2016

VISUALIZING NP-COMPLETENESS THROUGH CIRCUIT-BASED WIDGETS

Steven R. Vegdahl
University of Portland, vegdahl@up.edu

VISUALIZING NP-COMPLETENESS THROUGH CIRCUIT-BASED WIDGETS *

Steven R. Vegdahl
University of Portland
5000 N. Willamette Blvd.
Portland, OR 97203
503 943-7215
vegdahl@up.edu

ABSTRACT

The concept of intractability is an important one in a CS curriculum. Students should know that some problems are computationally infeasible; ideally, they should have the ability to examine a problem and determine whether it is intractable. This paper presents two widgets that help students gain better intuition about NP-completeness. Each maps circuit satisfiability to a target problem in a visually intuitive way.

1. INTRODUCTION

The concept of intractability is an important one for a computer scientist to understand [3,5]. The notion that there are many real-world problems that are not likely to have an efficient solution should be in a person's mind when attempting to solve a problem. Determining that a problem is intractable leads a designer change the approach to the problem—for example, by using an approximation solution [3,5].

Several concepts must be understood in order to understand NP-completeness. The first is the concept of a problem transformation: a valid transformation is one where a solution to the target problem implies a solution to the original, and vice versa.

The second is that when polynomial-time transformations are combined, the result is yet another polynomial-time transformation. Once a problem is proven to be NP-complete, it is a valid starting point for future NP-completeness proofs.

Finally, there is the idea of a "first" NP complete problem—one to which any problem in NP can be (polynomially) transformed. Ideally, a student will understand why the problem is NP-complete, rather than simply accepting on faith.

CIRCUIT SATISFIABILITY (CSAT)

Cook [2] used boolean satisfiability (SAT) as the problem from which other problems would be (directly or indirectly) shown to be intractable. A number of textbook authors have followed his lead, and have used SAT as a starting point [7,12].

2.1 Pedagogical Problems with SAT

The use of SAT gives rise to several pedagogical difficulties. First, a student must be familiar with Turing machines, which are not particularly intuitive. Second, the transformation is rather cumbersome. A student might be able step through the proof, but without a solid understanding. (Several textbook authors simply state Cook's theorem without proof, apparently considering it too tedious [1,6,10].) A third problem is that SAT is based on symbol manipulation, so the transformations tend to be non-intuitive and difficult to visualize.

2.2 Approaches with Circuit Satisfiability

CSAT is easy to state: given a single-output combinational circuit consisting of standard boolean gates, is there any setting of its inputs that causes its output to be true?

Several textbook authors use CSAT as the starting point [3,4,9]. This has a number of advantages, including:

* It is based on elementary concepts of logic design. Computer science students typically learn this material early in their undergraduate careers, and generally understand that logic gates are a basis for universal computation. They understand how they can be combined to produce adders, comparators—and ultimately, general-purpose computers [10].

* Most of the commonly studied problems that are in NP can be checked with a combinational circuit that is very straightforward to envision. Figure 1 sketches an example circuit that checks that its two 16-bit input values are both greater than one, and that their product is 14353. If there exist values for these free inputs that cause the output bit to be one, then these input values represent two non-trivial factors of 14353. In other words, the circuit is satisfiable if and only if there exist such factors.

