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

Sheldon S. Nicholl

January 22, 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Not for publication!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 as a function of the length of its input. On the other hand, we also show that the automaton for any sparse language grows only polynomially with its input. Finally, using the assumption that there exists an NP-Complete sparse language, we use the polynomial-time reduction to build an automaton for Clique smaller than the minimal automaton. 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 notation $g; e$ refers to the graph which results from adding edge e to graph g.

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. We extend δ to deal with whole strings with the extended transition function $\hat{\delta} : Q \times \Sigma^* \to Q$ which is defined inductively where $a \in \Sigma$ and $\hat{\delta}(q, \epsilon) = q$ and $\hat{\delta}(q, ax) = \hat{\delta}(\delta(q, a), x)$. All automata in this paper are presumed to be minimal.

If $a \in \Sigma$ and the state $s' = \delta(s, a)$ exists, then we can view the pair (s, s') as a a transition in A. So the automaton defines a directed graph with vertices S and edges $\{(s, s') \mid \ s \in S \text{ and } a \in A\}$. 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 split V into two halves: the "top half" and the "bottom half". We refer to vertices directly via their indices. So let the top be $\{1_\mathrm{T}, 2_\mathrm{T}, 3_\mathrm{T}, , ..., 2n_\mathrm{T}\}$ and let the bottom be $V_\mathrm{B} = \{1_\mathrm{B}, 2_\mathrm{B}, 3_\mathrm{B}, , ..., 2n_\mathrm{B}\}$. So then let V be the disjoint union $V_\mathrm{T} \sqcup V_\mathrm{B}$. This lets us define a new graph structure $G_2 = (V, E, V_\mathrm{T}, V_\mathrm{B})$. We capture this "top half, bottom half" idea with $\Gamma_2(n)$:

$$\begin{aligned}\Gamma_2(n) = \{g \mid (V, E) \in \Gamma_1(n), g = (V, E, V_\mathrm{T}, V_\mathrm{B}),\\ V_\mathrm{T} = \{1_\mathrm{T}, 2_\mathrm{T}, 3_\mathrm{T}, , ..., 2n_\mathrm{T}\},\\ V_\mathrm{B} = \{1_\mathrm{B}, 2_\mathrm{B}, 3_\mathrm{B}, , ..., 2n_\mathrm{B}\}, V = V_\mathrm{T} \sqcup V_\mathrm{B}\}\end{aligned} \tag{4}$$

Define *clique parts* to be two sets, C_T and C_B such that $C_\mathrm{T} \subseteq V_\mathrm{T}$ and $C_\mathrm{B} \subseteq V_\mathrm{B}$. This lets us define yet another new structure $G_3 = (V, E, V_\mathrm{T}, V_\mathrm{B}, C_\mathrm{T}, C_\mathrm{B})$.

We restrict $\Gamma_2(n)$ further to another set $\Gamma_3(n)$ so that

$$\begin{aligned}\Gamma_3(n) = \{g \mid (V, E, V_\mathrm{T}, V_\mathrm{B}) \in \Gamma_2(n),\\ g = (V, E, V_\mathrm{T}, V_\mathrm{B}, C_\mathrm{T}, C_\mathrm{B}),\\ C_\mathrm{T} \subseteq V_\mathrm{T}, C_\mathrm{B} \subseteq V_\mathrm{B},\\ C_\mathrm{T} \text{ and } C_\mathrm{B} \text{ are cliques of size exactly } n,\\ V_\mathrm{T} \text{ and } V_\mathrm{B} \text{ contain no other cliques of size } \geq n\}\end{aligned} \tag{5}$$

Definition of *connectors*: connectors are edges in G that always start in C_T and go to vertices in V_B, so they span the gap between V_T and V_B. More importantly, they allow the formation of a size $2n$ clique if C_B happens to land there. That is, there will be a clique of size $2n$ iff C_T and C_B line up. We build these connectors in several steps. First, we define a function f to map directly from a vertex in V_T to a vertex in V_B:

$$f(i_\mathrm{T}) = i_\mathrm{B} \tag{6}$$

So for example, $f(1_\mathrm{T}) = 1_\mathrm{B}$, $f(2_\mathrm{T}) = 2_\mathrm{B}$, and so on.

