A Reformulation of the Cook-Levin Theorem

Ben Morris

May 1, 2023

Introduction

In the mid 20th century, mathematicians began to study the emerging field of computer science. Many point to Alan Turing's 1936 definition of a device now known as a Turing Machine as the paper that started this field, as it defined what it means to do a computation. Since then, computer scientists have shown that a Turing Machine is able to simulate any other computation we have conceived of so far. This includes so-called Nondeterministic Turing Machines (NTMs), in which multiple branches of a path can be followed at the same time. However, it was observed that a deterministic Turing Machine (DTM) took a lot longer to do the same calculation as a NTM - an exponential time complexity compared to an NTM's polynomial. Whether or not it is possible for a DTM to do the same calculation as an NTM in polynomial time - the so-called P vs NP problem - is one of the largest unsolved problems in computer science today.

A group of problems have been discovered to be polynomial (P); that is, solvable in polynomial time. This is the most basic complexity class, as most polynomial problems can be solved given enough computing power and time. A (possibly) wider breadth of problems is called NP. These are problems that, given a solution, can be *checked* in polynomial time. Trivially, any problem in P is also in NP, as one can check a solution by just solving the problem. The name NP means "Nondeterministic Polynomial" - able to be run in polynomial time on a NTM.

It has since been shown that many problems in NP, but not thought to be in P, are actually the same problem; that is, solving one will give an algorithm to solve the rest. This group of problems is called NP-Complete. The question arises, if NP-Complete problems are defined by their relation to each other, what was the first NP-Complete problem, and how was it shown to be NP-Complete? This is where Cook comes in. In a paper in 1973, Cook proved that if a certain problem (known as SAT) can be solved in polynomial time, then any NTM can be run in polynomial time. SAT then became the seed to show that any other problem is NP-Complete.

1 Background Definitions

1.1 Turing Machines

Turing Machines (TMs) are one of the first formal definitions of what a computation is. The basic picture is a tape with a start that extends infinitely far to the right. Starting on the leftmost square is a read/write head, which has the ability to look at the tape and change its value depending on the current "state" the head is in. The head can then move left or right and change its state. A string of symbols is put on the tape, and the head moves through them, modifying them. Eventually, the head will set itself to one of two states: "accept" or "reject", depending on if the TM accepted or rejected the string. As soon as this happens, the TM halts.

The formal definition of a TM M is a tuple of seven values: $(Q, \Sigma, \Gamma, \delta, s, a_{cc}, r_{ej})$ where:

- Q is a finite set representing each of possible states M can be in
- Σ is a finite set of symbols that can be put on the tape as input. The first symbol will be put at the position 0 and the rest will be put, in order, to the right. Any symbols past the end of the input string will be set to a blank symbol (here represented as $\llcorner\!\lrcorner$)
- Γ is a finite set of symbols that the TM can write onto the tape. This is often larger than Σ, but note that it *must* be a superset of Σ. Most often, $\Gamma = \Sigma \cup \{\llcorner\!\lrcorner\}$

- δ is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$. This defines how the TM moves and changes state. Specifically, if the TM is in state q reading symbol a, and $\delta(q, a) = (p, b, d)$, then the TM will transition from state q to state p, switch the symbol at the current position from a to b, and finally move in the direction given by d. If this would cause the TM to move off the left end of the tape, the machine instead stays still.
- s is an element of Q that defines the state that the TM starts in.
- a_{cc} is an element of Q, different from s, that defines the "accepting" state
- r_{ej} is an element of Q, different from both s and a_{cc}, that defines the "rejecting" state. Note that the previous three definitions mean that Q must consist of at least three elements - that is, $Q \supseteq \{s, a_{cc}, r_{ej}\}$.

A TM M is said to "accept" a string w if and only if, when w is given as input to M, M eventually reaches the accepting state. If instead M reaches the rejecting state when given w, M is said to "reject" w. If, for every input $w \in \Sigma^*$,[1] M either accepts or rejects w, then M is said to be a "halting" TM. The set of all strings that a TM M accepts is said to be the language of M. Formally, $L(M) = \{w \in \Sigma^* \mid M \text{ accepts } w\}$.

1.2 Nondeterministic Turing Machines

The previous section is the classical definition of a Turing Machine, which is often called a "Deterministic Turing Machine" (DTM). There is a separate version of a TM called a "Nondeterministic Turing Machine" (NTM). Whereas a DTM can only transition to one other state, an NTM can transition to *any number of possible states.* If any of the possible paths reaches the accept state, the input string is accepted. If all possible paths reach the reject state, the input is rejected. There are two ways of thinking about this:

1. Every time the NTM is told to transition to multiple states, it creates multiple copies of itself, each transitioning to a different state. This effectively creates a tree of branching paths the NTM takes
2. The NTM is "extremely lucky" and always takes the path which results in an accepting state in the least amount of steps.

