Redux: An Interactive, Dynamic Knowledge Base for Teaching NP-completeness

Kaden Marchetti
marckade@isu.edu
Idaho State University
Pocatello, Idaho, USA

Andrija Ševaljević
andrijasevaljevic@isu.edu
Idaho State University
Pocatello, Idaho, USA

Alex Diviney
alexdiviney@isu.edu
Idaho State University
Pocatello, Idaho, USA

Caleb Eardley
eardcale@isu.edu
Idaho State University
Pocatello, Idaho, USA

Russell Phillips
russellphillips@isu.edu
Idaho State University
Pocatello, Idaho, USA

Rajiv Khadka
rajivkhadka@isu.edu
Idaho State University
Idaho Falls, Idaho, USA

Daniel Igbokwe
igbodani@isu.edu
Idaho State University
Pocatello, Idaho, USA

Paul Bodily
bodipaul@isu.edu
Idaho State University
Pocatello, Idaho, USA

ABSTRACT

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. We present an open-source web app called Redux that provides a dynamic interactive user interface atop a practical knowledge base 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 knowledge base extensible, allowing students to contribute and compare their reductions and solutions to those already available. Two surveys were administered, with respondents overwhelmingly indicating that Redux helped them to better understand mapping reductions; that they would prefer using Redux to solve similar problems manually; and that Redux makes learning NP-complete reductions more enjoyable. Redux is accessible online via https://redux.portneuf.cose.isu.edu/.

CCS CONCEPTS

• **Mathematics of computing → Discrete mathematics**; • **Theory of computation → Problems, reductions and completeness**; **Complexity theory and logic**; **Design and analysis of algorithms**; • **Human-centered computing → Web-based interaction**; **Visualization**; • **Applied computing → Digital libraries and archives**.

KEYWORDS

Computational Theory, Computer Science Education

ACM Reference Format:
Kaden Marchetti, Andrija Ševaljević, Alex Diviney, Caleb Eardley, Russell Phillips, Rajiv Khadka, Daniel Igbokwe, and Paul Bodily. 2024. Redux: An Interactive, Dynamic Knowledge Base for Teaching NP-completeness. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649217.3653544

1 INTRODUCTION

The ability to understand, recognize, and reduce nondeterministic polynomial-time (NP) complete problems remains an important skill for students of computer science (CS) [15, 21]. Despite this importance, significant shortcomings exist in the effectiveness with which the subject is taught in CS curricula due to the abstract nature inherent in learning NP-Completeness and other associated topics [17]. 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 [9]. The most notable of these tools is JFLAP [22], an automata visualization simulator that student surveys indicate makes learning such concepts easier and more enjoyable [23]. 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;

ITiCSE 2024, July 8–10, 2024, Milan, Italy

ACM ISBN 979-8-4007-0600-4/24/07
https://doi.org/10.1145/3649217.3653544

scheduling problems; nuclear core fuel reload optimization; route planning; 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. 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 [5].

A recent literature review [19] 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 [20, 22, 24]; context-free grammars [1, 22]; pushdown automata [11, 22]; pumping lemmas [6, 22]; and Turing machines [13, 22]. 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 [18]. 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 students to interact with or modify the example instances. Other contributions of theoretical NP-complete visualization tools have been abandoned or are no longer centered on NP-completeness [8]. 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, 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 a catalog of several canonical NP-complete problems, allows the user 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 knowledge base and provides students the educational opportunity to compare their reduction implementations and heuristic solution algorithms against those already present.

2 METHODS

Redux consists of two distinct parts: a knowledge base of computational problems, mapping reductions, and problem solvers accessible via a RESTful API; and a dynamic, interactive web-based graphical user interface (GUI). This two-pronged approach—developed with an eye toward both long-term sustainability and expansion—provides expansive capabilities to programmers and researchers who benefit from the efficiency of bypassing the GUI while simultaneously providing a more interactive, dynamic medium for beginners, non-programmers, and instructors. The GitHub repository for the backend is available at https://github.com/ReduxISU/Redux and the repository for the frontend is at https://github.com/ReduxISU/Redux_GUI. Full API documentation is available at https://api.redux.portneuf.cose.isu.edu/swagger/index.html.