We then define a function g analogous to f which maps a set of vertices in V_T to a set of vertices in V_B:

$$g(S_\mathrm{T}) = \{i_\mathrm{B} \mid v \in S_\mathrm{T}, f(v) = i_\mathrm{B}\} \tag{7}$$

So for example, $g(\{1_\mathrm{T}, 2_\mathrm{T}, 3_\mathrm{T}\}) = \{1_\mathrm{B}, 2_\mathrm{B}, 3_\mathrm{B}\}$. We are now finally ready to define the connectors like a cross-product between a set S_T and its image $g(S_\mathrm{T})$:

$$h(S_\mathrm{T}) = \{(v, w) \mid v \in S_\mathrm{T}, w \in g(S_\mathrm{T})\} \tag{8}$$

So we further restrict the edges of $\Gamma_3(n)$ to always include connectors, yielding $\Gamma_4(n)$; this is the restricted Clique problem we will be studying:

$$\Gamma_4(n) = \{g \mid g \in \Gamma_3(n), g = (V, E, V_\mathrm{T}, V_\mathrm{B}, C_\mathrm{T}, C_\mathrm{B}), h(C_\mathrm{T}) \subseteq E\} \tag{9}$$

Here are two examples:

2.1 Connectors that form a clique of size $2n$

Let $n = 3$. Then $V_\mathrm{T} = \{1_\mathrm{T}, 2_\mathrm{T}, 3_\mathrm{T}, 4_\mathrm{T}, 5_\mathrm{T}, 6_\mathrm{T}\}$ and $V_\mathrm{B} = \{1_\mathrm{B}, 2_\mathrm{B}, 3_\mathrm{B}, 4_\mathrm{B}, 5_\mathrm{B}, 6_\mathrm{B}\}$. Let's choose $C_\mathrm{T} = \{1_\mathrm{T}, 3_\mathrm{T}, 5_\mathrm{T}\}$. In computing g, $g(C_\mathrm{T}) = \{1_\mathrm{B}, 3_\mathrm{B}, 5_\mathrm{B}\}$. So the connectors are:

$$h(C_\mathrm{T}) = \{(1_\mathrm{T}, 1_\mathrm{B}), (1_\mathrm{T}, 3_\mathrm{B}), (1_\mathrm{T}, 5_\mathrm{B}), (3_\mathrm{T}, 1_\mathrm{B}), (3_\mathrm{T}, 3_\mathrm{B}), (3_\mathrm{T}, 5_\mathrm{B}), (5_\mathrm{T}, 1_\mathrm{B}), (5_\mathrm{T}, 3_\mathrm{B}), (5_\mathrm{T}, 5_\mathrm{B})\} \tag{10}$$

Let's choose $C_\mathrm{B} = \{1_\mathrm{B}, 3_\mathrm{B}, 5_\mathrm{B}\}$. C_B is guaranteed to be a clique on its own already. Now with this set of connectors, $C_\mathrm{T} \cup C_\mathrm{B}$ form a clique of size $2n$: that is, vertices $\{1_\mathrm{T}, 3_\mathrm{T}, 5_\mathrm{T}, 1_\mathrm{B}, 3_\mathrm{B}, 5_\mathrm{B}\}$ form a clique of size 6.

2.2 Connectors that do not form a clique of size $2n$

As before, let $n = 3$, $C_\mathrm{T} = \{1_\mathrm{T}, 3_\mathrm{T}, 5_\mathrm{T}\}$, $V_\mathrm{T} = \{1_\mathrm{T}, 2_\mathrm{T}, 3_\mathrm{T}, 4_\mathrm{T}, 5_\mathrm{T}, 6_\mathrm{T}\}$, $V_\mathrm{B} = \{1_\mathrm{B}, 2_\mathrm{B}, 3_\mathrm{B}, 4_\mathrm{B}, 5_\mathrm{B}, 6_\mathrm{B}\}$, and the connectors are the same as equation 10.

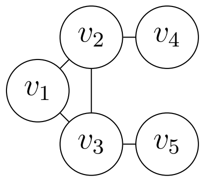
But now change C_B to be $C_\mathrm{B} = \{10_\mathrm{B}, 11_\mathrm{B}, 12_\mathrm{B}\}$.

C_T and C_B are as always guaranteed to be cliques on their own already. But with this set of connectors, $C_\mathrm{T} \cup C_\mathrm{B}$ do not form a clique of size $2n$: that is, vertices $\{1_\mathrm{T}, 3_\mathrm{T}, 5_\mathrm{T}, 10_\mathrm{B}, 11_\mathrm{B}, 12_\mathrm{B}\}$ do not form a clique of size 6 because we've excluded choices of G that place cliques among the vertices in C_B unless C_B is placed there explicitly, which in this case, it isn't.