To get to the definition of an NTM from a DTM, one only needs to change δ to:

- δ: a function from $Q \times \Gamma$ to $\mathcal{P}(Q \times \Gamma \times \{L, R\})$,[2] where, if $\delta(q, a) = S$, then, when the NTM reads symbol a while in state q, $\forall (p, b, d) \in S$, the NTM will create a clone of itself where it switches its state from q to p, switches the symbol from a to b, and moves in the direction given by d. Equivalently, the NTM chooses the $(p, b, d) \in S$ that would result in reaching the accept state the fastest.

It has been shown that NTMs are not more powerful than DTMs - that is, anything an NTM can compute, a DTM can also compute. However, with current methods, a DTM takes significantly more time. This time difference is exponential - for a calculation that an NTM can do in polynomial time, a DTM takes exponential time. This is because a DTM essentially does BFS on the tree created by the NTM - a process which takes an exponential amount of time as the number of steps increases. Whether this time increase is necessary is core to the question of P = NP.

1.3 The SAT Problem

The Boolean Satisfiability Problem, or SAT for short, is the problem of determining if a given boolean formula is able to be satisfied. A boolean formula is defined as a set of literals combined with the operations NOT ($\neg$), AND ($\wedge$), and OR ($\vee$). Most commonly, these expressions are written in Conjuctive

[1] Σ^* is the set of all strings of finite length consisting of only symbols from Σ.

[2] $\mathcal{P}(\mathcal{S})$ denotes the powerset of S - the set of all subsets of S, including $\emptyset$. However, with the exception of a_{cc} and r_{ej}, the output of δ has to contain at least one element.

Normal Form (CNF), which means the literals are combined by ORs, and the groups are combined by ANDs. For example, $(x_1 \lor x_2 \lor \neg x_3) \land (x_3 \lor x_4)$ is in CNF.

This problem is definitely in NP - given a solution, represented as a boolean vector (with each element of the vector mapping to a specific literal), one can evaluate this formula in polynomial time and see if it results in true.

1.4 P-Reduction and Oracle Machines

Polynomial Reduction, abbreviated P-Reduction or just reduction, is the process of showing that one problem can be solved by using another without changing its complexity class. The intuition is a black box that can solve a problem B. To prove that A reduces to B, you must show that you can modify the input to A to be the input to B in polynomial time, and then show that you can modify the corresponding output of B to be the output of A, also in polynomial time. Since the only additions to the algorithm of B have been polynomial time, the complexity class of the problem has not changed when going from B to A. Therefore, B must be at least as complex as A (if it was less complex, you could solve A faster than its own complexity class by doing the process described above).

Cook formalized this definition in the same paper in which he showed that NTMs reduce to SAT through the use of an Oracle Machine. An Oracle Machine (OM) is an extension of a DTM which has three extra states: query, yes, and no. The OM will modify the input string, then set itself to the query state. It will then be switched (by an "oracle") to either yes or no depending on the value of the tape. This operation is assumed to be a single step. The OM will continue from there.

Formally, an OM M is a tuple of 11 elements: $(Q, \Sigma, \Gamma, \delta, s, a_{cc}, r_{ej}, O, q_{ry}, y_{es}, n_o)$. The first seven elements are the same as a regular DTM. The remaining are defined as follows:

- O: A function from Γ^* to $\{True,\ False\}$. It takes in the current value of the tape, and returns a boolean value. This value corresponds to whether a different decision problem would accept that string.
- q_{ry}: An element of Q that the OM transitions into to activate the oracle.
- y_{es}: An element of Q that the OM transitions into after q_{ry} if the result of $O(\text{tape})$ is $True$
- n_o: An element of Q that the OM transitions into after q_{ry} if the results of $O(\text{tape})$ is $False$.

Note that q_{ry}, y_{es}, and n_o are three different states, each of which are distinct from s, a_{cc}, and r_{ej}.

If the OM is able to halt on any input within a number of steps proportional to the length of the input raised to some power, then the problem the OM solves is said to P-reduce to the problem that O solves.