2.1 The Knowledge Base and RESTful API

Our initial goal in creating Redux was to build a tool for teaching about the theory of NP-completeness. A common approach to proving that a problem is NP-complete requires proving, first, that there exists a polynomial-time *mapping reduction* from an existing NP-complete problem and, second, that there exists a polynomial-time algorithm (called a verifier) to *verify* valid solutions. Beyond featuring the problems themselves, Redux's knowledge base also includes repositories of mapping reductions and verifiers. Including a repository of *solution* algorithms caters to users interested in algorithms for solving NP-complete problems (a common topic often covered separately in an algorithms course).

With teaching NP-completeness as our initial focus, Redux primarily features NP-complete decision problems. The backend also includes several (NP-hard) optimization algorithms, and we plan to expand to more problems and algorithms from the classes P, NP-hard, etc. Table 1 outlines the problems, reductions, solvers, and visualizations currently available in Redux.

2.1.1 Problems. Both problem and problem instance strings are defined in Redux using traditional discrete math notation. For example, the 3SAT problem in Redux is defined as $3SAT = \{\phi | \phi$ is a satisfiable Boolean formula in 3CNF$\}$. We generally assume that users will have a previous introduction to concepts such as a "satisfiable Boolean formula" or "3CNF"; however, for novice learners, this definition is augmented with hyperlinks (e.g., to Wikipedia) to find such background information. In many instances, this background information will be important to understanding reductions and solutions. In the case of 3SAT, this background information includes knowledge that: a Boolean formula is comprised of literals where a *literal* is a negated or non-negated Boolean variable (as in x or $\overline{x}$); a *clause* is several literals connected with disjunctions, e.g., $(x_1 \vee x_2 \vee \overline{x_3})$; *conjunctive normal form* (CNF) is defined as a conjunction of clauses; and k-CNF formula requires that each clause in the formula has exactly k literals.

Each problem implementation also provides an example instance that demonstrates the expected syntax for defining an instance of the problem. As a plethora of problems share common input data types (e.g., graphs, sets, boolean formulae, etc.), we use a standardized parsing method to simplify implementing new problems, reductions, and solutions. In Redux, problem instances are sent to the backend as a string where they are parsed using regular expressions (regex) into data structures as specified by the problem definition.

2.1.2 Reductions. Reductions in Redux are currently limited to mapping reductions. A mapping reduction from a problem A to a problem B is a function f that transforms an arbitrary instance a of A into an instance $f(a)$ of B such that a is a valid instance of A

Problems	Polynomial-time Reduction(s) to	Solution Algorithm(s)	Problem Visualizer	Polynomial-time Verifier
0-1 INTEGER PROGRAMMING		Brute-force		x
3D MATCHING		Brute-force, Hurkens Schrijver* [14]		x
3SAT	0-1 INTEGER PROGRAMMING, 3D MATCHING, CLIQUE*, GRAPH COLORING	Backtracking [25]	x	x
CLIQUE	VERTEX COVER*	Bron-Kerbosch [4], Brute-force	x	x
CLIQUE COVER		Brute-force	x	x
CUT		Brute-force	x	x
DIRECTED HAMILTONIAN		Brute-force	x	x
EXACT COVER	SUBSET SUM	Brute-force, Dancing Links [16]		x
FEEDBACK ARC SET		Brute-force, Naïve approximation* [12]	x	x
FEEDBACK NODE SET		Brute-force	x	x
GRAPH COLORING	CLIQUE COVER, EXACT COVER, SAT	Brute-force, DSatur* [3]	x	x
HAMILTONIAN		Brute-force	x	x
HITTING SET	EXACT COVER	Brute-force		x
INDEPENDENT SET	CLIQUE	Brute-force		x
JOB SEQUENCING		Brute-force		x
KNAPSACK		Brute-force		x
PARTITION	WEIGHTED CUT	Brute-force		x
SAT	3SAT	Brute-force		x
SET COVER		Brute-force		x
STEINER TREE		Brute-force	x	x
SUBSET SUM	KNAPSACK, PARTITION	Brute-force		x
TRAVELING SALESPERSON		Branch and Bound, Brute-force	x	x
VERTEX COVER	FEEDBACK ARC SET*, FEEDBACK NODE SET, SET COVER	Brute-force, Greedy* [7]	x	x
WEIGHTED CUT		Brute-force	x	x

Table 1: *Current functionality in Redux*. A list of problems, reductions, solvers, and visualizations available in Redux. Reductions with implemented visualizations are asterisked. Solution algorithms for NP-hard optimization problems are also asterisked.

