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

Sheldon S. Nicholl

June 16, 2022

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Not for publication!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 Clique (for a fixed size n) 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 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: $L_n = \{s \mid s \in L \text{ and } |s| = n\}$.

The notation g_n refers to a graph with n edges. G_n is the set of all graphs with up to n edges. The notation $g; e$ refers to the graph which results from adding edge e to graph g.

A *clique* is a subset of the vertices of a graph such that every pair in the subset is adjacent. A *subclique* is a clique smaller than a given desired size. If g is a graph with n edges and $k > 1$ is a constant, we define $P(g, k)$ to be the Clique decision problem where the answer is "yes" if there is a clique of size $\lceil \frac{n}{k} \rceil$ in g, "no" otherwise. We observe that $P(g, k)$ is an NP-Complete problem. We define Clique(n, k) to be the set of graphs with n edges such that $P(g, k)$ is “yes”.

Following Sipser [3], a deterministic finite automaton (DFA) is a tuple $A = (\Sigma, S, F, \delta, s_0)$ 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 acyclic [1], [2].

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.

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 [3].

1.2 Defining Automata

We will use the adjacency matrix encoding for Clique(n, k). So in particular, there is a finite alphabet $\Sigma_C = \{0, 1\}$ to represent Clique(n, k). By the construction in Section 1.1, we can create the automaton:

$$C = (\Sigma_C, S_C, F_C, \delta_C, s_{C0}) \tag{1}$$

So C accepts x iff $x \in$ Clique(n, k). For clarity, we observe that this automaton reads one edge at a time from a bit string $\{0, 1\}^n$.

2 Results

Lemma 2.1. *For a fixed $k \geq 1$ as n grows, the minimal automaton for Clique(n, k) has an exponential number of states $\Omega(2^{\sqrt{n}/2k})$ at level $\lceil \frac{n}{2} \rceil$.*

Proof. The Myhill-Nerode theorem will be used to prove this. We will use the usual idea of a *distinguishing extension* to set this up. In general, given a language L and strings x and y where $x \neq y$, a distinguishing extension is a string z such that only one of the two strings xz or yz is in L.

In our case, let the language $L_n \subseteq \Sigma_C^*$ be the representation of Clique(n, k) as described above. We set up x and y as two different prefixes of instances of Clique(n, k) which don't themselves contain cliques. In more detail, we first note that in a graph of v vertices, there are potentially $\binom{v}{m}$ cliques of size m. Also, $\binom{2w}{w} = \Omega(2^w)$, and more generally, $\binom{2kw}{w} = \Omega(2^w)$ for $k \geq 1$. So in a graph with $2 * k * w$ vertices where we can choose cliques of size w, the number of choices is $\Omega(2^w)$. This means, then, in a graph with v vertices where we can choose cliques of size $\frac{v}{2k}$, the number of choices is $\Omega(2^{v/2k})$. And since the relationship between edges and vertices in a graph is at worst quadratic, the number of choices is $\Omega(2^{\sqrt{n}/2k})$.

So we can choose among these size $\frac{v}{2k}$ subcliques to form the prefix strings x and y.

The string z, then, is a suffix such that $xz \in L_n$ and $yz \notin L_n$. This is accomplished by choosing z so that it completes the subclique encoded in x but not in y. Since there are $\Omega(2^{\sqrt{n}/2k})$ subcliques encoded in strings x, y, and z, we have $\Omega(2^{\sqrt{n}/2k})$ states in the minimal automaton at level $\lceil \frac{n}{2} \rceil$ by the Myhill-Nerode theorem. □

Corollary 2.1.1. *For a fixed $k \geq 1$ as n grows, the minimal automaton for Clique(n, k) has an exponential number of states $\Omega(2^{\sqrt{n}/2k})$.*

Proof. By Lemma 2.1, a single level of this minimal automaton has $\Omega(2^{\sqrt{n}/2k})$ states. So therefore the size of the whole machine must be bounded below by at least this number. □

Lemma 2.2. *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 2.2.1. *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 2.2, 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: $O(n^c)$. □

3 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 language of Clique(n, k). This is a contradiction, so there can be no such NP-Complete sparse language.

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

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 $g \in$ Clique(n, k) 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 3: 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^*:

$$\theta(\sigma, k) = \{g \mid g \in \Sigma_C^*, |g| = k, k \leq n, \sigma = s(r(g)), \sigma \in S_L\} \tag{2}$$

So each new equivalence class is a set of graphs, as shown by the first expression in equation 2: $g \in \Sigma_C^*$. The next expression, $|g| = k$, limits the size of the graph 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 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 graphs 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 4 below.

Transition function, equation 5: 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 7).

Similarly, the final states of the X machine (equation 6) are the same as the final states of the sparse automaton.

$$\Sigma_X = \Sigma_C \tag{3}$$

$$S_X = \{\theta(\sigma, k) \mid \sigma \in S_L, k \leq n\} \tag{4}$$

$$\delta_X = \{(\theta(s(r(g)), k-1), e) \rightarrow \theta(s(r((g; e))), k) \mid |g| = k-1, k < n\} \tag{5}$$

$$F_X = F_L \tag{6}$$

$$s_{X0} = s_{L0} \tag{7}$$

So we've built a machine X that accepts iff the original Clique machine C accepts. By Corollary 2.2.1, S_X has only $O(n^c)$ states in the whole machine for some fixed c. But by Corollary 2.1.1, there are $\Omega(2^{\sqrt{n}/2k})$ states in the minimal machine. 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, $\mathsf{P} \neq \mathsf{NP}$.

$\square$

References

[1] J. Bubenzer. Minimization of acyclic dfas. In *Proceedings of the Prague Stringology Conference 2011*, NordiCHI, pages 207–216. Prague Stringology Club, Czech Technical University, 2011.

[2] D. Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(2):181–189, 1992.

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