An integer programming proof of Cook's theorem of NP-completeness

Benjamin Letham

Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA
bletham@mit.edu

September 2011

Abstract

The theory of NP-completeness has its roots in a foundational result by Cook, who showed that Boolean satisfiability (SAT) is NP-complete and thus unlikely to admit an efficient solution. We prove an analogous result using binary integer programming in the place of SAT. The proof gives deeper insight into the theory of NP-completeness to operations researchers for whom the language of integer programming is more natural than that of Boolean logic.

1 Introduction

In 1971, Cook [1] provided the first proofs that a collection of problems of practical interest are NP-complete, meaning that they are, in a particular sense, among the hardest problems in NP. Since that time, the theory of NP-completeness has guided researchers to either find efficient algorithms, or to know when to stop looking and to focus on approximation schemes. The theory of NP-completeness is critically important for operations researchers, as demonstrated by the extensive literature on proving various problems in operations research are NP-hard [see 2, 3, 4, 5, for example].

Almost all proofs of NP-completeness base themselves on reductions from problems that have already been proven to be NP-complete. At the root of all of these reductions lies Cook's theorem, the foundational result in the theory of NP-completeness which showed that the Boolean satisfiability problem (SAT), among other problems, is at least as hard as every other problem in NP. This first proof of NP-completeness appeals to Turing machines and the definition of the set NP.

The standard proof of Cook's theorem [6, for example] relies on using Boolean constraints to model the way in which a Turing machine solves an arbitrary problem in NP. Operations researchers are likely more familiar with the language of integer programming than the language of Boolean satisfiability. We provide and prove an analog to Cook's theorem based on the zero-one integer linear programming feasibility problem (ZOIP), rather than SAT. The proof gives operations researchers easier understanding of and deeper insight into this fundamentally important result.

2 Turing machines and the class NP

NP is a class of decision problems, that is, problems for which every instance is either a YES instance or a NO instance. For example, an instance of ZOIP is a collection of linear constraints over binary variables, which is either feasible (YES instance) or not (NO instance). A decision problem is in NP if for every YES instance there exists a corresponding certificate that can be verified in polynomial time. A certificate for a ZOIP YES instance is a feasible solution, which can be verified in polynomial time simply by checking that all of the constraints are satisfied. The idea of the proof is to show that for an arbitrary problem in NP, the process by which a Turing machine verifies the certificate to a YES instance can be modeled with zero-one integer linear constraints. We thus proceed with a brief description of Turing machines to introduce the necessary definitions and notation.

A deterministic one-tape Turing machine (DTM) is an abstract computing device consisting of a tape, a state, and a read/write head. The tape is an infinite sequence of squares, each square containing a symbol from an alphabet of v distinct symbols, labeled $1, \ldots, v$. Each square is given an integer label, extending towards both positive and negative infinity from square 0. At any step of computation the read/write head is located over a single square. Throughout the computation the read/write head moves along the tape, reading and writing symbols. The state of the machine at any step of computation is one of a set of r distinct states, labeled $1, \ldots, r$.

Computation begins with the machine in state 1, the read/write head located at square 1, and both the problem instance and its certificate written on the tape. The problem instance and certificate are encoded using symbols from the tape alphabet $(1, \ldots, v)$ as $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_m$ respectively. The problem instance is written to the tape on squares 1 through n and the certificate is written in squares -1 through $-m$. All other squares (square 0, squares above n, and squares below $-m$) are initialized with a special "blank" symbol, which we label as symbol 1.

A step of computation on a DTM consists of three main dynamic elements: the machine state is updated, the symbol in the current square is overwritten by a new symbol, and the read/write head is moved to a new square. These changes proceed according to a predetermined set of instructions, and depend on the current state of the machine and the symbol that is currently being read by the read/write head. Suppose the machine is in state k and the current square contains symbol l. The state update instructions are contained in an $r \times v$ matrix A^Q, of which each element takes values in $\{1, \ldots, r\}$. The element $A^Q_{k,l}$ contains the new state of the machine, given the current state k and the current symbol l. Similarly, the symbol update instructions are contained in an $r \times v$ matrix A^S, of which each element takes values in $\{1, \ldots, v\}$ with element $A^S_{k,l}$ being the new symbol that is written in the current square. Finally, the update to the head position follows an $r \times v$ matrix A^H of which each element is $+1$ or -1, indicating that the head moves one square to the right or to the left, respectively.

Computation is finished and the machine halts when one of two special states, state YES or state NO, is reached. We label the YES and NO halting states as states 2 and 3 respectively. For any problem Π in the class NP, there exists a DTM M and a polynomial $p(n)$ such that M halts in state YES in at most $p(n)$ steps of computation if and only if x is a YES instance to Π and y is a corresponding certificate.