(written $a \in A$) iff $f(a) \in B$. NP-completeness theory guarantees that polynomial-time mapping reductions exist between any two NP-complete problems; however, discovering such reductions is non-trivial. As reductions is one of the most challenging topics to teach in computational theory, our purpose in creating a catalog of reductions is that it will not only help to clarify the topic in students' minds but also help students see practical applications for reductions and even inspire them to discover and contribute new reductions and reduction applications. As an example, one feature already implemented in Redux is that because mapping reductions are transitive, Redux automatically detects and facilitates reduction between problems via transitivity. Thus if mapping reductions f and g are implemented in Redux reducing problem A to B and B to C respectively, then Redux has functionality to automatically reduce A to C via a composition of these implemented functions (for an arbitrary number of transitions). The reduction of 3SAT to VERTEX COVER via CLIQUE is an example currently available in Redux (see Figure 1).

Besides the theoretical value in proving problems to be NP-complete, reduction can have practical value, enabling one problem to be solved by a host of algorithms developed to solve entirely different problems. In future work, we envision seeking out novel NP-complete problems in industrial or research settings and applying this feature of Redux to quickly experiment with a variety of existing solution algorithms via the implementation of only one or a few reductions to problems already in Redux.

2.1.3 Solvers and Verifiers. Whereas the long-term goal is to provide an exhaustive catalog of solution algorithms to a spectrum of computational problems (including, e.g., canonical polynomial-time algorithms taught in most algorithms courses), Redux currently implements solvers for 24 NP-complete problems, the majority being brute-force exact solvers (i.e., a solver guaranteed to return a correct

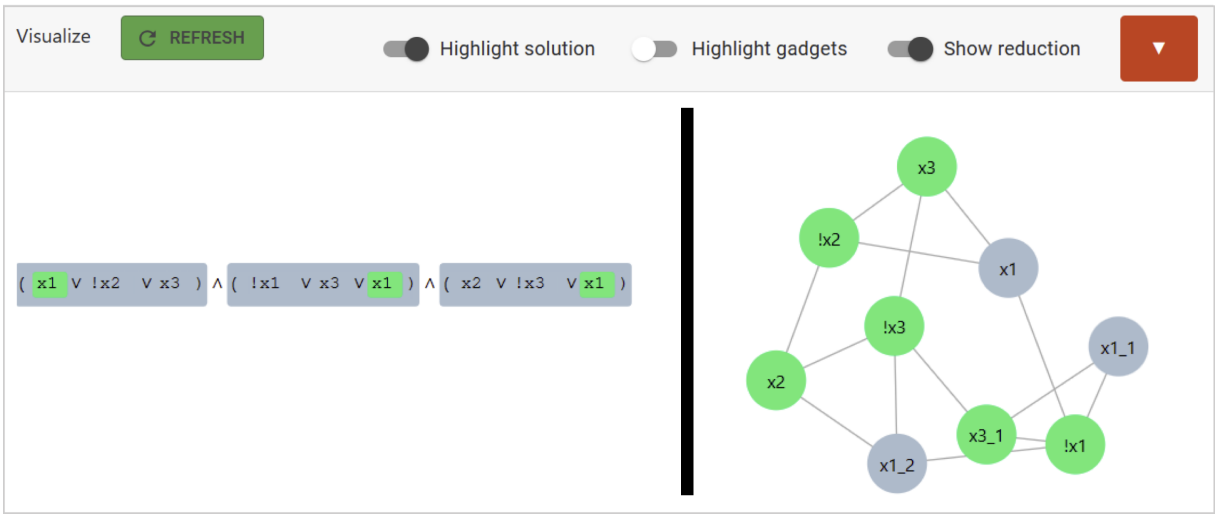

Figure 1: *Transitivity of reductions.* Because Redux implements both the 3SAT to CLIQUE (where K = 3) and the CLIQUE to VERTEX COVER (where K = 6) reductions, a reduction from 3SAT to VERTEX COVER is derived automatically. A solution is highlighted in green.

solution). Because exact solvers become inefficient for increasingly large inputs (particularly on NP-complete problems), of greater interest are the implementation of heuristic and approximation algorithms, several of which have already been implemented in Redux (see Table 1).

