Visualizing the 3SAT to CLIQUE Reduction Process

Kaden Marchetti
Department of Computer Science
Idaho State University
Pocatello, ID, USA
marckade@isu.edu

Paul Bodily
Department of Computer Science
Idaho State University
Pocatello, ID, USA
bodipaul@isu.edu

2022 Intermountain Engineering, Technology and Computing (IETC) | 978-1-6654-8653-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/IETC54973.2022.9796851

***Abstract*—As the needs of industry and research change, computer science education has gradually begun to incorporate a more intense concentration on theoretical discussions surrounding, among others, automaton theory, NP-Completeness, and reduction theory. These concepts have repeatedly been ranked as some of the most difficult concepts in computer science education by students. This is largely due to the abstract nature of the material along with the perceived lack of industry applicability. In order to bridge this pedagogical gap, this paper showcases a visual tool developed to teach students of computational theory the reduction process. The application allows students to dynamically generate the reduction between *3SAT* and *CLIQUE*. The tool is developed to allow for any *3SAT* input. This allows students to explore how changes applied to an instance of one problem impact are reflected in the reduced instance of a second problem. Results are presented and we demonstrate a working implementation of the visualization tool. Further work still needs to be done to expand on intractability and implement additional problems.**

***Index Terms*—Computational theory, 3SAT, CLIQUE, visualization, computer science pedagogy**

I. 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 [1]. 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 incredibly 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. 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.

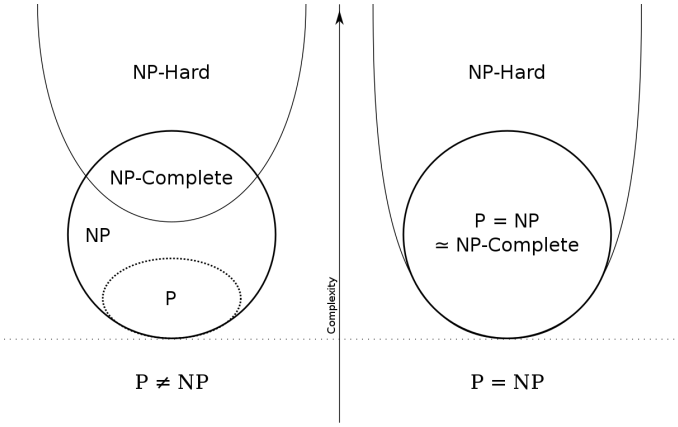

Fig. 1. An Euler diagram of complexity classes for P, NP, and NP-Complete. The right side is valid under the assumption that P equals NP while the left relies on the assumption that P does not equal NP. The vertically of each class correlates to its complexity. Most computer scientists believe P $\neq$ 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 [2]. 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. Studies have repeatedly shown that the use of visualization technology to teach complex topics can help relay complex topics like computational theory