2.3 Automaton construction

We now wish to create a finite-state automaton to decide the restricted Clique problem $\Gamma_4(n)$. So we encode the entire graph $G \in \Gamma_4(n)$ 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 $G \in \Gamma_4(n)$. The following graph is purely illustrative 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,

$$s_1 = 1100 \tag{11}$$

$$s_2 = 110 \tag{12}$$

$$s_3 = 01 \tag{13}$$

$$s_4 = 0 \tag{14}$$

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

$$\theta = s_1 \cdot s_2 \cdot s_3 \cdots s_{4n-1} \tag{15}$$

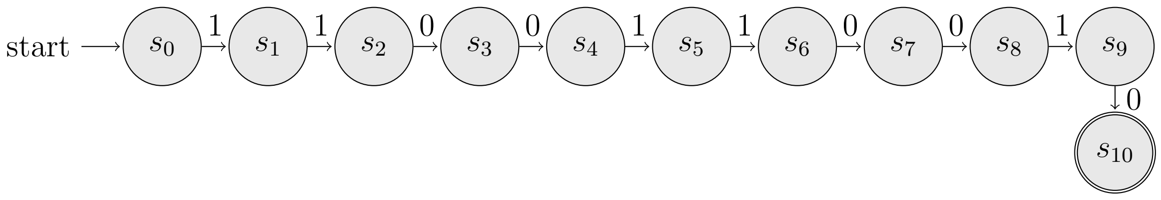
Which in our running example becomes:

$$\theta = s_1 \cdot s_2 \cdot s_3 \cdot s_4 = 1100 \cdot 110 \cdot 01 \cdot 0 = 1100110010 \tag{16}$$

Now let Θ_n be the encodings of those graphs:

$$\Theta_n = \{\theta \mid g \in \Gamma_4(n) \text{ and } \theta \text{ encodes } g\}$$

There is an automaton to decide Θ_n. Now since Θ_n is a finite set of finite strings, there exists an acyclic deterministic finite-state automaton $M_n = (\Sigma, S, s_0, \delta, F)$ to decide it: M_n accepts θ iff $g \in \Gamma_4(n)$. 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 $2n$, where $\theta = s_h \cdot s_p$:

$$s_h = s_1 \cdot s_2 \cdot s_3 \cdots s_{2n} \tag{17}$$

Now consider that level L_n of M_n where s_h has been read into M_n. We are guaranteed by our construction that s_h contains the encoding of a clique, C_T.

Lemma 2.1. *As n grows, the minimal automaton for $\Gamma_4(n)$ has an exponential number of states $\Omega(2^n)$ at level $2n$.*

Proof. First, consider two distinct graphs g and g' where $g \in \Gamma_4(n)$ and $g' \in \Gamma_4(n)$. Choose g such that its two cliques C_T and C_B form a clique of size $2n$, and choose g' such that C'_T is located in a different place compared to C_T but C'_B is in the same place as C_B. Finally, let $V_\mathrm{B} \in g$ and $V'_\mathrm{B} \in g'$, but set $V_\mathrm{B} = V'_\mathrm{B}$ so that not only their vertices but their edges are identical. So by the constraints imposed by $\Gamma_4(n)$, g' cannot have a clique of size $2n$. Note that there are $\binom{2n}{n}$ possible choices for C_T.

This is the setup for the Myhill-Nerode argument to follow. Let the string $s_h \cdot s_p$ be the encoding of g, and let $s_{h'} \cdot s_{p'}$ be the encoding of g'. Since $V_\mathrm{B} = V'_\mathrm{B}$ all the way down to the edges, we have $s_p = s_{p'}$. So $s_{h'} \cdot s_p$ is the encoding of g'. Since g has a clique of size $2n$ and g' does not, their encodings reflect this as follows: $s_h \cdot s_p \in \Theta_n$ 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 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 $\Omega(4^n \cdot n^{-1/2})$ states at level L_n in M_n. □

3 Automata for Sparse Languages

Lemma 3.1. *For all sparse languages L and for all n, there exists a constant c such that the subset $L_n \subseteq L$ produces a minimal DFA such that each level has only $O(n^c)$ states.*

Proof. By the definition of a sparse language,

$$|L_n| = O(n^c)$$

Now if we take the set of all prefixes at a certain depth d,

$$S_n = \{w \mid wv \in L_n, |w| = d\}$$

S_n is also polynomially bounded:

$$|S_n| = O(n^c)$$

Note that none of the prefixes outside of S_n is distinguishable from any other since they're all outside the language. So by the Myhill-Nerode theorem, there are only a polynomial number of states at each depth in the minimal automaton, even if all the prefixes are distinguishable from one another by different suffixes. □

Corollary 3.1.1. *For all sparse languages L and for all n, there exists a constant c such that the subset $L_n \subseteq L$ produces a minimal DFA that has only $O(n^c)$ states.*

Proof. By Lemma 3.1, the size of each level of this automaton has only $O(n^d)$ states for some constant d. Since there are only n levels in the whole automaton, the total number of states is still polynomially bounded: $O(n^c)$. □

4 Conclusions

Proof Idea: Using the assumption that there exists an NP-Complete sparse language, we will use that to build a small automaton for Clique, smaller than the minimal finite automaton based on the restricted Clique language $\Gamma_4(n)$. 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. Suppose there exists an NP-Complete sparse language L. Since L is NP-Complete, there is a polynomial-time reduction which takes a graph g and maps it to an element of $x \in L$. Let's call this function $r(g)$. We note that $r(g)$ is correct and complete, so $g \in \Gamma_4(n)$ iff $r(g) \in L$.