We have implemented a polynomial-time verifier for solutions to each problem that allows users to check whether a particular candidate solution to a problem instance is valid. The verifier serves a pedagogical purpose as a defining characteristic of an NP-complete problem.

2.1.4 RESTful API. Redux was implemented in C# using the React web framework and a RESTful API backend. The API allows users to access Redux's knowledge base in two distinctly powerful ways. First, the API supports a dynamic, interactive, web-based GUI for instructors to use as a pedagogical resource (e.g., demonstrations, homework assignments, generating visualizations, etc.), students to use as a learning resource, and non-CS users interested, e.g., in obtaining solutions to complex combinatorial and optimization problems. Second, the API can be accessed directly for users interested in automation, incorporation of Redux's algorithms into custom software, or meta-level research of problems, reductions, or solutions present in Redux. Any (validated) contribution on the backend is made automatically available via the API. This design pattern allows students focusing on algorithms to make public contributions without needing to modify the frontend.

2.2 The Dynamic, Interactive Web-Based GUI

The Redux knowledgebase—without an effective user interface—has limited pedagogical efficacy. A second, equally important contribution is Redux's dynamic, interactive, web-based GUI. This interface—available at https://redux.portneuf.cose.isu.edu/ and built using React Native and Material UI (MUI)—features five collapsable panes whose content changes dynamically based on user interaction. The *Problem* pane includes selection of a problem A and input field for problem instance a. The *Reduce* pane includes selection of a problem B to which reductions from A have been implemented, selection of a particular reduction algorithm f reducing A to B, and an output field for the instance $f(a)$. The *Visualize* pane includes visualization of a with options to highlight a solution, to highlight instance gadgets (discussed below), and to simultaneously visualize $f(b)$ (with its solution and gadgets). The *Solve* includes selection of a solver h for A and an output field for the solution computed by $h(a)$. The *Verify* panes includes selection of a verifier v for A, an input field for a certificate (i.e., candidate solution) c, and an output field for the result computed by $v(c)$. Each pane provides an information button with access to details such as definitions, citations, contributors, and hyperlinks to find more information.

2.2.1 Problem Visualization and Solution Highlighting. With thousands of unique problems in computer science—not to mention their specific applications in real-world contexts—visualization procedures must be implemented separately for each problem in the knowledgebase. Redux currently allows for one visualization implementation per problem. The solution highlighting feature highlights the solution within the visualized instance as computed by the solver selected in the *Solve* pane (see Figure 1). Visualizations and solution highlighting for 13 problems have so far been implemented using the D3 JavaScript library [2] (see Table 1).

2.2.2 Reduction Visualization and Gadget Highlighting. Given the pedagogical goals motivating Redux, each reduction also requires a custom visualization to effectively elucidate the mechanisms by which the reduction works. Although a visualization might exist for a problem B, a separate visualization must be implemented for each *reduction* to B. Reductions generally operate on the notion of a *gadget*, a subunit of the reduced problem instance that simulates one of the fundamental units of the original problem instance. The