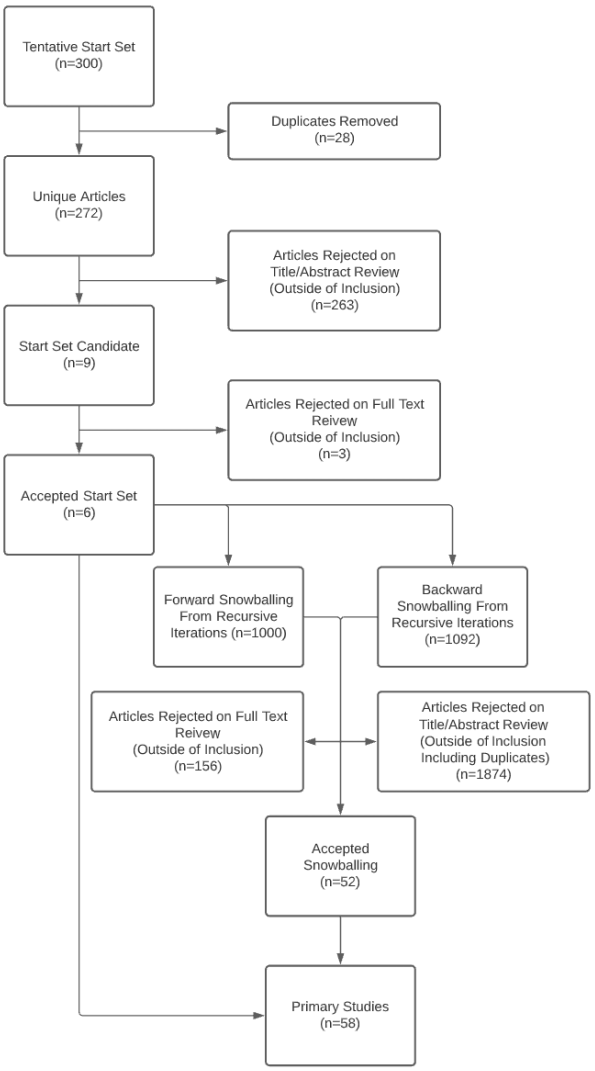

Fig. 2. A breakdown of a systematic mapping study performed on interactive, visualization technology for computational theory. A total of 2492 papers are examined using rules prescribed by Peterson et al. We find that there is a lack of visual technology focused on reduction theory and NP-Completeness.

[3]. Using Two.js and a standard web framework stack, we have created a interactive visualization tool to allow students to visualize the reduction process and appreciate NP-Completeness. While still in its infancy, the results in this paper showcase a strong foundation of growth towards the ultimate goal of providing a public facing API and GUI to visualize polynomial mapping reductions for pedagogical benefit.

II. 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 [4] while T.L. Naps finds that they provide an effective pedagogical tool for computer science educators [3]. 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. [5]. 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.

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

Fig. 3. Example of a valid 3SAT formula ϕ with three literals and three clauses.

A total of 2492 papers were examined in this study. Of these, the most notable in relation with this project come from White [6] and Rodger [7] who develop Java widgets to simulate other computational theory concepts. Most notable, however, is that we find a lack of visual technologies focused on reduction theory and NP-Completeness.

III. 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 [8], 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.

A. 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.

B. 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. 4.

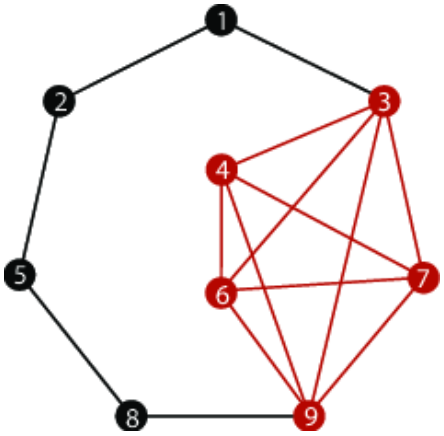

Fig. 4. Example of a 5-CLIQUE in a graph as shown in red between nodes 3, 4, 6, 7, and 9. [9]

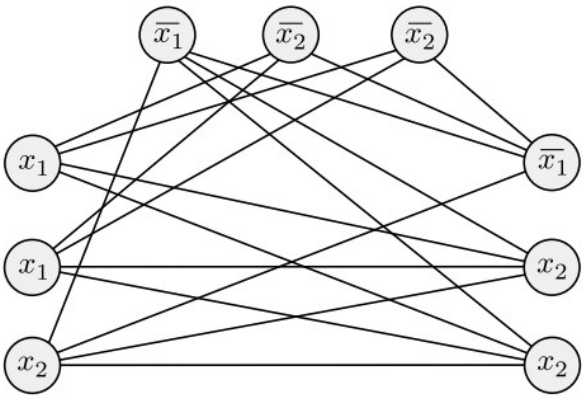

Fig. 5. A valid construction from $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_2 \vee x_2)$. The nodes are edges are formulated via $f(\phi)$ [8].

