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

Sheldon S. Nicholl

June 1, 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Not for publication!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.

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].

2 Automata to Decide a Restricted Clique Problem

A graph of size $2n$ has $\binom{2n}{n} = \Omega(4^n \cdot n^{-1/2})$ potential cliques in it of size $n \in \mathbb{N}$.

Let us now turn our attention to graphs of size $4n$ and call this set $\Gamma_1(n)$:

$$\Gamma_1(n) = \{g \mid g = (V, E) \text{ and } |V| = 4n\} \tag{3}$$

Consider a graph $G = (V, E)$ where $G \in \Gamma_1(n)$. What we want to do is restrict G to graphs with two halves: a "top half" and a "bottom half". So in the following construction we simply specify a graph which is the union of two disjoint isomorphic subgraphs. We capture this "top half, bottom half" idea with $\Gamma_2(n)$:

$$\begin{aligned}\Gamma_2(n) = \{g \mid\ & g' = (V, E) \in \Gamma_1(n),\\ & g_\text{T} \cong g_\text{B},\\ & g_\text{T} \cap g_\text{B} = \emptyset,\\ & g = g_\text{T} \cup g_\text{B},\ g = (V, E, g_\text{T}, g_\text{B})\}\end{aligned} \tag{4}$$

Define *clique parts* to be two sets, C_T and C_B such that $C_\text{T} \subseteq v(g_\text{T})$ and $C_\text{B} \subseteq v(g_\text{B})$. This lets us define yet another new structure Γ_3 as described below. So we restrict $\Gamma_2(n)$ further to $\Gamma_3(n)$ so that

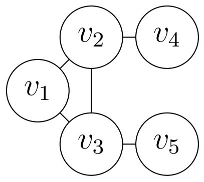
$$\begin{aligned}\Gamma_3(n) = \{g \mid\ & (V, E, g_\text{T}, g_\text{B}) \in \Gamma_2(n),\\ & g = (V, E, g_\text{T}, g_\text{B}, C_\text{T}, C_\text{B}),\\ & C_\text{T} \subseteq v(g_\text{T}), C_\text{B} \subseteq v(g_\text{B}),\\ & C_\text{T} \text{ and } C_\text{B} \text{ are cliques of size} \geq n\}\end{aligned} \tag{5}$$

We observe that $\Gamma_3(n)$ is an NP-Complete problem.

2.1 Automaton construction

We now wish to create a finite-state automaton to decide the restricted Clique problem $\Gamma_3(n)$. So we encode the entire graph $g \in \Gamma_3(n)$ 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 $g \in \Gamma_3(n)$. 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

Since we require $4n$ vertices in g, there are $4n$ rows in the upper triangular matrix. Consider a row r_i. We wish to treat each digit in r_i as a character and concatenate all those characters together to form a string s_i. So in our example,

$$s_1 = 1100 \tag{6}$$

$$s_2 = 110 \tag{7}$$

$$s_3 = 01 \tag{8}$$

$$s_4 = 0 \tag{9}$$

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:

$$\theta = s_1 \cdot s_2 \cdot s_3 \cdot s_4 = 1100 \cdot 110 \cdot 01 \cdot 0 = 1100110010 \tag{11}$$

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 $\Gamma_3(n)$:

$$\Theta_n = \{\theta \mid g \in \Gamma_3(n) \text{ and } e(g) = \theta\}$$

There is an automaton to decide Θ_n. Now since Θ_n is a finite set of finite strings, 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 $g \in \Gamma_3(n)$. In our running example, the automaton looks like this:

$$\begin{tikzcd}
\text{start} \arrow[r] & s_0 \arrow[r, "1"] & s_1 \arrow[r, "1"] & s_2 \arrow[r, "0"] & s_3 \arrow[r, "0"] & s_4 \arrow[r, "1"] & s_5 \arrow[r, "1"] & s_6 \arrow[r, "0"] & s_7 \arrow[r, "0"] & s_8 \arrow[r, "1"] & s_9 \arrow[d, "0"] \\
& & & & & & & & & & s_{10}
\end{tikzcd}$$

In what follows, we will take particular interest in the concatenation s_h of all strings up to $2n$, where $\theta = s_h \cdot s_p$:

$$s_h = s_1 \cdot s_2 \cdot s_3 \cdots s_{2n} \tag{12}$$

Lemma 2.1. *As n grows, the minimal automaton for $\Gamma_3(n)$ has an exponential number of states $\Omega(2^n)$ at level $2n$ in M_n.*

Proof. First, consider two distinct graphs g and g' where $g \in \Gamma_3(n)$ and $g' \notin \Gamma_3(n)$. So by the constraints imposed by $\Gamma_3(n)$, we can choose a g' outside of $\Gamma_3(n)$ where $\neg(g \cong g')$, in particular, $g = (V, E, g_{\mathrm{T}}, g_{\mathrm{B}}, C_{\mathrm{T}}, C_{\mathrm{B}})$ and $g' = (V', E', g'_{\mathrm{T}}, g'_{\mathrm{B}}, C'_{\mathrm{T}}, C'_{\mathrm{B}})$, where $\neg(g_{\mathrm{T}} \cong g'_{\mathrm{T}})$ but notably $g_{\mathrm{B}} \cong g'_{\mathrm{B}}$ and $C_{\mathrm{T}} \neq C'_{\mathrm{T}}$, that is, g_{T} and g'_{T} both contain cliques but the cliques are in different places. Note that there are $\binom{2n}{n}$ possible choices for the locations of the clique in g_{T}.