We finally define NP-completeness. A problem Π_1 is *polynomially reducible* to Π_2 if there exists a polynomial transformation f that maps each instance x of Π_1 to an instance $f(x)$ of Π_2 with the property that $f(x)$ is a YES instance of Π_2 if and only if x is a YES instance of Π_1. A problem Π is called *NP-complete* if it is in the class NP, and every other problem in NP is polynomially reducible

to Π.

3 Modeling a DTM using ZOIP

The insight behind Cook's original result was that an arbitrary DTM could be modeled using a collection of boolean formulas. Here we use a similar strategy to show that an arbitrary DTM can be modeled using zero-one integer linear constraints. The proof parallels that given for SAT in Garey and Johnson [6]. We introduce three sets of decision variables: Q for the state of the machine, H for the location of the head, and S for the symbols on the tape. Specifically, we design the constraints so that

$$Q_{i,k} = \begin{cases} 1, & \text{if state is } k \text{ at step } i \\ 0, & \text{otherwise} \end{cases}$$

$$H_{i,j} = \begin{cases} 1, & \text{if head scans square } j \text{ at step } i \\ 0, & \text{otherwise} \end{cases}$$

$$S_{i,j,l} = \begin{cases} 1, & \text{if square } j \text{ has symbol } l \text{ at step } i \\ 0, & \text{otherwise.} \end{cases}$$

We now enumerate the properties of a DTM, and specify the sets of constraints that ensure these variables will faithfully model the DTM.

3.1 Decision variables and machine components

We are primarily interested in determining if the DTM enters state YES by computation step $p(n)$, as this is a necessary and sufficient condition for x to be a YES instance. Thus we need to model only steps $i = 0, \ldots, p(n)$ of computation. Similarly, because the read/write head moves only one square per step of computation, it is sufficient to consider squares $j = -p(n)+1, \ldots, p(n)+1$. We consider the full range of states $k = 1, \ldots, r$ and the full range of symbols $l = 1, \ldots, v$.

At each step of computation i, the DTM is in exactly one state, scans exactly one square, and each square on the tape contains exactly one symbol. These constraints are naturally expressed in the language of ZOIP modeling:

$$Q_{i,k} \in \{0,1\}, \qquad \forall i,k, \tag{1}$$

$$H_{i,j} \in \{0,1\}, \qquad \forall i,j, \tag{2}$$

$$S_{i,j,l} \in \{0,1\}, \qquad \forall i,j,l, \tag{3}$$

$$\sum_{k=1}^{r} Q_{i,k} = 1, \qquad \forall i, \tag{4}$$

$$\sum_{j=-p(n)+1}^{p(n)+1} H_{i,j} = 1, \qquad \forall i, \tag{5}$$

$$\sum_{l=1}^{v} S_{i,j,l} = 1, \qquad \forall i,j. \tag{6}$$

3.2 Initial conditions

Recall the starting conditions for the DTM: The state is 1, the read/write head scans square 1, square 0 is blank, squares 1 through n contain the input x, squares -1 through $-m$ contain the certificate y, and the remaining squares are blank. As ZOIP constraints,

$$Q_{0,1} = 1, \tag{7}$$
$$H_{0,1} = 1, \tag{8}$$
$$S_{0,0,1} = 1, \tag{9}$$
$$S_{0,j,x_j} = 1, \qquad j = 1, \ldots, n, \tag{10}$$
$$S_{0,j,1} = 1, \qquad j > n. \tag{11}$$

Given an instance x, we do not necessarily know its certificate: We know only that if x is a YES instance (the case of interest) then a certificate exists. We leave $S_{0,j,l}$ unconstrained for $j < 0$ so that if a certificate exists, there will be a satisfying variable assignment.

3.3 DTM dynamics

The decision variables must evolve with each time step as a DTM. Specifically, at each step, the state at the next step must update according to A^Q, the head position must update according to A^H, and the symbol in the current square must update according to A^S; each of these, of course, depending on the current state of the machine and the symbol in the current square. This is accomplished by the following set of constraints:

$$Q_{i+1,A^Q_{k,l}} \geq -2 + H_{i,j} + S_{i,j,l} + Q_{i,k}, \qquad \forall j,k,l,i < p(n), \tag{12}$$
$$H_{i+1,j+A^H_{k,l}} \geq -2 + H_{i,j} + S_{i,j,l} + Q_{i,k}, \qquad \forall j,k,l,i < p(n), \tag{13}$$
$$S_{i+1,j,A^S_{k,l}} \geq -2 + H_{i,j} + S_{i,j,l} + Q_{i,k}, \qquad \forall j,k,l,i < p(n). \tag{14}$$