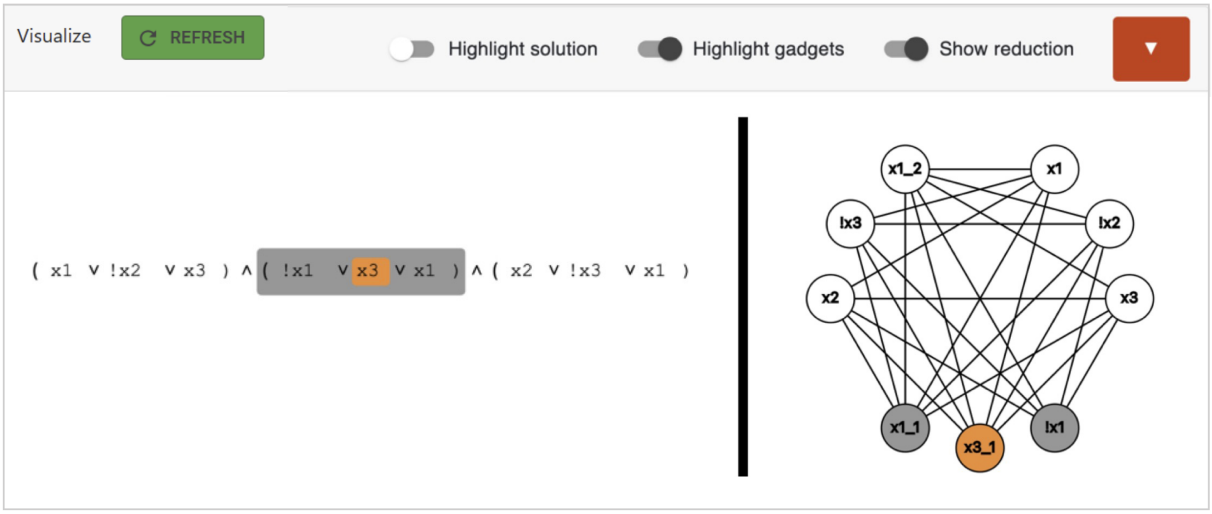

Figure 2: *Dynamic, interactive visualization of NP-complete mapping reductions.* Shown is the result of reducing an instance of 3SAT (left) to an equivalent CLIQUE instance (right). Redux allows users to select elements of one instance (e.g., the Boolean literal in orange on the left) and see its analogous element (called a *gadget*) highlighted in the other instance (the orange node on the right).

gadget highlighting feature allows a user to select units in a visualized instance and see the corresponding gadgets highlighted in the reduced instance and vice versa (see Figure 2). Redux currently implements 3 reduction visualizations (see Table 1).

2.3 Open-Source Contributing

Redux is an open-source platform welcome to contributions. Instructors and students submit merge requests on GitHub, with approved changes automatically updating the live website. It also offers utility functions, such as parsing problem definitions and validating instance formatting, to streamline contributions, and prioritize algorithmic implementation.

3 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.

3.1 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, groups were separated and surveyed to assess their understanding 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. For example, given a simple 3SAT or CLIQUE instance, respondents were asked to identify which of several options was a valid solution. The second assessed the ability to extend theoretical concepts to new scenarios. For example, given a 3SAT instance, respondents were asked to identify the CLIQUE instance resulting from a particular mapping reduction. Respondents were asked to identify how reductions can be applied both to prove NP-completeness and to solve NP-complete problems indirectly. Overall, the control group performed slightly better (63.3 ± 29.6% versus 60.0 ± 16.1%) on the practical application; the test group performed better (77.3 ± 23.7% versus 68.3 ± 31.9%) on the theoretical implication.

Careful consideration of the results from the pilot study suggested several important changes that needed to be made both to improve Redux's 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 and reliable.

3.2 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.

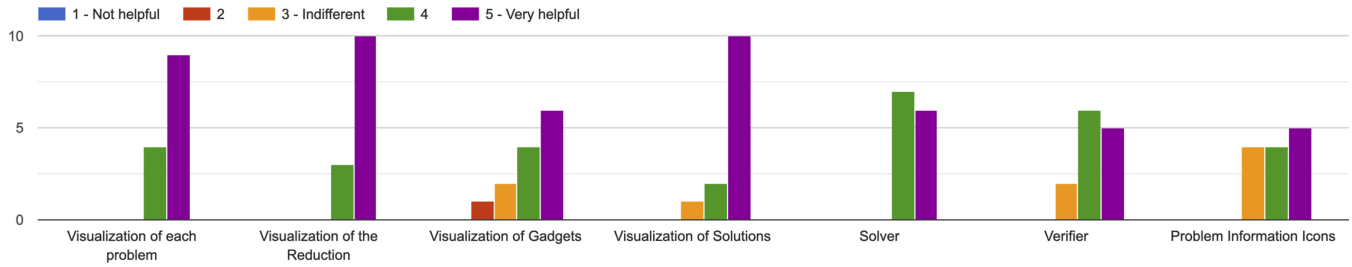

Figure 3: *The Most Helpful Redux Features.* Survey two asked participants to rank how helpful each Redux feature was in learning the underlying concepts. The four highest-ranked features all relate to the visualization of problems, reductions, gadgets (Figure 2), and solutions (Figure 1).

