On the Fine Grained Complexity of Finite Automata Non-Emptiness of Intersection

Mateus de Oliveira Oliveira[*1] **and Michael Wehar**[2]

1 **University of Bergen, Bergen, Norway**
mateus.oliveira@uib.no
2 **CapSen Robotics**[†]**, Pittsburgh, PA, USA**
mwehar@buffalo.edu

Abstract

In the DFA non-emptiness of intersection problem (DFA-NEI) we are given a list $\langle \mathcal{A}_1, ..., \mathcal{A}_k \rangle$ of DFA's over a common alphabet Σ, and the goal is to determine whether $\bigcap_{i=1}^{k} \mathcal{L}(\mathcal{A}_i) \neq \emptyset$. First, using a suitable binary encoding for finite automata, we provide an unconditional space complexity lower bound showing that DFA-NEI is not in NSPACE($o(n/\log n)$). The previous best lower-bound for DFA-NEI was that this problem does not belong to NSPACE($o(n/(\log n)(\log\log n))$) when the input automata are specified straightforwardly by their list of transitions.

Next, we study the fine-grained complexity of k-DFA-NEI, the variant of DFA-NEI where the number k of input automata is fixed. We provide conditional time complexity lower bounds showing that slight improvements on state of the art parameterized algorithms for k-DFA-NEI would have remarkable complexity theoretic consequences.

Finally, we study the fine-grained complexity of DFA-NEI when restricted to DFA's over a unary alphabet. We show that emptiness of intersection for two automata operating over a unary alphabet can be solved in time $O(n \log n)$ and emptiness of intersection for three automata operating over a unary alphabet can be solved in time $O(n^{\alpha/2})$ for every $\alpha > 2$ such that triangle finding can be solved in time $O(n^{\alpha})$.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.3 Complexity Measures and Classes, F.4.3 Formal Languages

Keywords and phrases Finite Automata, Intersection Non-Emptiness, Fine Grained Complexity, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.

1 Introduction

In this work we analyse the DFA non-emptiness of intersection problem (DFA-NEI) from a complexity theoretic perspective. In this problem, we are given a list $\langle \mathcal{A}_1, ..., \mathcal{A}_n \rangle$ of finite automata over a common alphabet Σ, and are asked to determine whether the intersection of the languages $\mathcal{L}(\mathcal{A}_1), ..., \mathcal{L}(\mathcal{A}_n)$ is non-empty. This problem is well known to be PSPACE complete [18]. For each constant k, let k-DFA-NEI be the restrictions of DFA-NEI where the input consists of k DFA's over $\Sigma = \{0, 1\}$. We note that this problem can be solved in time $O(n^k)$ using the standard product construction.

* Mateus de O. Oliveira acknowledges support from the Bergen Research Foundation.
† The second author is currently affiliated with CapSen Robotics and University of Pittsburgh, although this work was pursued while a PhD student at University at Buffalo.

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Daniel Marx, and Don Sannella;
Article No. ; pp. :1–:17


Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Karakostas, Lipton, and Viglas showed that if non-emptiness of intersection for k deterministic finite automata (k-DFA-NEI) could be solved in time $n^{o(k)}$, then NTIME$[n] \subseteq$ DTIME$[2^{\delta n}]$ for every $\delta > 0$ [16]. Additionally, they showed that such an upper bound would imply faster algorithms for subset sum and integer factorization. Further, improving their results, it was shown that if k-DFA-NEI could be solved in time $n^{o(k)}$, then $NL \neq P$ [31, 33]. The parameterized complexity of emptiness of intersection for DFA's has also been addressed in [17, 19, 28, 25, 11].

We focus on a reduction from simulating a nondeterministic space bounded Turing machine to DFA-NEI. First, we fill in some gaps in the literature by showing that:

1. If we can solve k-DFA-NEI in time $n^{o(k)}$, then NSPACE$[n] \subseteq$ DTIME$[2^{o(n)}]$ [30].[1]
2. If there exists $k \geq 2$ and $\varepsilon > 0$ such that k-DFA-NEI in time $O(n^{k-\varepsilon})$, then NSPACE$[n] \subseteq$ DTIME$[2^{\delta \cdot n}]$ for some $\delta < 1$ [1, 33].

