A polynomial time solution for 2-SAT

William M. Leiserson

MIT
Cambridge, MA 02140
willtor@mit.edu

Abstract

The question of whether the P and NP complexity classes are equivalent is unresolved. It is conjectured that they are inequivalent because of the vast number of NP-Complete languages (including the ever-popular 3-SAT) for which there is no known polynomial time solution. The author of this paper respectfully disagrees and presents a polynomial time solution to a problem closely related to 3-SAT: 2-SAT.

1. Introduction

The complexity class, P $= TIME(O(n^k))$, for a problem of size n and some finite k. NP $= NTIME(O(n^k))$, which allows for non-determinism in the Turing Machine that solves it. Alternatively, NP can be thought of as the set of languages for which inclusion can be verified with a short certificate. It is trivial to show that P $\subseteq$ NP (ignore the non-deterministic capabilities of the NTM solver), but it is not known whether NP $\subseteq$ P. The problem was first formalized by Cook in [1]. That NP $\not\subseteq$ P is widely conjectured.[2][3]

The most natural method of demonstrating that the two classes are inequivalent is to identify a problem, p, such that $p \in$ NP and $p \notin$ P. The hardest problems in NP are natural candidates. Problems like 3-SAT are called NP-Complete because every problem, $q \in$ NP, can be reduced to 3-SAT in polynomial time: $q \leq_P$3-SAT. On the other hand, showing the reverse, 3-SAT$\leq_P q$, where a deterministic polynomial-time solution to q is known, would demonstrate that NP $\subseteq$ P, and therefore P = NP.

Although no deterministic polynomial-time solution to 3-SAT is known, such a solution is known for the related problem, 1-SAT.[3] In this paper, we introduce an algorithm for 2-SAT, a more closely related problem, that proves 2-SAT is clearly within P. It seems that it is only a matter of time before a deterministic poly-time solution to 3-SAT is discovered.

A more detailed look at complexity and especially the P versus NP problem are provided in section 2. The 2-SAT solution and proof follow in section 3. And, lastly, the implications and possible directions for future research are outlined in section 4.

DOI: http://dx.doi.org/10.1145/(to come)

Additionally, liberal use of graphs and charts has been made for the sake of being accepted to SIGTBD, a systems conference, since systems researchers seem to like that kind of thing.

2. Background

2.1 Complexity Classes and Hardness

Let P $= TIME(O(n^k))$ where n is the size of the input and k is a constant. NP $= NTIME(O(n^k))$, permitting non-determinism. It is an open problem whether P = NP. If it can be shown that the hardest problems in NP, the NP-C problems, can be solved in polynomial time without non-determinism, then P = NP. The most well-known of the NP-C problems is 3-SAT:

$$\{\langle\phi\rangle|\phi \text{ is a satisfiable 3-CNF formula.}\}$$

A 3-CNF formula is a boolean formula represented in conjunctive normal form wherein each disjunction has 3 literals. E.g.,

$$\phi = (x_0 \vee \overline{x_1} \vee x_2) \wedge (\overline{x_0} \vee v_3 \vee x_4) \wedge ...$$

3-SAT is NP-C because any problem in NP can be reduced to it in deterministic polynomial time. Therefore, if there is a deterministic polynomial-time solver for 3-SAT, any problem in NP can be reduced to 3-SAT and solved using that solver. The total number of steps is polynomial and P = NP. It is conjectured that P $\neq$ NP, though some work has been done to relate them.

The following problems are relevant to this work:

$$\text{1-SAT} = \{\langle\phi\rangle|\phi \text{ is a satisfiable 1-CNF formula.}\}$$

$$\text{2-SAT} = \{\langle\phi\rangle|\phi \text{ is a satisfiable 2-CNF formula.}\}$$

1-SAT $\in$ P by an algorithm that is presented below. 2-SAT has probably been an open problem until now. At any rate, its poly-time solution is almost certainly a very large step towards showing that 3-SAT $\in$ P.

2.2 Related Work

A polynomial time algorithm for the related 1-SAT problem is known:[3]

1. For each clause, take the single literal in that clause and make it true. If the literal is non-negated, set the variable to *true*, and if it is negated, set it to *false*. If the variable has already been set differently, reject the formula.
2. If every clause has been satisfied, accept.

The algorithm runs deterministically in polynomial time. There is no backtracking or searching. This allows for hope that 3-SAT may be solvable in polynomial time, though 1-SAT is a long way from 3-SAT.

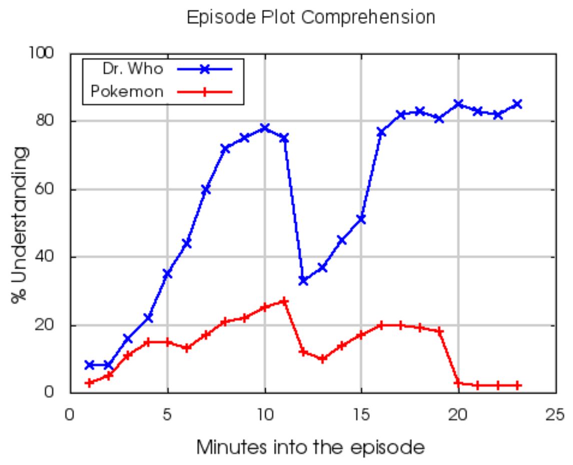

Figure 1. The author's comprehension of two hitherto-unfamiliar TV shows based on the viewing of a single episode of each. The aberration at minute 12 of Dr. Who is entirely attributed to Daleks and uncertainty as to why they are always yelling at each other.

