$\mathsf{P} \neq \mathsf{NP}$

Sheldon S. Nicholl

October 20, 2023

Abstract

Not for publication! Still in draft form; probably contains errors.

In this paper, we use finite state automata rather than Boolean circuits to study the P versus NP problem. The overall approach is pretty simple. First, we show that the minimal automaton for a restricted Clique problem has an exponential number of states at a particular depth as a function of the length of its input. Next, if we add the assumption that there exists an NP-Complete sparse language, we use the polynomial-time reduction to build an automaton for the same Clique problem that has a number of states at that same depth that grows only polynomially with its input. Since this is a contradiction, there can be no such sparse language and P does not equal NP.

1 Definitions

If a string $s = vw$, then v is a *prefix* of s and w is a *suffix* of s. We access the ith character c of string s with the following notation: $c = s[i]$.

For a language L, define $L_n \subseteq L$ to consist of strings in L of length n:

$$L_n = \{s \mid s \in L \text{ and } |s| = n\} \tag{1}$$

The function v takes a graph g and returns its vertices, so if $g = (V, E)$ then $v(g) = V$. We write $g_1 \cong g_2$ if graphs g_1 and g_2 are isomorphic.

Following Sipser [1], a deterministic finite automaton (DFA) is a tuple

$$A = (\Sigma, S, F, \delta, s_0) \tag{2}$$

where Σ is a finite alphabet, S is a finite set of states, $F \subseteq S$ is the set of final states, $\delta : S \times \Sigma \to S$ is a mapping called the transition function, and $s_0 \in S$ is the initial state.

A state q is considered *accessible* if there is a path from the start state to q. A state q is called *co-accessible* if there is a path from q to a final state. Finally, an automaton is called *trim* if all its states are both accessible and co-accessible. To accommodate trim automata, we allow the state transition function δ to be a partial function. All automata in this paper are presumed to be minimal and acyclic.

If $a \in \Sigma$ and the state $s' = \delta(s, a)$ exists, then we can view the pair (s, s') as a transition in A. So the automaton defines a directed graph with vertices S and edges based on δ. We then get the usual graph concepts like a path, shortest path, and depth in their usual meanings. The *depth* of a state $s \in S$ is the length of the shortest path from the start state s_0 to s. A *level* is the set of all states in S at a given depth. The notation $|M|$ refers to the *size* of the automaton: the number of states in the automaton.

1.1 Creating an Automaton from a Finite Language

We will be creating a finite state automaton from a finite language via the following standard procedure. Since the language is finite, it can be written $\{w_1, w_2, w_3, ..., w_m\}$ where $w_i \in \Sigma^*$. This set can then be turned into the following regular expression: $E = w_1 \cup w_2 \cup w_3 \cup ... \cup w_m$. E is then converted to a non-deterministic finite automaton N. Then N is converted into a deterministic finite automaton D. Finally, D is minimized to yield the minimal automaton we are looking for [1].

1.2 Creating a Propositional Expression from a Finite Language

In this section, we create a propositional expression for a finite language $L \subseteq \{0,1\}^n$, where $|L| = m$. Each word $w_i \in L$ is of length n by definition. So we first create a new propositional variable for each character position in w_i: $p_1, p_2, p_3, \ldots, p_n$. We then build up the literals q_j depending on whther the word w_i has a 1 or a 0 there as follows: if $w_i[j] = 1$ then $q_j = p_j$ else $q_j = \neg p_j$. Then word w_i can just be recognized with the conjunction $v_i = q_1 \wedge q_2 \wedge q_3 \wedge \ldots \wedge q_n$. Finally the finite language L can be recognized by a finite disjunction: $v_1 \vee v_2 \vee v_3 \vee \ldots \vee v_m$.