Then, we consider unconditional space complexity lower bounds for DFA-NEI. More specifically, we show that by treating the input for DFA-NEI as a suitably encoded binary string of length n, DFA-NEI cannot be solved in NSPACE$[o(n/\log n)]$. Previously, the best known unconditional lower bound was that DFA-NEI does not belong to NSPACE$[o(n/(\log n)(\log\log n))]$ [31]. In other words, we improve the best previously known lower bound by a $\log\log n$ factor. This is important because any asymptotic improvement upon this lower bound would show unconditionally[2] that DFA-NEI cannot be solved in deterministic linear time. We note that when proving sublinear lower bounds for the space complexity of computational problems the encoding of the input instances may affect the lower bounds. Intuitively, the greater number of bits necessary to encode the input, the worse are the lower bounds. In particular, the lower bound in [31] was proven with respect to a straightforward encoding of the input automata as lists of transitions. In this work we introduce a slightly more succinct representation for finite automata. The difficulty in proving the new lower bound relies in showing that the main steps of the proof in [31] can be lifted to the new encoding. We note however that this lifting is non-trivial and will require a careful analysis of the involved steps.

Next, we show that if k-DFA-NEI can be solved in time $n^{o(k)}$, then we obtain faster algorithms for satisfiability of Boolean formulas and satisfiability of bounded fan-in Boolean circuits. In particular, we show that if k-DFA-NEI can be solved in time $n^{o(k)}$, then satisfiability of Boolean circuits of depth $O(n)$ and size $2^{o(n)}$ can be solved in time $2^{o(n)}$. This is a consequence which is considered very unlike by complexity theorists. For instance, a great deal of lower bounds for computational problems are proven under the exponential time hypothesis (ETH) [14], which states that satisfiability of CNF formulas with n-inputs cannot be solved in time $2^{o(n)}$. We note that CNF formulas of polynomial size are a very weak model of computation, which are unable for instance to compute the parity of their input bits. On the other hand, circuits of linear depth can already simulate complicated cryptographic primitives. Going further, we show that if there exists $k \geq 2$ such that k-DFA-NEI can be solved in time $n^{k-\varepsilon}$ for some $\varepsilon > 0$, then satisfiability of fan-in-2 n-input Boolean formulas of sub-exponential size can be solved in time $2^{(1-\delta)n}$ for some $\delta > 0$. This would contradict the strong exponential time hypothesis (SETH) [14] under which many lower bounds in parameterized complexity theory are based. Finally, using these results together with results obtained within the context of William's algorithms vs lower bounds framework [34, 35, 3, 2],

[1] This work was not formally published.
[2] Using the well known fact that deterministic linear time can be simulated in $DSPACE[n/\log n]$ [12].

we infer that faster algorithms for k-DFA-NEI would imply non-uniform circuit lower bounds that are much sharper than what is currently known.

We conclude our work by establishing connections between automata intersection and hard problems in polynomial time. In particular, we show that there exists a fine-grained reduction from triangle finding for a graph with n vertices and m edges to non-emptiness of intersection for two DFA's where the first DFA has $m\log(n)$ states and the second DFA has $n\log(n)$ states. Additionally, we show that for each k, there exists a fine-grained reduction from 3SUM for a set of n numbers in the range $[-n^k, n^k]$ to non-emptiness of intersection for three DFA's over a binary alphabet where each DFA has at most $kn\log(n)$ states.

On the positive side, we show that emptiness of intersection for two DFA's operating over a unary alphabet can indeed be solved in time $O(n\log(n))$. In other words, this problem admits an algorithm that is much faster than the trivial $O(n^2)$ algorithm. Additionally, we show that for every $\alpha > 2$, if triangle finding can be solved in n^α time, then non-emptiness of intersection for three DFA's over a unary alphabet can be solved in $n^{\frac{\alpha}{2}}$ time.[3] On the opposite direction, we show that if non-emptiness of intersection for three DFA's over a unary alphabet can be solved in $n^{\frac{\alpha}{2}}$ time, then triangle finding can be solved in n^α time.