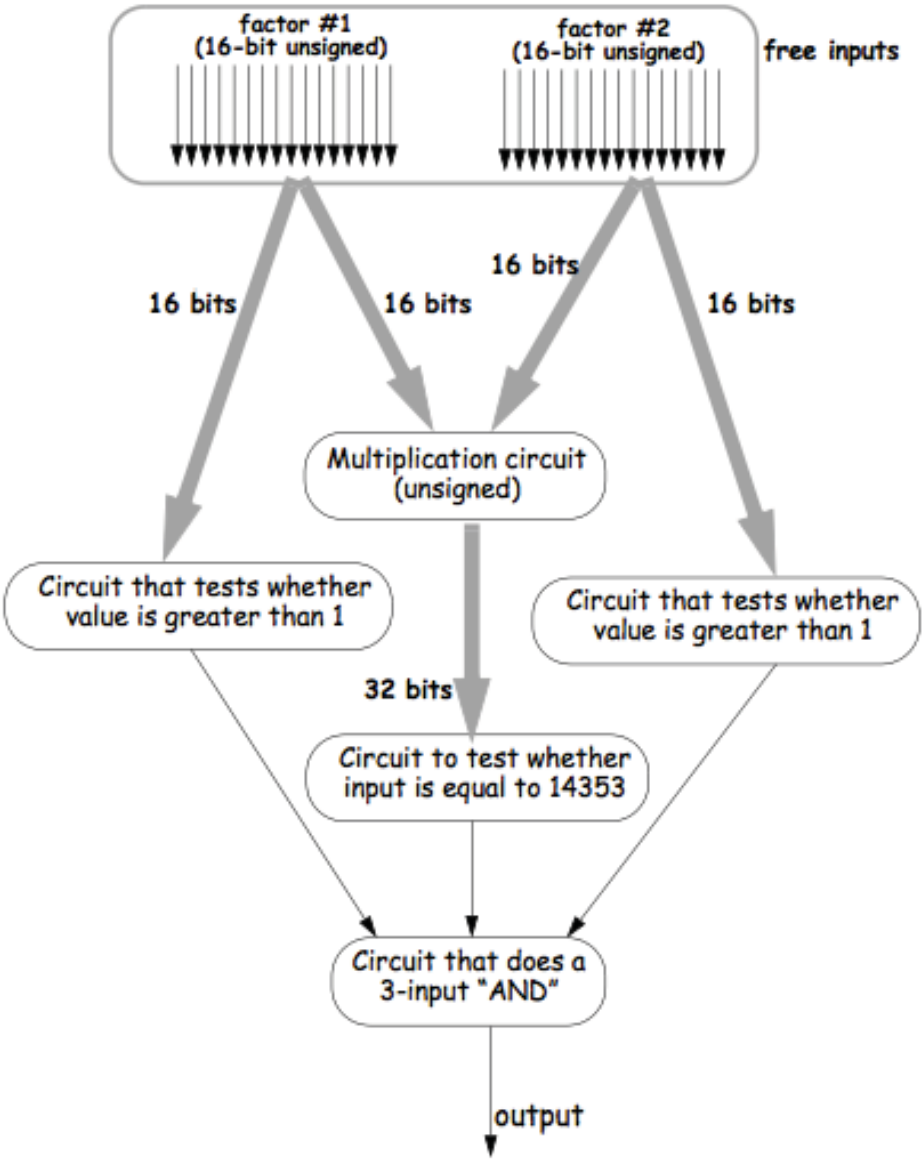

Figure 1. Sketch of a combinational circuit to test if 14353 has non-trivial factors

2.3 Direct Transformation from CSAT

CSAT-based presentations of NP-completeness in textbooks [3,4,9] all appear to take the following approach:

* Begin by showing that any problem in NP can be polynomially transformed to CSAT.

* Immediately transform CSAT to SAT. This is quite straightforward: for example, a logic gate representing C = A NAND B can be modeled with the three boolean clauses $\neg A \vee \neg B \vee \neg C$, $A \vee C$ and $B \vee C$.

* Use SAT as the starting point for all other problem transformations.

While this approach is valid mathematically, it is not as appealing pedagogically because the visual structure of the circuit is immediately lost when the problem is converted to symbols. The goal of this work is that of developing transformations that preserve intuition by maintaining the "shape" of the circuit in the target-problem.

2.4 Using NAND Gates

Textbooks that use CSAT as a starting point generally use AND, OR, and NOT (and sometimes equality) as their primitive gate types [3,4,9]. This complicates problem transformations from CSAT because a mapping must be created from each such gate type to the target domain. This paper employs NAND gates, so that only one transformation is needed. Students who have taken a logic design course understand that NAND is a universal gate. There are straightforward mappings from NAND-based CSAT to a number of common NP-complete problems; two of these are discussed in Section 3.

3. TRANSFORMATIONS FROM CSAT

It was argued above that students gain better intuition if they can visualize the transformation to the target problem directly from a checking circuit. This section presents NP-completeness widgets that can be used to transform (two-input) NAND-based CSAT directly into two such problems: graph three-colorability and vertex cover [3,5]. The circuit in Figure 2 is used to illustrate each transformation.