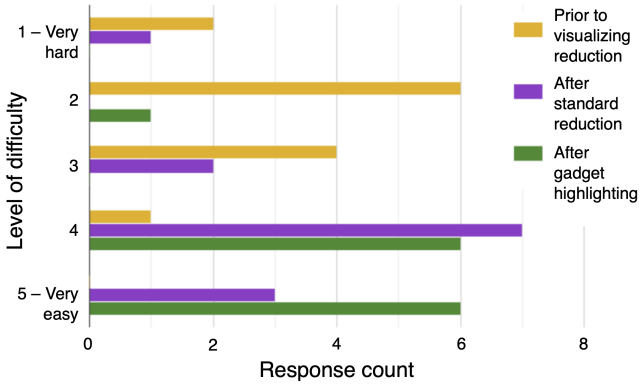

Figure 4: *Understanding the Reduction at each visualization step.* Survey results suggest that both visualization and gadget highlight contribute to greater understanding.

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 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 DISCUSSION AND CONCLUSION

Redux provides a pedagogical tool that can be used in a wide array of core CS curricular areas. Some possible examples include: assigning students in a discrete structures course to analyze and experiment with the string notations of the 3SAT problem definition and its problem instances within Redux; using gadget highlighting on the 3SAT to CLIQUE reduction to demonstrate the concept of proof by construction; requiring computational theory students to research, implement, and contribute a new NP-complete problem, reduction, and/or solution algorithm; requiring students in an algorithms course to implement and contribute a heuristic solver for an NP-complete problem of their choosing; assigning students in a graphics course to implement and contribute a missing problem or reduction visualization; assigning students a full-stack development project involving contributions on the backend, the frontend, and the API.

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 practitioners alike, for facilitating improved comparative analysis of algorithmic solutions, and for exploring ways to 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 backend 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. Integrating the class P in Redux could also be used to demonstrate P algorithms, for example, Dijkstra's algorithm [10] for the shortest path in a weighted, directed graph.

Redux represents a highly extensible framework. As an open-source application, we have opened the application's codebase 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.

REFERENCES

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

[2] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* 17, 12 (2011), 2301–2309.

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

[4] Coen Bron and Joep Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

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

[6] Joshua Joseph Cogliati et al. 2004. *Visualizing the pumping lemma for regular languages*. Ph. D. Dissertation. Montana State University-Bozeman, College of Engineering.

[7] TH Cormen, EC Leiserson, LR Rivest, and C Stein. 2001. Section 35.1: The vertex-cover problem. *Introduction to Algorithms (2nd ed.) MIT Press and McGraw-Hill* (2001), 10241027.

[8] Pierluigi Crescenzi. 2010. Using AVs to explain NP-completeness. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*. 299–299.

[9] Pierluigi Crescenzi, Emma Enström, and Viggo Kann. 2013. 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*. 16–21.

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

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

[12] Guy Even, Joseph Naor, Baruch Schieber, and Madhu Sudan. 1998. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica* 20 (1998), 151–174.

[13] Barry Fagin and Dino Schweitzer. 2012. 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*. 333–338.

[14] Cor A. J. Hurkens and Alexander Schrijver. 1989. 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 (1989), 68–72.

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

[16] Donald E Knuth. 2000. Dancing links. *arXiv preprint cs/0011047* (2000).

[17] Michael Luu, Matthew Ferland, Varun Nagaraj Rao, Arushi Arora, Randy Huynh, Frederick Reiber, Jennifer Wong-Ma, and Michael Shindler. 2023. What is an algorithms course? Survey results of introductory undergraduate algorithms courses in the US. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 284–290.

[18] Nabanita Maji. 2015. *An interactive tutorial for NP-completeness*. Ph. D. Dissertation. Virginia Tech.

[19] Kaden Marchetti. 2023. *Redux: An Interactive, Dynamic Tool for Learning NP-Completeness and Mapping Reductions*. Ph. D. Dissertation. Idaho State University.

[20] Georgiana Mihaela NiÈ et al. 2017. FASharpSim: A Software Simulator for Deterministic and Nondeterministic Finite Automata. *Journal of Electrical Engineering, Electronics, Control and Computer Science* 2, 2 (2017), 13–20.

[21] Joint Task Force on Computing Curricula. 2023. *ACM Computer Science Curricula 2023*. Technical Report. Association for Computing Machinery. https://www.acm.org/binaries/content/assets/education/cs2023-final-report.pdf

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

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

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

[25] Michael Sipser. 1996. Introduction to the Theory of Computation. *ACM Sigact News* 27, 1 (1996), 27–29.