2 Reductions

2.1 Reducing acceptance in NSPACE[n] to DFA-NEI

We show that acceptance for nondeterministic linear space bounded Turing machines is reducible to DFA-NEI. For any k, the reduction in Theorem 1 outputs k DFA's each with at most $O(m^2 \cdot n^2 \cdot 2^{\frac{n}{k}})$ states where m denotes the number of states in the Turing machine and n denotes the input string length.

2-tape Turing Machines: A 2-tape Turing machine with binary alphabet is a tuple $M = (Q, \{0,1\}, q_0, F, \delta)$ where Q is a set of states, $q_0 \in Q$ is an initial state, F is a final state, and

$$\delta : Q \times \{0,1\}^2 \to \mathcal{P}(Q \times \{-1,0,1\}^2 \times \{0,1\})$$

is a partial transition function that assigns to each triple $(q, b_1, b_2) \in Q \times \{0,1\}^2$, a set of tuples $\delta(q, b_1, b_2) \subseteq Q \times \{-1,0,1\}^2 \times \{0,1\}$. We say that a tuple $(q, d, d', w) \in Q \times \{-1,0,1\}^2 \times \{0,1\}$ is an instruction which sets the machine to state q, moves the input head from position p to position $p + d$, the work head from position p' to position $p' + d'$, and writes bit w at position $p' + d'$ in the work head. The transition function δ specifies that if the machine M is currently at state q, reading symbol b_1 at the input tape and symbol b_2 at the work tape, then the next instruction of the machine must be an element of the set $\delta(q, b_1, b_2)$.

Configurations: A *space-m configuration* for M at input $x \in \{0,1\}^*$ is a tuple

$$(q, h, l, y) \in Q \times [|x|] \times [m] \times \{0,1\}^m$$

where intuitively, $q \in Q$ is the current state of M, $h \in [|x|]$ is the position of M's input-tape head, $l \in [m]$ is the position of M's work-tape head, and $y \in \{0,1\}^m$ is the binary string corresponding the first m bits of the work tape of M.

[3] The authors would especially like the thank Joseph Swernofsky for advice and feedback that helped in obtaining this result as well as Ronald Fagin for some early feedback and encouragement.

Configuration Sequences: A *space-m configuration sequence* for M at input $x \in \{0,1\}^*$ is a sequence of the form

$$\begin{aligned} S \equiv (q_0, h_0, h'_0, y_0) &\xrightarrow{(q_1,d_1,d'_1,r_1,r'_1,w_1)} (q_1, h_1, h'_1, y_1) \\ &\xrightarrow{(q_2,d_2,d'_2,r_2,r'_2,w_2)} (q_2, h_2, h'_2, y_2) \\ &\dots \\ &\xrightarrow{(q_k,d_k,d'_k,r_k,r'_k,w_k)} (q_k, h_k, h'_k, y_k) \end{aligned}$$

satisfying the following conditions.

1. For each $i \in \{0,1,...,k\}$, (q_i, h_i, h'_i, y_i) is a space-m configuration for M at input x.
2. q_0 is the initial state of M, q_k is a final state of M, $y_0 = 0^m$, meaning that the work tape is initialized with zeros, and $h_0 = h'_0 = 1$, meaning that the input-tape head and work-tape head are in the first position of their respective tapes.
3. For each $i \in \{1,...,k\}$, $(q_i, d_i, d'_i, w_i) \in \delta(q_{i-1}, x[h_{i-1}], y_{i-1}[h'_{i-1}])$, meaning the state of the machine at time i, the directions taken by both heads at time i, and the symbol written at the work tape at time i are compatible with the transition function δ, and depend only on the state at time $i-1$ and on the symbols which are read at time $i-1$.
4. For each $i \in \{1,...k\}$, $h_i = h_{i-1} + d_i$, $h'_i = h'_{i-1} + d'_i$, $r_i = x_i[h_i]$, $r'_i = y_i[h'_i]$, and y_i is obtained from y_{i-1} by substituting w_i for the symbol $y_{i-1}[h'_{i-1}]$, and by leaving all other symbols untouched. Intuitively, this means that the configuration at time i is obtained from the configuration at time $i-1$ by the application of the transition (d_i, e_i, q_i, s_i).