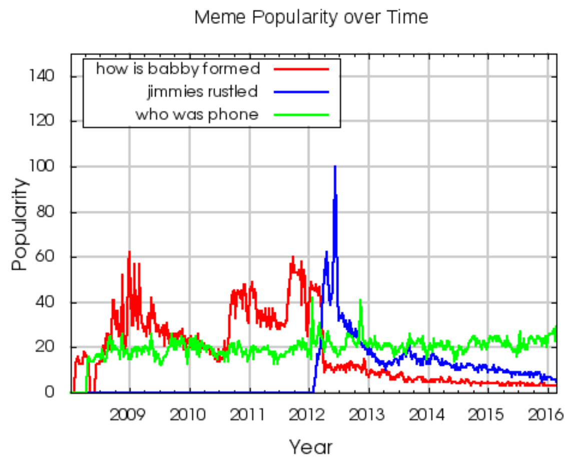

Figure 2. A visualization of the contrast between flash-in-the-pan memes such as "That really rustled my jimmies" and more perennial memes like "Who was phone?". "How is babby formed" is included as something in-between.[4]

3. Contribution

One-indexing has been used instead of zero-indexing because of the inherent mathiness of this paper. The author extends his apologies to the more systems-oriented readers.

Theorem 1. *Membership of 2-SAT can be determined in polynomial time.*

Lemma 1. *If $\phi \in$ 2-SAT, a satisfying assignment can be found in polynomial time.*

Proof. Assume $\phi \in$ 2-SAT. $\phi = c_1 \wedge c_2 \wedge c_3 \wedge \ldots$, where each clause, c_i is a disjunction of two literals. Select c_1 and set the first literal to *true* by providing the appropriate assignment to its variable. This places no constraints on the other literal in c_1. But anywhere else that variable appears in the formula, it either satisfies the clause in which it appears or constrains the other variable in that clause.

Therefore, recursively marking clauses as satisfied and assigning constrained variables requires no backtracking unless a contradiction is discovered (reassignment of a constrained variable). If a contradiction is discovered, the original guess must be wrong because every step since then has been constrained. Therefore, the other guess must be correct, since $\phi \in$ 2-SAT.

In some cases, clauses may be satisfied without introducing new constraints on the other literals. If the chain of constraints ends and there are unsatisfied clauses remaining, then the process can be repeated using ϕ', a formula identical to ϕ but missing the satisfied clauses. The clauses in ϕ' have no overlap in variables in $\phi - \phi'$ because any clauses with assigned variables must have been satisfied and removed. Therefore, a new guess can be made for the first literal in $c'_1 \in \phi'$.

Each guess takes polynomial time (linear, because the assignments are constrained), and there can be no more than a polynomial number of guesses since each guess resolves one or more variables in ϕ. □

Lemma 2. *If $\phi \notin$ 2-SAT, this can be proved in polynomial time.*

Proof. Using the procedure above, if any guess and its negation both lead to a contradiciton, then ϕ is unsatisfiable. Again, this is because all assignments that lead to the contradiction are constrained. □

Theorem 1 follows from lemmas 1 and 2. Membership in 2-SAT can be decided in polynomial time.

4. Conclusion and Future Work

Although no polynomial time algorithm for 3-SAT is known, the algorithm for 2-SAT is a significant step towards the goal of showing P = NP. We anticipate that it is only a matter of time before 3-SAT is cracked with an algorithm.

An alternate approach is a non-constructive proof of the existence of a polynomial time algorithm for 3-SAT using induction based on the 1-SAT base case and leveraging this algorithm for the inductive step. This second method is intuitively promising because non-constructive proofs, like Hansel, are so hot right now.[5]

4.1 Acknowledgements

I'd like to give a shout-out to Kanye East who, although he always insists he's doing his Kanye Least, is a veritable fount of knowledge and provides a Kanye Feast of ideas. Without his encouragement, this paper would not have been written. I also appreciate his allowing me to finish even though Sara Achour wrote one of the best papers of all time... one of the best papers of all time!

References

[1] S. Cook. The complexity of theorem-proving procedures *Proceedings of the third annual ACM symposium on Theory of Computing*, 1971

[2] S. Aaronson. Shtetl-Optimized: The Blog of Scott Aaronson *http://www.scottaaronson.com/blog/?p=1720* March, 2014

[3] C. Sense. Srsly? Is this not obvious?

[4] Google Trends *http://www.google.com/trends/* Retrieved Feb. 26, 2016

[5] B. Stiller. Zoolander *Paramount Pictures*, September 2001