2 Automata to Decide a Restricted Clique Problem

Consider a graph of size $4n$ where $n \geq 2$ with numbered vertices $V = \{v_1, v_2, v_3, \ldots, v_{4n}\}$. We'll also have two subsets $V_1 \subseteq V$ and $V_2 \subseteq V$ where $|V_1| = n$, $|V_2| = 3n$, $V_1 = \{v_1, v_2, v_3, \ldots, v_n\}$ and $V_2 = \{v_{3n+1}, v_{3n+2}, v_{3n+3}, \ldots, v_{3n+n}\}$.

$$\begin{aligned}\Gamma(n) = \{g \mid g = {} & (V, E, V_1, V_2), \\ & V = \{v_1, v_2, v_3, \ldots, v_{4n}\}, \\ & V_1 \subseteq V, \\ & V_2 \subseteq V, \\ & V_1 = \{v_1, v_2, v_3, \ldots, v_n\}, \\ & V_2 = \{v_{n+1}, v_{n+2}, v_{n+3}, \ldots, v_{4n}\}\}\end{aligned} \tag{3}$$

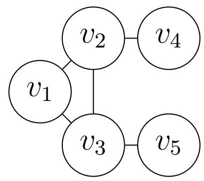
We observe that the problem P of finding a clique of size $n+2$ in $\Gamma(n)$ where 2 of the vertices are in V_1 and n of the vertices are in V_2 is an NP-Complete problem. So let $\Gamma_2(n)$ be all those graphs that satisfy P:

$$\Gamma_2(n) = \{g \mid g \text{ satisfies } P\} \tag{4}$$

2.1 Automaton construction

We now wish to create a finite-state automaton to decide the restricted Clique problem $\Gamma_2(n)$. So we encode the entire graph $g \in \Gamma_2(n)$ as a string, so we choose an adjacency matrix representation and encode g over the alphabet $\{0, 1\}$, i.e., a bit string.

The encoding of graphs is described below in some detail. We start with the adjacency matrix encoding of a graph $g \in \Gamma_2(n)$. The following graph only intended to illustrate the encoding process and does not capture the $4n$ argument we've been making:



Since the graph is undirected, the adjacency matrix is symmetric, making the top and bottom triangular matrices redundant, so we will encode only the upper triangular matrix, which looks like this:

	1	1	0	0	r_1
		1	1	0	r_2
			0	1	r_3
				0	r_4
					r_5

To make the exposition of the proof clearer, we'll use column-major order to flatten the matrix into a linear form. Consider a column c_i. We wish to treat each digit in c_i as a character and concatenate all those characters together to form a string s_i. So in our example, we start with an empty string since column 1 is on the diagonal and go from there:

$$s_1 = \epsilon \tag{5}$$

$$s_2 = 1 \tag{6}$$

$$s_3 = 11 \tag{7}$$

$$s_4 = 010 \tag{8}$$

$$s_5 = 0010 \tag{9}$$

Finally all the s_i are concatenated to form the encoding θ:

$$\theta = s_1 \cdot s_2 \cdot s_3 \cdots s_{4n-1} \tag{10}$$

Note that there is no string s_{4n} because the diagonal is not included in the upper triangular matrix. So our running example becomes:

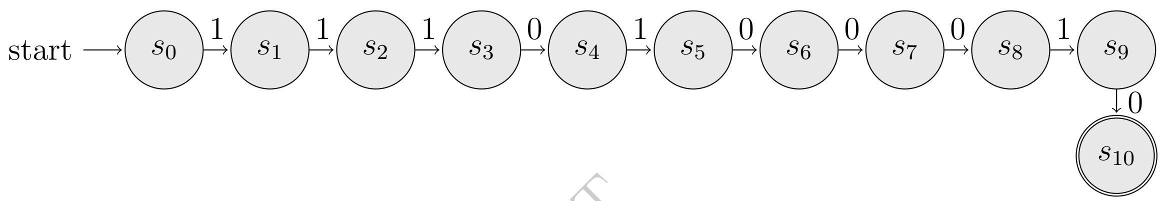
$$\theta = s_1 \cdot s_2 \cdot s_3 \cdot s_4 \cdot s_5 = \epsilon \cdot 1 \cdot 11 \cdot 010 \cdot 0010 = 1110100010 \tag{11}$$

Now define a function $e : V \times E \to \Sigma^*$ which maps graphs into their encodings as described above, so for example, $e(g) = \theta$. We extend e naturally to sets of graphs, so applying e to a set of graphs yields the set of encodings: $e(G) = \{e(g) \mid g \in G\}$.