We say that the sequence $I \equiv (q_1, d_1, d'_1, r_1, r'_1, w_1)(q_2, d_2, d'_2, r_2, r'_2, w_2)...(q_k, d_k, d'_k, r_k, r'_k, w_k)$ that induces a configuration sequence S as above is a *space-m instruction sequence* for M at input x. We say that I is accepting if $q_k \in F$.

▶ **Theorem 1.** *Given a nondeterministic m-state 2-tape Turing machine M with binary tape alphabet and an input string x of length n. If M uses at most n bits on the work tape, then for every k, we can efficiently compute k DFA's $\langle \mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k \rangle$ each with a binary alphabet and $O(m^2 \cdot n^2 \cdot 2^{\frac{n}{k}})$ states such that M accepts x if and only if $\bigcap_{i=1}^{k} \mathcal{L}(\mathcal{A}_i) \neq \emptyset$.*

Proof. The Turing machine M accepts x if and only if there exists an accepting space-n instruction sequence for M at x. We build k DFA's that read in a binary string and collectively determine whether the string encodes an accepting space-n instruction sequence for M at x.

Consider splitting the work tape of M into k equal sized blocks each consisting of $\frac{n}{k}$ work tape cells. A block-i space-n configuration for M at input x consists of the state, input tape head, work tape head, and the contents of the work tape from position $lbound_i := i \cdot \frac{n}{k} + 1$ to position $rbound_i := (i+1) \cdot \frac{n}{k}$. We construct k DFA's $\langle \mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_k \rangle$ where each DFA $\mathcal{A}_i$ keeps track of the current block-i space-n configuration for M at input x. The DFA's read in space-n instructions one at a time and transition accordingly where each instruction is encoded as a unique bit string of length $O(\log(m))$.

The start state of DFA $\mathcal{A}_i$ represents the block-i space-n configuration $(q_0, 1, 1, 0^{\frac{n}{k}})$ where q_0 is the start state of M. Further, a state representing a block-i space-n configuration $(q_j, h_j, h'_j, block_j)$ is accepting if q_j is an accepting state of M. Suppose that the DFA $\mathcal{A}_i$ is currently at a state representing a block-i space-n configuration $(q_j, h_j, h'_j, block_j)$ and reads

in a space-n instruction (q, d, d', r, r', w). The DFA $\mathcal{A}_i$ transitions to a state representing a block-i space-n configuration $(q_{j+1}, h_{j+1}, h'_{j+1}, block_{j+1})$ if:

1. $(q, d, d', w) \in \delta(q_j, r, r')$ and $q = q_{j+1}$
2. $h_{j+1} = h_j + d$ and $h'_{j+1} = h'_j + d'$
3. $1 \leq h_j, h'_j, h_{j+1}, h'_{j+1} \leq n$
4. $r = x[h_j]$
5. if $lbound_i \leq h'_j \leq rbound_i$, then $r' = block_j[h'_j - lbound_i]$ and $w = block_{j+1}[h'_j - lbound_i]$

Collectively the DFA's determine whether the input string encodes an accepting space-n instruction sequence for M at x. Therefore, the Turing machine M accepts x if and only if there exists an accepting space-n instruction sequence for M at x if and only if $\bigcap_{i=1}^{k} \mathcal{L}(\mathcal{A}_i) \neq \emptyset$. Further, the DFA's each have at most $O(m^2 \cdot n^2 \cdot 2^{\frac{n}{k}})$ states because there are $O(m)$ space-n instructions and $O(m \cdot n^2 \cdot 2^{\frac{n}{k}})$ block-i space-n configurations. ◀

▶ **Corollary 2.** *We obtain the following directly from the preceding theorem:*