This is the setup for the Myhill-Nerode argument to follow. Let $e(g) = s_h \cdot s_p$, and let $e(g') = s_{h'} \cdot s_{p'}$. Since $g_{\mathrm{B}} \cong g'_{\mathrm{B}}$, we have $s_p = s_{p'}$, which means that $e(g') = s_{h'} \cdot s_p$. So their encodings reflect this as follows: since $\neg(g_{\mathrm{T}} \cong g'_{\mathrm{T}})$, $s_h \neq s_{h'}$ so $s_h \cdot s_p \in \Theta_n$ and $s_{h'} \cdot s_p \notin \Theta_n$. But these two membership relations show that s_p 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 $|s_h| = 2n$ and there are $\binom{2n}{n}$ possible choices for C_{T}, there are therefore at least $\binom{2n}{n}$ distinct equivalence classes for the encodings s_h and therefore $\Omega(4^n \cdot n^{-1/2}) = \Omega(2^n)$ states at level $2n$ in M_n. □

3 Union of Concatenated Languages

Lemma 3.1. *If $k, m, i, j, p \in \mathbb{N}$ and $k, m, i, j, p \geq 1$, with alphabet $\{0, 1\}$ and non-empty languages $L_i \subseteq \Sigma^k$ and $L'_i \subseteq \Sigma^m$, where all of the L'_i are mutually disjoint, that is, for*

$i \neq j$, we have $L'_i \cap L'_j = \emptyset$, then the number of states in the minimal trim deterministic finite-state automaton for the language

$$\bigcup_{i=1}^{p} L_i \circ L'_i$$

at level k is p.

Proof. By the Myhill-Nerode theorem, the only distinguishing extensions come from the L'_i, and there are only p mutually disjoint sets of those, so there are only p states at level k of the automaton. All strings not in any of the L_i will land in the sink state, which is trimmed out of the minimal trim automaton. □

4 Conclusions

Proof Idea: Using the assumption that there exists an NP-Complete sparse language, we will use that to build a different automaton for Clique with fewer states at level $2n$ based on the same restricted Clique language $\Gamma_3(n)$. This is a contradiction, so there can be no such NP-Complete sparse language.

Theorem 4.1. $\mathsf{P} \neq \mathsf{NP}$.

Proof. The proof technique here is Reductio ad Absurdum.

Define Γ_4, the set of all graphs in the top half of Γ_3:

$$\Gamma_4 = \{g_{\mathrm{T}} \mid g = (V, E, g_{\mathrm{T}}, g_{\mathrm{B}}, C_{\mathrm{T}}, C_{\mathrm{B}}) \in \Gamma_3\} \tag{13}$$

Similarly, define Γ_5, the set of all graphs in the bottom half of Γ_3:

$$\Gamma_5 = \{g_{\mathrm{B}} \mid g = (V, E, g_{\mathrm{T}}, g_{\mathrm{B}}, C_{\mathrm{T}}, C_{\mathrm{B}}) \in \Gamma_3\} \tag{14}$$

Let there be a finite alphabet Σ_2. Suppose there exists an NP-Complete sparse language $L \subseteq \Sigma_2^*$ for Clique. Since L is NP-Complete, there is a polynomial-time reduction $r(g)$ which takes a graph g and maps it to an element $x \in L$ such that

$$(g \in \text{Clique}) \longleftrightarrow (r(g) \in L) \tag{15}$$

We will use the assumed existence of a sparse language to build another deterministic trim finite state automaton to decide Γ_3, exactly the same language as mentioned above in Lemma 2.1. We now define the inverse function r^{-1} for r which takes an element $a \in L$ and maps it to the collection of all graphs which map to a under r:

$$r^{-1}(a) = \{g \mid r(g) = a\} \tag{16}$$

We now gather all the elements of L which come from Γ_5 via r into a set called L_5. Note that since L is a sparse language, $|L_5|$ is polynomially bounded, that is, for some fixed c, $|L_5| = O(n^c)$:

$$L_5 = \{a \mid g \in \Gamma_5, a = r(g)\} \tag{17}$$

From L_5 we can now create the set of sets of graphs in Γ_5 corresponding to each member of L_5.

$$\Lambda = \{G \mid a \in L_5, G = r^{-1}(a), G \subseteq \Gamma_5\} \tag{18}$$

We observe that since r is a function, all the G's are distinct. Let $t = |L_5|$. We can give each G in Λ an index as follows:

$$\Lambda = \{G_1, G_2, \ldots, G_t\} \tag{19}$$

Since Γ_4 and Γ_5 are isomorphic, we build an isomorphic set Λ' the same way we built Λ:

$$\Lambda' = \{G'_1, G'_2, \ldots, G'_t\} \tag{20}$$

We are now in a position to apply Lemma 3.1. In that lemma, let $p = t = |L_5|$, let $k = m = |e(g)|$ for any graph in either Λ or Λ' since all those graphs are the same size, let $L_i = e(G'_i)$ for $G'_i \in \Lambda'$, and let $L'_i = e(G_i)$ for $G_i \in \Lambda$. So by that lemma, there are only $p = |L_5| = O(n^c)$ states for some fixed c at level $2n$ of the corresponding automaton which decides $\Gamma_3(n)$.

But by Lemma 2.1, there are $\Omega(2^n)$ states in the minimal machine at level $2n$ of the automaton for the exact same $\Gamma_3(n)$ problem. This is a contradiction. So by Reductio ad Absurdum, our original assumption that there exists a sparse NP-Complete language is false.

The biconditional form of Mahaney's Theorem states that there is a sparse NP-Complete language iff $\mathsf{P} = \mathsf{NP}$. Since we've just shown there is no sparse NP-Complete language, $\mathsf{P} \neq \mathsf{NP}$. $\square$

References

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