.

[79] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[80] Hüsamettin Yüksel. Ganifa ng: The next generationof algorithm visualizations of finiteautomata, 2014.

[81] Leonardo Zambito. The traveling salesman problem: a comprehensive survey. *Project for CSE*, 4080, 2006.