1. *If we can solve* k-DFA-NEI *in time* $n^{o(k)}$, *then* NSPACE$[n] \subseteq$ DTIME$[2^{o(n)}]$.
2. *If there exists* $k \geq 2$ *and* $\varepsilon > 0$ *such that* k-DFA-NEI *in time* $O(n^{k-\varepsilon})$, *then* NSPACE$[n] \subseteq$ DTIME$[2^{\delta \cdot n}]$ *for some* $\delta < 1$.[4]

2.2 Cutwidth & a binary encoding for automata

We formally define finite automata, cutwidth for finite automata, and a binary encoding for finite automata. We use these notions in the following subsection to provide an improved unconditional space complexity lower bound for DFA-NEI.

Let Σ be a finite set of symbols. A non-deterministic finite automaton (NFA) over Σ is a tuple $\mathcal{A} = (\mathcal{Q}, \Sigma, \mathfrak{q}_0, \mathcal{F}, \Delta)$ where $\mathcal{Q}$ is a set of states, $\mathfrak{q}_0 \in \mathcal{Q}$ is an initial state, $\mathcal{F} \subseteq \mathcal{Q}$ is a set of final states, and $\Delta \subseteq \mathcal{Q} \times \Sigma \times \mathcal{Q}$ is a transition relation. We say that a word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if there exists a sequence of transitions

$$\mathfrak{q}_0 \xrightarrow{w[1]} \mathfrak{q}_1 \xrightarrow{w[2]} \ldots \xrightarrow{w[m]} \mathfrak{q}_m$$

such that $\mathfrak{q}_m \in \mathcal{F}$, and for each $j \in \{1, ..., m\}$, the transition $\mathfrak{q}_{j-1} \xrightarrow{w[j]} \mathfrak{q}_j$ belongs to Δ. We let $\mathcal{L}(\mathcal{A})$ be the set of all words accepted by $\mathcal{A}$. We say that $\mathcal{A}$ is a deterministic finite automaton (DFA) if for each pair $(\mathfrak{q}, a) \in \mathcal{Q} \times \Sigma$, there is at most one[5] $\mathfrak{q}' \in \mathcal{Q}$ such that $\mathfrak{q} \xrightarrow{a} \mathfrak{q}'$ is a transition of Δ.

Let $\mathcal{A} = (\mathcal{Q}, \Sigma, \mathfrak{q}_0, \mathcal{F}, \Delta)$ be a finite automaton. If X is a subset of $\mathcal{Q}$, then we let $Cut(X) = \{(q, a, q') \mid q \in X,\ q' \in \mathcal{Q} \backslash X, (q, a, q') \in \Delta\}$ be the set of transitions with one state in X and another state in $\mathcal{Q} \backslash X$. Let m denote the number of states in $\mathcal{Q}$. Now, consider a bijection $\alpha : [m] \to \mathcal{Q}$. The cutwidth of $\mathcal{A}$ with respect to α is defined as

$$cw(\mathcal{A}, \alpha) = \max_{j \in [m]} |Cut(\{\alpha(1), \alpha(2), ..., \alpha(j)\})|.$$

[4] See related results in [1, 33].

[5] Notice that we do not require for there to be a $\mathfrak{q}'$. In other words, unlike some common conventions, we do not require for there to be a transition for every alphabet character. We omit dead states.

The cutwidth of $\mathcal{A}$ is defined $cw(\mathcal{A}) = \min_\alpha cw(\mathcal{A}, \alpha)$ where α ranges over all bijections from $[m]$ to $\mathcal{Q}$. For readers familiar with the notion of cutwidth of graphs, we remark that the cutwidth of an automaton $\mathcal{A}$ is simply the cutwidth of the graph induced by the transition relation of $\mathcal{A}$.