For those constraints in particular the language of ZOIP modeling is much more convenient than that of SAT. We must additionally ensure that all symbols other than the one being modified by the read/write head remain unchanged:

$$S_{i+1,j,l} \geq S_{i,j,l} - H_{i,j}, \qquad \forall j,k,l,i < p(n), \tag{15}$$
$$S_{i+1,j,l} \leq S_{i,j,l} + H_{i,j}, \qquad \forall j,k,l,i < p(n). \tag{16}$$

When the DTM halts at some $i < p(n)$, we wish for the decision variables to maintain the same state, head position, and tape contents until $i = p(n)$. This is accomplished by ensuring that in the DTM update instructions $A^Q_{2,l} = 2$, $A^Q_{3,l} = 3$, $A^H_{2,l} = A^H_{3,l} = 0$, and $A^S_{2,l} = A^S_{3,l} = l$.

The constraints (1)-(16) ensure that the decision variables Q, H, and S are a faithful representation of a DTM applied to problem instance x.

4 Proof of NP-completeness

All that remains of the proof are one more constraint and a few simple arguments.

Lemma 1. *Consider a DTM M with problem instance x on the tape. The ZOIP instance given by constraints (1)-(16), together with the constraint*

$$Q_{p(n),2} = 1 \tag{17}$$

is feasible if and only if there exists a certificate y such that when written on the tape, M enters state YES *by time step $p(n)$.*

Proof. We saw in Section 3 that the constraints in (1)-(16) ensure that the decision variables Q, H, and S describe exactly the behavior of M. Now consider the addition of (17): Because the decision variables Q are exactly the state of M, this constraint will be satisfied if and only if M enters state YES (state 2) by time step $p(n)$. A feasible assignment to the variables $S_{0,j,l}$ for $j < 0$ corresponds to a valid certificate. □

The main theorem, the analog of Cook's theorem for ZOIP, follows directly.

Theorem 1. *ZOIP is NP-complete.*

Proof. We already discussed how ZOIP is in NP. It remains to be shown that an arbitrary problem Π in NP is polynomially reducible to ZOIP. By the definition of NP, x is a YES instance of Π if and only if for some certificate y, the DTM M corresponding to Π enters state YES in at most $p(n)$ steps. Now we use (1)-(17) to construct the ZOIP instance that models M. By Lemma 1, this ZOIP instance is feasible (a YES instance) if and only if x is a YES instance to Π. It is easy to see that there are a total of $O(p(n)^2)$ variables and $O(p(n)^2)$ constraints in the ZOIP instance, thus it is a polynomial reduction. □

5 Conclusions

In Cook's foundational paper he showed that several problems of practical interest, such as SAT, are NP-complete. Here we proved a similar result using ZOIP in the place of SAT that is notationally cleaner and more straightforward to operations researchers, for whom the language of ZOIP is more natural than that of SAT. This proof gives deeper insight to operations researchers into Cook's important result, and also provides a natural way to teach NP-completeness in an integer programming course. In fact, many textbooks on integer programming discuss NP-completeness and provide proofs of NP-completeness using reductions from other NP-complete problems, but do not provide a proof of Cook's theorem [7, 8, 9, for example]. The proof that we give here is a natural, rigorous, and insightful way to discuss NP-completeness in the context of integer programming.

References

[1] S. A. Cook, The complexity of theorem-proving procedures, in: Proceedings of the third annual ACM symposium on Theory of computing, STOC '71, ACM, New York, NY, USA, 1971, pp. 151–158.

[2] J. Du, J. Y. Leung, Minimizing total tardiness on one machine is NP-hard, Math. Oper. Res. 15 (1990) 483–495.

[3] O. Ben-Ayed, C. E. Blair, Computational difficulties of bilevel linear programming, Oper. Res. 38 (1990) 556–560.

[4] M.-S. Chern, On the computational complexity of reliability redundancy allocation in a series system, Oper. Res. Lett. 11 (1992) 309 – 315.

[5] M. O. Ball, B. L. Golden, R. V. Vohra, Finding the most vital arcs in a network, Oper. Res. Lett. 8 (1989) 73 – 76.

[6] M. R. Garey, D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman & Co., New York, NY, USA, 1979.

[7] D. Bertsimas, R. Weismantel, Optimization Over Integers, Dynamic Ideas, Belmont, MA, USA, 2005.

[8] G. L. Nemhauser, L. A. Wolsey, Integer and Combinatorial Optimization, Wiley-Interscience, New York, NY, USA, 1999.

[9] A. Schrijver, Theory of Linear and Integer Programming, Wiley Series in Discrete Mathematics & Optimization, John Wiley & Sons, 1998.