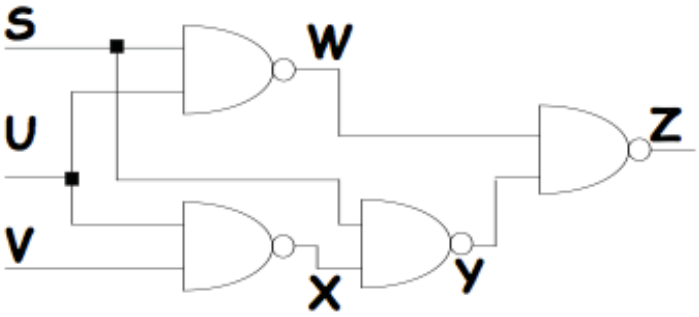

Figure 2. Example circuit, with wires labeled

Note that it is also straightforward to transform this circuit to SAT:

$(\neg S \vee \neg U \neg \vee W) \wedge (S \vee W) \wedge (U \vee W) \wedge (\neg U \vee \neg V \vee \neg X) \wedge (U \vee X) \wedge (V \vee X) \wedge$

$(\neg S \vee \neg X \vee \neg Y) \wedge (S \vee Y) \wedge (X \vee Y) \wedge (\neg W \vee \neg Y \vee \neg Z) \wedge (W \vee Z) \wedge (Y \vee Z) \wedge Z$

Each set of three clauses models one of figure's two-input NAND gates; the final 'Z' term ensures that the circuit's output is true.

3.1 Graph Three-Colorability (G3C)

Graph three-colorability (G3C) is a classic NP-complete problem. Each vertex of an undirected graph is to be colored with one of three colors, where connected vertices are not allowed to have the same color [3,5].

In this transformation from CSAT to G3C, one color represents the value false, and two colors represent the value true. (This is not unlike the C programming language [8], where one value represents false, and many values represent true.) It uses:

* One three-vertex reference clique, whose vertices are labeled T0, T1 and F. The color of vertex F represents false; T0 and T1 each represent true.

* A nine-vertex widget that models a two-input NAND gate (Figure 3).

* A two-vertex widget that models a wire that connects two gates (Figure 4).

The vertex that represents the output of the entire circuit is connected to the F-vertex in the reference clique. This ensures that any legal coloring must correspond to a circuit whose output value is *true*.

NAND-gate widget: The widget that models a NAND gate is shown in Figure 3; the reference clique connects to each NAND-widget in six locations. The figure is labeled to illustrate vertex A having the color T0, and vertex B having the color F; all subsequent colorings are forced (by the three-color constraint), resulting in vertex C having the color T0. It is not difficult to verify that remaining eight combinations of colors for A and B all produce results that are consistent with the NAND function.

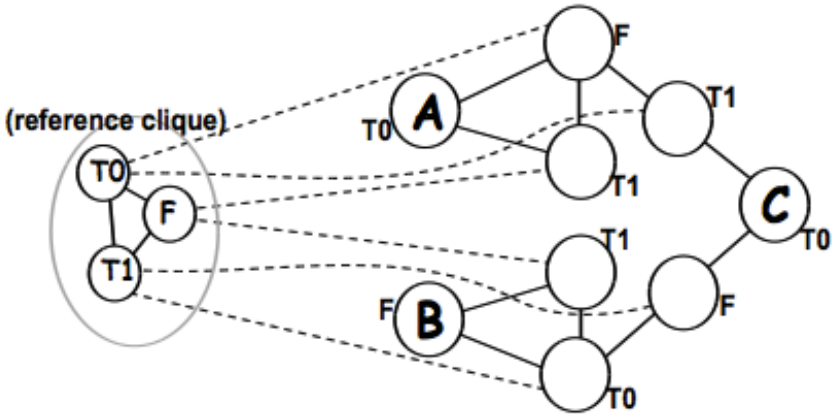

Figure 3: G3C widget for C = A NAND B

Wire widget: Modeling a wire is straightforward, as illustrated in Figure 4. In any three-coloring, vertices X and Y (not part of the wire-widget) must have the same color. (If not, a fourth color is necessary because neither can have the same color as either interior vertex.)

Figure 4: G3C widget for a circuit wire
(connects X to Y)

With widgets defined both for wires and for NAND gates, the circuit in Figure 2 can now be modeled. The resulting graph is shown in Figure 5.

If there is a legal three-coloring of the graph, a solution to the circuit can be found by mapping the colors of the vertices in the graph to the corresponding wires in the circuit. Because the vertex corresponding to the circuit's output wire is connected to the F-vertex in the reference clique, the circuit's output must be true, corresponding to T0 or T1. Conversely, if the circuit is satisfiable, the input-settings for a solution to the circuit can be used to easily generate a three-coloring for the graph.