C. 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. 5.

IV. 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.

A. 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.

B. 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.

C. 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.

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

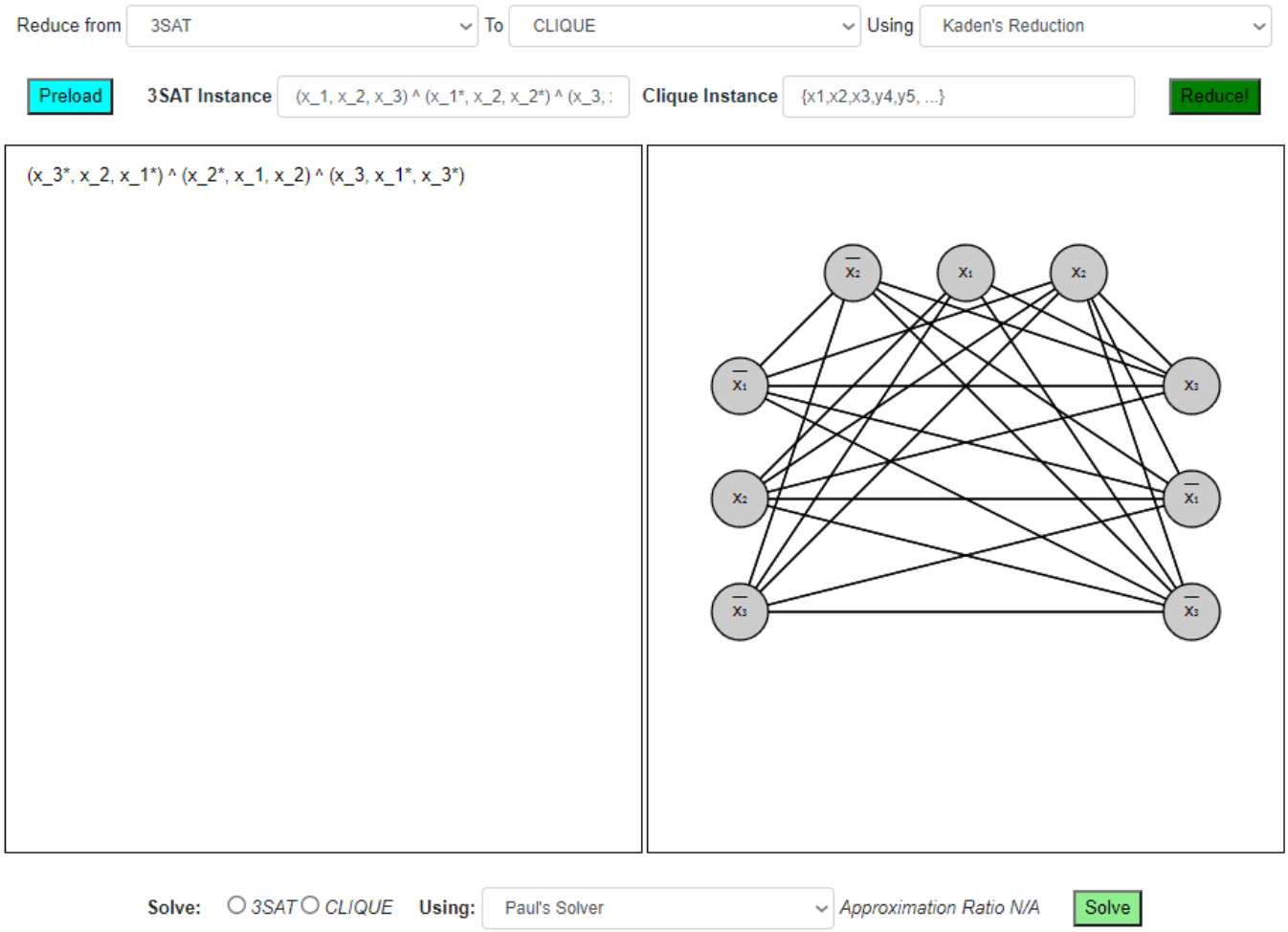

Fig. 6. A Sample output of the visualization application. At the top, the user can select a problem to reduce from, reduce to, and a polynomial mapping function that facilitates the reduction. Users can manually enter a problem instance or pre-load an existing configuration before reducing. The CLIQUE 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.

Fig. 6 showcases a graph that has gone through every stage of our reduction function and represents a 3SAT problem.

V. 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.

VI. 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*.

As the first step towards a powerful pedagogical tool for helping students understand NP-Completeness and polynomial mapping reductions, we are confident that the results thus far represent a bright future for computational theory education.

REFERENCES

[1] E. Enström, "On difficult topics in theoretical computer science education," Ph.D. dissertation, KTH Royal Institute of Technology, 2014.

[2] C. A. Shaffer, V. Karavirta, A. Korhonen, and T. L. Naps, "Opendsa: beginning a community active-ebook project," in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, 2011, pp. 112–117.

[3] T. L. Naps, "Jhavé: Supporting algorithm visualization," *IEEE Computer Graphics and Applications*, vol. 25, no. 5, pp. 49–55, 2005.

[4] J. Stasko, A. Badre, and C. 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*, 1993, pp. 61–66.

[5] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic mapping studies in software engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*, 2008, pp. 1–10.

[6] T. M. White and T. P. Way, "jFast: A Java finite automata simulator," in *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, 2006, pp. 384–388.

[7] S. H. Rodger and T. W. Finley, *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[8] M. Sipser, "Introduction to the theory of computation," *ACM Sigact News*, vol. 27, no. 1, pp. 27–29, 1996.

[9] P. Bloem and S. de Rooij, "A tutorial on mdl hypothesis testing for graph analysis," *arXiv preprint arXiv:1810.13163*, 2018.