Now let Θ_n be the encodings of those graphs for $\Gamma_2(n)$:

$$\Theta_n = \{\theta \mid g \in \Gamma_2(n) \text{ and } e(g) = \theta\}$$

There is an automaton to decide Θ_n. Now since Θ_n is a finite set of finite strings, we get from the argument in section 1.1 that there exists a trim acyclic deterministic finite-state automaton $M_n = (\Sigma, S, s_0, \delta, F)$ to decide it: M_n accepts θ iff $g \in \Gamma_2(n)$. In our running example, the automaton looks like this:



In what follows, we will take particular interest in the concatenation s_h of all strings up to n, where $\theta = s_h \cdot s_t$:

$$s_h = s_1 \cdot s_2 \cdot s_3 \cdots s_n \tag{12}$$

Lemma 2.1. *As n grows, the minimal automaton for $\Gamma_2(n)$ has an exponential number of states $\Omega(2^n)$ at level n in M_n.*

Proof. First, consider a graph $g \in \Gamma_2(n)$. So by definition, $g = (V, E, V_1, V_2)$. Suppose further that there exists a clique $C \subseteq V_2$ among the vertices V_2 where $|C| = n$ with no other cliques of size $\geq n$ in the whole graph g. Now pick out two vertices $v_i, v_j \in V_1$ among the vertices V_1. Suppose further that there are edges connecting v_i and v_j to each vertex in C: $R = \{(v, w) \mid v \in \{v_i, v_j\}, w \in C\}$ such that those edges are indeed among the edges in g: $R \subseteq E$. Finally suppose that there is an edge between v_i and v_j: $(v_i, v_j) \in E$.

Now consider a graph $g' = (V', E', V_1', V_2')$ which is equal to g in all respects except there is no edge between v_i and v_j: $(v_i, v_j) \notin E'$. This equality includes the edges among the vertices V_2 and V_2' which induce the subgraphs g_2 and g_2'. But since the edge (v_i, v_j) is different, the subgraphs g_1 and g_1' among by V_1 and V_1' are different.

This is the setup for the Myhill-Nerode argument to follow. Let $e(g) = s_h \cdot s_t$, and let $e(g') = s_h' \cdot s_t'$. Since $g_2 \cong g_2'$, we have $s_t = s_t'$, which means that $e(g') = s_h' \cdot s_t$. So their encodings reflect this as follows: since $g_1 \ncong g_1'$, we get $s_h \neq s_h'$ so $s_h \cdot s_t \in \Theta_n$ and $s_h' \cdot s_t \notin \Theta_n$. But these two membership relations show that s_t is a Myhill-Nerode distinguishing extension for s_h and s_h', so s_h and s_h' cannot be in the same Myhill-Nerode equivalence class and therefore by the Myhill-Nerode theorem, they cannot be in the same state in a minimal finite-state machine to recognize Θ_n. Since $|s_h| = n$ and there are 2^n distinct binary strings of length n, there are therefore 2^n possible distinct equivalence classes for the encodings s_h and therefore $\Omega(2^n)$ states at level n in M_n. $\square$

3 Conclusions

Theorem 3.1. $\mathsf{P} \neq \mathsf{NP}$.

Proof. By Lemma 2.1, we have the result that as n grows, the minimal automaton for $\Gamma_2(n)$ has an exponential number of states S_n at level n in M_n where $|S| = \Omega(2^n)$. By the Myhill-Nerode theorem, each such state corresponds to an equivalence class, a set of strings. Each of these sets of strings then corresponds to a Boolean formula by the construction in section 1.2. We also have that there is a formula ϕ_n for $\Gamma_2(n)$ by that same construction. We need to determine how big ϕ_n is.

Consider two states s_i and s_j at level n in M_n. Since states s_i and s_j are distinct, their corresponding languages L_i and L_j are distinct, which means their corresponding Boolean formulas p_i and p_j are distinct, in particular, it is not the case that both $p_i \vDash_I p_j$ and $p_j \vDash_I p_i$ if we go across all semantic interpretations I. So the set of all the p_i together is irreducible and exponential: $|\{p_i\}| = \Omega(2^n)$, and even if we were to replace each formula p_i with a single Boolean variable v_i and neglect the strings above the S_n, there are still an exponential number of independent variables v_i in ϕ_n, so ϕ_n is exponential in size: $|\phi_n| = \Omega(2^n)$.

$\square$

References

[1] Michael Sipser. *Introduction to the Theory of Computation.* Cengage, Boston, third edition, 2013.