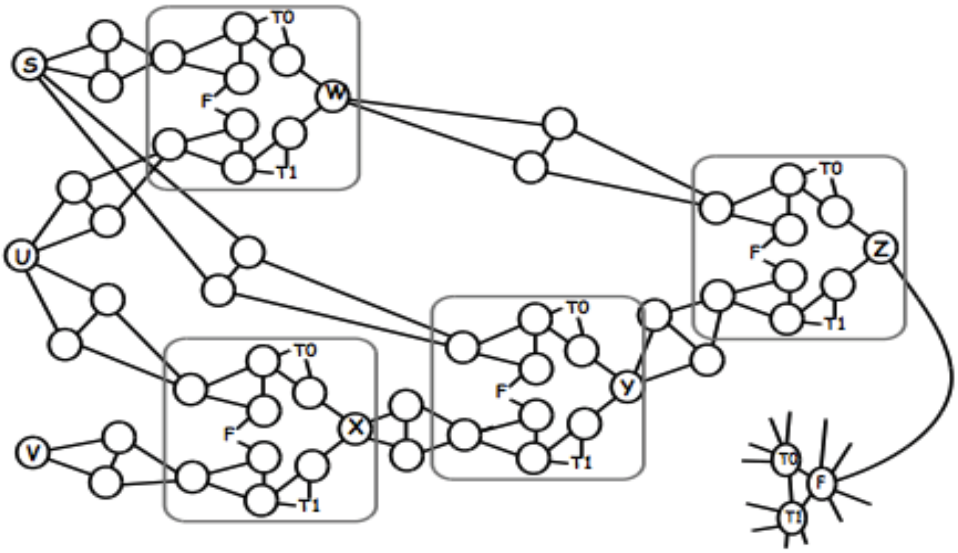

Figure 5: Example circuit, transformed to G3C

3.2 Vertex Cover (VC)

Vertex cover (VC) is another classic NP-complete problem [3,5]. Here, an undirected graph is given, along with a positive integer, N. The problem is solvable if a subset of N of the graph's vertices can be marked so that each edge is covered (that is, touching at least one marked vertex). The transformation from CSAT to VC uses:

* A nine-vertex widget that models a NAND gate (Figure 6).

* A two-edge widget that models a wire connecting two gates (Figure 7).

* The vertex representing the circuit's output is connected to an otherwise isolated vertex. (This will force the output to be true in any legal covering.)

* The integer N that is selected is five times the number of NAND gates in the circuit, plus the number of free inputs in the circuit.

NAND-gate widget: Figure 6 shows the nine-vertex widget that models a two-input NAND gate.

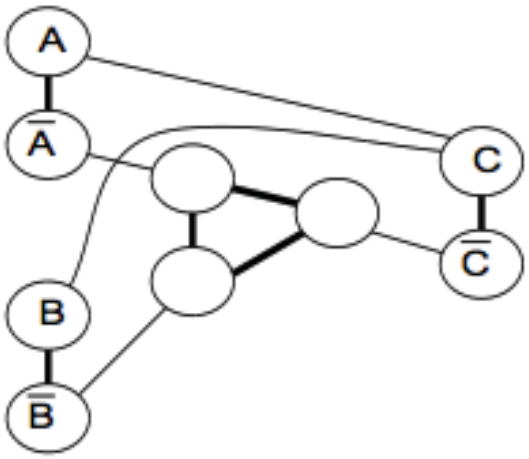

Figure 6: VC widget for C = A NAND B

The requirement is that a five-cover be found for this widget. It is easy to see that no covering of less than five vertices is possible, because five vertices are required to cover just the emboldened edges in the figure. In particular:

* At least one of the vertices A and $\overline{A}$must be marked because they are connected; similarly for vertices B $\overline{B}$ and, and for vertices C and $\overline{C}$.

* At least two of the vertices in the emboldened triangle must be marked. Otherwise, a triangle edge will be left uncovered.

* Thus, if the widget is five-covered, then exactly one of A and $\overline{A}$ will be marked (and similarly for B $\overline{B}$ and, and for C and $\overline{C}$); two of the triangle-vertices will be marked.

The mapping from CSAT is as follows: A being selected corresponds to the A input being true, andbeing selected corresponds to the A input being false; similarly for B and C. With this correspondence, it is not difficult to show that a five-covering for the widget induces the relationship C = A NAND B.

Wire widget: The wire widget is simply two "crossing edges", as shown in Figure 7. Each represents a vertex in a five-covered NAND-widget, so that exactly one of X andwill be marked, and similarly for Y and. (The vertex-count constraints of the target problem will enforce this.) If, for example, X and were both marked, then the edge between and Y would remain uncovered.

Figure 7: VC widget modeling a circuit wire

Figure 8 shows the VC-transformed circuit from Figure 2. Because there are three free inputs and four gates, the number of vertices in the covering is 23 (3 for the inputs, plus 5x4=20 for the coverings). The extra vertex extending from Z cannot be marked because all 23 markings are needed elsewhere. This means that any solution to this VC problem will have vertex Z marked; this corresponds to the circuit's output being true.