2 Proof that SAT is NP-Complete

Now, I will reconstruct Cook's proof that NTMs can reduce to SAT - that is, that there is an OM that, given an oracle to solve SAT, can simulate NTMs in polynomial time. This will show that the complexity class of SAT is at least as complex as that of NTMs; should someone find an algorithm to solve SAT in polynomial time, they can also simulate NTMs in polynomial time.

First, we will define an NTM M. Then, we will create a SAT problem from the elements of M and an input of $w \in \Sigma^*$ that is solvable if and only if M accepts w. Finally, we will show that the creation of this SAT problem takes polynomial time.

Given an NTM $M = (Q, \Sigma, \Gamma, \delta, s, a_{cc}, r_{ej})$, we will show that there is an Oracle Machine which, given an oracle to solve SAT problems, can simulate M in polynomial time.

Let's say that M can always halts within $P(|w|)$ steps $\forall w \in \Sigma^*$, for some polynomial P. For the rest of the proof, we will hold w constant, and define $T = P(|w|)$. As the NTM can only move one square per step, observe that no squares to the right of square T will ever be put into consideration.

2.1 The Literals

We will define three sets of literals:

- $Z^i_{j,t}$ $\forall i \in \Gamma$ $\forall j,t \in \{1...T\}$. $Z^i_{j,t}$ is true if and only if square j contains symbol i at step t. Note that we can restrict i to be less than T because no squares right of T matter.
- Q^q_t $\forall q \in Q$ $\forall t \in \{1...T\}$. Q^q_t is true if and only if the NTM is in state q at timestep t.
- $S_{j,t}$ $\forall j,t \in \{1...T\}$. $S_{s,t}$ is true if and only if the NTM is reading square j at timestep t.

2.2 The Formula

We will define a SAT formula A, defined as $B \wedge C \wedge D \wedge E \wedge F \wedge G$. B through D will give the constraints that, for each timestep, the NTM can only have one square scanned, one symbol per square, and one state. E will specify the initial conditions. F will define how the literals are allowed to change between each time step. Finally, G will assert that the machine eventually reaches a halting state. I will specify what these are in the following sections.

2.2.1 B through D: One-at-a-Time Constraints

B will maintain that exactly one square will be scanned for each timestep. We will define B as $\wedge^T_{t=1} B_t$, where $B_t = (\vee^T_{j=1} S_{j,t}) \wedge (\wedge_{1 \leq i < j \leq T} \neg S_{i,t} \vee \neg S_{j,t})$. That is, at least one square must be scanned, but no two squares can both be scanned.

C and D can be defined in similar ways:

- $C = \wedge^T_{j=1} \wedge^T_{t=1} C_{j,t}$, where $C_{j,t} = (\vee_{i \in \Sigma} Z^i_{j,t}) \wedge (\wedge_{i,k \in \Sigma, i \neq k} \neg Z^i_{j,t} \vee \neg Z^k_{j,t})$
- $D = \wedge^T_{t=1} D_t$, where $D_t = (\vee_{q \in Q} Q^q_t) \wedge (\wedge_{q,p \in Q, q \neq p} \neg Q^q_t \vee \neg Q^p_t)$

2.2.2 E: Initial Conditions

E will give the initial conditions of the NTM. Specifically:

$$E = Q^s_1 \wedge S_{1,1} \wedge (\wedge^{|w|}_{j=1} Z^{w_j}_{j,1}) \wedge (\wedge^T_{j=|w|+1} Z^{\text{␣}}_{j,1})$$

That is, in the first timestep:

1. M is in state s
2. M is scanning square 1
3. w is spread across the first $|w|$ squares
4. all other possible squares M can visit read ␣

2.2.3 F: Transitions

F will maintain that updates to the various variables happen according to the rules of an NTM[3]. F will be of the form $\wedge_{q \in Q} \wedge_{i \in \Sigma} \wedge_{1 \leq j,t \leq T} F^{q,i}_{j,t}$, such that $F^{q,i}_{j,t}$ specifies the constraints for if M is in state q, reading symbol i at square j during timestep t. This can be represented as:

$$F^{q,i}_{j,t} = \neg Z^i_{j,t} \vee \neg S_{j,t} \vee \neg Q^q_t \vee (\vee_{(p,b,d) \in \delta(q,i)} Z^b_{\max(j+d,0),t+1} \wedge S_{\max(j+d,0),t+1} \wedge Q^p_t)$$

This constraint means that either the symbol is not read, the square is not scanned, M is not currently in the state, or M transitions into one of the states given by the result of its δ function. Due to the nature of SAT, solving the SAT problem is analogous to M just getting "extremely lucky" when guessing the proper paths to take, as, if there is a path that fits these constraints and results in an accepting state, then the SAT oracle will return true.