Let's restrict L to inputs of size p where p is big enough to accommodate all the input graphs with up to n edges:

$$p = \max\{r(g) \mid |g| \leq n\}$$

For notational convenience later, we'll call this restriction $L(p)$. Using the standard construction described in section 1.1, we create a minimal finite-state automaton for $L(p)$; we'll call it $M(L(p)) = (\Sigma_L, S_L, F_L, \delta_L, s_{L0})$. So $M(L(p))$ accepts x iff $x \in L(p)$.

Let's define the function $s(x) = \hat{\delta}(s_{L0}, x)$; this takes us from the start state of $M(L(p))$ to the state in $M(L(p))$ corresponding to input x. So $s(x)$ takes $x \in L(p)$ and maps it to a state s in $M(L(p))$.

Our job now is to build a machine for Clique based on the machine for L: $X = (\Sigma_X, S_X, F_X, \delta_X, s_{X0})$.

Alphabet, equation 19: The alphabet for Clique is $\Sigma_C = \{0, 1\}$. The alphabet for the X machine is the same as the machine for clique because they are reading in the same inputs: graphs.

States: Each state in the L machine $\sigma \in S_L$ is an equivalence class in L. But to avoid circularities as we translate from the Clique language Σ_C^*, we'll divide each such equivalence class into smaller equivalence classes based also on the length of the input string in Σ_C^*:

$$\theta(\sigma, k) = \{g \mid g \in \Sigma_C^*, |g| = k, k \leq n, \sigma = s(r(g)), \sigma \in S_L\} \tag{18}$$

So each new equivalence class is a set of graph encodings, as shown by the first expression in equation 18: $g \in \Sigma_C^*$. The next expression, $|g| = k$, limits the size of the graph encoding to one fixed size k. After that, the next expression, $k \leq n$, limits the size of k to be less than n, the size of the overall input graph encoding we're considering. The next expression connects the language of graphs to states in the automaton for the sparse language, $M(L(p))$, so $\sigma = s(r(g))$ denotes that state σ to be the state in the sparse automaton corresponding to what one would get by translating g with r into the sparse language and then mapping that string in the sparse language into the state in the sparse automaton. Finally, the last clause, $\sigma \in S_L$, ensures that σ is really a state in the sparse automaton. So, briefly, $\theta(\sigma, k)$ is a set of graph encodings of size k which map to σ. Finally, $\theta(\sigma, k)$ is an equivalence class because σ is an equivalence class.

Each new equivalence class becomes a state in equation 20 below.

Transition function, equation 21: the transition function for the X machine is more complicated. What we are going to do is read in a graph, translate to the sparse language, then find the resulting state in the sparse automaton. This gives us the starting state in the transition we are about to create. We then take that same input graph and add a new edge. We then go through this same translation to the sparse language and then find the resulting state in the sparse automaton. This gives us the ending state of our new transition. Finally we just add the edge to complete the whole transition.

The start state of the X machine is the same as the start state in the sparse automaton (equation 23).

Similarly, the final states of the X machine (equation 22) are the states in S_X corresponding to final states in F_L.

$$\Sigma_X = \Sigma_C \tag{19}$$
$$S_X = \{\theta(\sigma, k) \mid \sigma \in S_L, k \leq n\} \tag{20}$$
$$\delta_X = \{(\theta(s(r(g)), k-1), e) \to \theta(s(r((g;e))), k) \mid |g| = k-1, k < n\} \tag{21}$$
$$F_X = \{\theta(\sigma, k) \mid \theta(\sigma, k) \in S_X, \sigma \in F_L\} \tag{22}$$
$$s_{X0} = s_{L0} \tag{23}$$

So we've built a machine X that accepts iff the original Clique machine C accepts. By Corollary 3.1.1, S_X has only $O(n^c)$ states in the whole machine for some fixed c. But by Lemma 2.1, there are $\Omega(2^n)$ states in the minimal machine for a more restricted problem. This is a contradiction. So by Reductio ad Absurdum, our original assumption that there exists a sparse NP-Complete language is wrong.

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

□

References

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