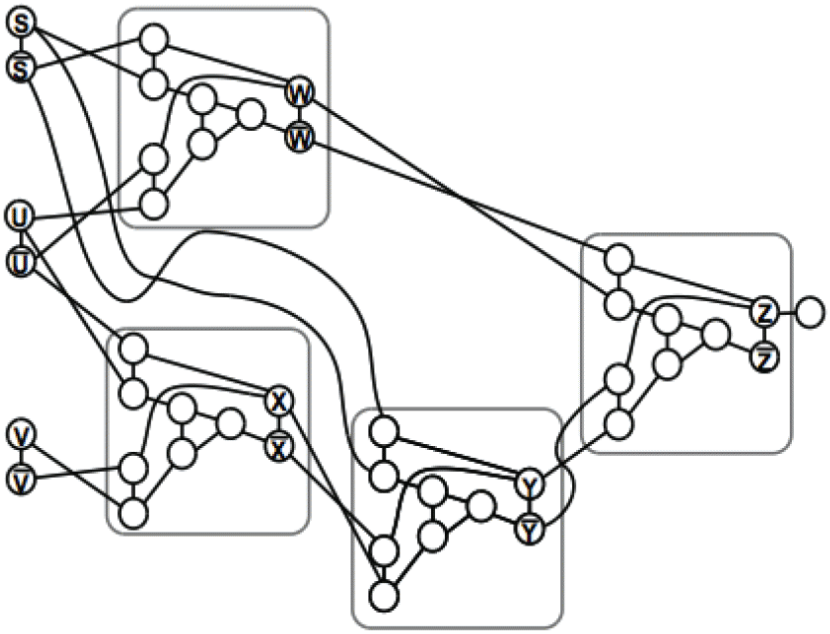

Figure 8: Example circuit, transformed to VC

As with the graph-coloring transformation, a solution to the circuit-problem leads directly to a solution for the corresponding VC problem, and vice versa.

4. SUMMARY

Students in the program were able to compare the Turing-machine and circuit-based approaches to NP-completeness because the topic is studied in two different courses. Student feedback indicated that the Turing machine approach was considered to be more rigorous, but that it was also more complicated. The transformations to and from CSAT were much simpler than for the Turing machine; students could independently apply CSAT transformations on homework. The G3C and VC widgets were sufficiently complex, however, that it took some effort to verify that they modeled NAND gates.

Students in a CS program often find the concept of NP-completeness difficult. Because many CS students will have taken a logic design course, the idea of creating a combinational circuit to check a solution candidate is something that makes sense. The availability of intuitive, visual transformations can help solidify their intuition about NP-completeness.

REFERENCES

[1] Baase, S., Van Gelder, A., *Computer Algorithms: Introduction to Design and Analysis (3rd Ed.),* Boston, MA: Addison-Wesley, 1999.

[2] Cook, S., The complexity of theorem proving procedures, *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, 151-158, 1971.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C., *Introduction to Algorithms (3rd Ed.)*, Cambridge, MA: The MIT Press, 2009.

[4] Dasgupta, S., Papadimitriou, C., Vazirani, U., *Algorithms*, New York, NY: McGraw-Hill, 2008.

[5] Garey, M. R., Johnson, D. S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, San Francisco, CA: Freeman, 1979.

[6] Hein, J. L., *Theory of Computation: An Introduction*, Sudbury, MA: Jones and Bartlett, 1996.

[7] Hopcroft, J. E., Motwani, R., Ullman J. D., *Introduction to Automata Theory, Languages, and Computation (3rd Ed.)*, Boston, MA: Addison-Wesley, 2006.

[8] Kernighan, B.W., Ritchie, D. M., *The C Programming Language (2nd Ed.)*, Upper Saddle River, NJ: Prentice Hall, 1988.

[9] Kleinberg, J., Tardos E., *Algorithm Design*, Boston, MA: Addison-Wesley, 2006.

[10] Mano, M. M., Kime, C., *Logic and Computer Design Fundamentals (4th Ed.)*, Upper Saddle River, NJ: Prentice Hall, 2007.

[11] McConnell, J. J., *Analysis of Algorithms (2nd Ed.)*, Sudbury, MA: Jones & Bartlett, 2008.

[12] Sipser, M., *Introduction to the Theory of Computation (3rd Ed.)*, Boston, MA: Cenage Learning, 2012.