[3]In Cook's original paper, he split this up into F, G, and H - one for each set of variables. However, that seems to only work for a DTM - not an NTM (though it is unclear if this really is the case, as Cook does not provide exact formulas any of them). This reformulation breaks the pattern that A is in CNF, but is clear that it works with an NTM.

2.2.4 G: Accepting State

Finally, we need to ensure that this NTM reaches an accepting state (or else SAT would return true whenever there is a valid path through the states, which is pretty much always). This is a really simple formula:

$$G = \vee_{t=1}^{T} Q_t^{acc}$$

That is, at least one state reached must be the accepting state.

2.3 Polynomial Time Proof

Cook himself kind of glazes over this part of the proof, so I will provide an approximation of how the proof would look, by showing that the number of literals included in A is bounded by a polynomial of $|w|$.

First, we will count the number of literals that appear (i.e. without repeats). There are three distinct categories, which we will add together:

- $Z_{j,t}^{i}$. There is one element for every symbol, square, and timestep, totalling to $|\Sigma|T^2$. As Σ is constant and finite, and T is a polynomial of $|w|$, this is polynomial.
- Q_t^q. There is one element for every state and timestep, totalling to $|Q|T$. As Q is constant and finite, and T is a polynomial of $|w|$, this is polynomial
- $S_{j,t}$. There is one element for every square and timestep, totalling to T^2. As T is a polynomial of $|w|$, this is polynomial.

The formula also has polynomial length. To count this length, we will count the number of literals with repeats, in order by section.

- B is made up of T terms, B_1 through B_T. For each $t \in \{1...T\}$, B_t has $T + \binom{T}{2}$ terms, totalling to $T^2 + T\binom{T}{2}$ total terms, which is polynomial on .
- C is made up of T^2 terms. For each $j, t \in \{1...T\}$, $C_{j,t}$ has $|\Sigma| + \binom{|\Sigma|}{2}$ terms, totalling to $T^2|\Sigma| + T^2\binom{|\Sigma|}{2}$ terms, which is polynomial as $|\Sigma|$ is constant and T is polynomial with $|w|$
- D is made up of T terms, D_1 through D_T. For each $t \in \{1...T\}$, D_t has $|Q| + \binom{|Q|}{2}$ terms, which is polynomial as $|Q|$ is constant and T is polynomial with $|w|$.
- E is made up of $T + 2$ terms - one for each square, one for the starting state, and one for the starting scan.
- F is made up of $|Q||\Sigma|T^2$ terms. For each $q \in Q,\ i \in \Sigma,\ j, t \in \{1...T\}$, $F_{j,t}^{q,i}$ has a maximum $3 + 3 * 2^{2|Q||\Gamma|}$[4]. This exponential is a problem, however, there is a subtle solution. The actual input string to the OM we are creating has more than just w (the input to the NTM). It also contains the NTM *iteself.* Let's call this full input - an encoding of the NTM followed by the input string - W. Since the length of W would account for the number of transition states each input to δ has, creating F is actually bounded by a polynomial dependent on the length of W.
- G is made up of T terms - one for each timestep. as T is polynomial with $|w|$, G is properly bounded.

Therefore, one can construct an OM M that takes as input an encoding of an NTM followed by an input to that NTM, modify that full input in polynomial time to a SAT formula whose satisfiability is equivalent to whether the NTM accepts the input given to it. As per the definition of reducibility, this means that the problem of NTMs can be reduced in polynomial time to SAT. That is, given a DTM that can solve SAT formulas in polynomial time, one can construct a DTM that can simulate any NTM in polynomial time, thus showing that P = NP. This property of SAT (plus the fact that it can be verified in polynomial time) shows that SAT is NP-Complete.

[4]This exponential may be a consequence of my reformulation, but, even with this information split up (as in Cook's paper), the number of possible states an NTM can switch to, each of which must be represented by the SAT formula, is bounded solely by an exponential due to the nature of the powerset

Slideshow

I have also made a slide deck to present this proof in a slightly-less-formal way. You can access the slide deck here.

Bibliography

1. Cook, Stephen: *The Complexity of Theorem-Proving Procedures.*
2. Up and Atom: *NP-COMPLETENESS - The Secret Link Between Thousands of Unsolved Math Problems.*
3. University of Waterloo: *Lecturing Effectively.*
4. Times Higher Education: *Top tips on how to make your lectures interesting.*