We define a binary string encoding for DFA's. Given a DFA A with n states over an alphabet of size c along with an indexing[6] of A's states $q_1, q_2, ..., q_m$ and alphabet characters $a_1, a_2, ..., a_c$. The string encoding of A includes three substrings separated by deliminators representing the start state, list of final states, and list of transitions. The start state q_i is encoded by the index i in binary. The list of final states is encoded as a sequence of indexes for final states in binary separated by delimiters. The list of transitions is encoded as a sequence where each transition (q_i, a_j, q_k) for alphabet character a_j, source state q_i, and target state q_k is represented by a string consisting of j and $k - i$ in binary separated by delimiters. The string representations of transitions are then in a sequence ordered by source state and alphabet character. Delimiters are used to separate transitions and also to group transitions that have the same source state.

Further, we replace each alphabet character with a binary string to convert this string encoding into a binary string encoding. This leads to a constant factor blowing up to the encoded string lengths. For the remainder of this work, we assume that all input DFA's are encoded relative to this binary encoding.

▶ **Lemma 3.** *Using the preceding binary encoding, a DFA A with an alphabet of size c, m states, and an indexing of states $q_1, q_2, ..., q_m$ with cutwidth w is encoded as a string of length at most $O(\, min\{m \cdot w \cdot \log(c),\ c \cdot m \cdot (\log(m) + \log(c))\}\,)$.*

Proof. We show two upper bounds on the length of the binary encoding of A. First, the encoding has length at most $O(c \cdot m \cdot (\log(m) + \log(c)))$ because there are m states where each state has at most c outgoing transitions such that each transition is encoded using $O(\log(m) + \log(c))$ bits.

Second, we use the cutwidth to define a recurrence relation for an upper bound of the total string length. The recurrence relation is $R(1) = O(1)$ and

$$R(m) = 2 \cdot R(m/2) + O(w \cdot (\log(c) + \log(m))).$$

We obtain this recurrence by partitioning the set of states into two equal size sets. Because the cutwidth is w, there are at most w transitions between the sets which could have length up to $O(\log(c) + \log(m))$ each. Then, within each set, the transitions can be represented using at most $R(m/2)$ bits. By applying elementary manipulations to the recurrence and applying masters theorem, we get $R(m) = O(m \cdot w \cdot \log(c))$. ◀

▶ **Corollary 4.** *Using the preceding binary encoding, a DFA A with a binary alphabet, m states, and an indexing of states $q_1, q_2, ..., q_m$ with bounded cutwidth is encoded as a string of length at most $O(m)$.*

2.3 An Improved Space Lower Bound for DFA-NEI

The following theorem improves upon Theorem 7 from [31].

[6] Notice that the indexing of A's states is just a bijection from $[m]$ to $\mathcal{Q}$.

▶ **Theorem 5.** *Consider any fixed nondeterministic 2-tape Turing machine M with binary tape alphabet. Let an input string x of length n be given. If M uses at most n bits on the work tape, then we can efficiently compute $2 \cdot n + 1$ DFA's $\langle \mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_{2n+1} \rangle$ each with a binary alphabet, $O(\log(n))$ states, and bounded cutwidth such that M accepts x if and only if $\bigcap_{i=1}^{2n+1} \mathcal{L}(\mathcal{A}_i) \neq \emptyset$.*

Proof. In a space-n instruction sequence, each space-n instruction has the form (q, d, d', r, r', w). We consider an *extended space-n instruction sequence* where each *extended space-n instruction* has the form $(q, d, d', h, h', r, r', w)$ where $h, h' \in [n]$.

The Turing machine M accepts x if and only if there exists an accepting extended space-n instruction sequence for M at x. We build $2n + 1$ DFA's that read in a binary string and collectively determine whether the string encodes an accepting extended space-n instruction sequence for M at x.

We construct $2n + 1$ DFA's $\langle \mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_{2n+1} \rangle$ as follows. For each $i \in [n]$, the DFA $\mathcal{A}_i$ reads in a sequence of extended space-n instructions. On each instruction, if the instruction is satisfactory, then the DFA cycles back to the start state. The start state is the only accepting state. Suppose that the DFA $\mathcal{A}_i$ is currently at the start state and reads in an extended space-n instruction $(q, d, d', h, h', r, r', w)$. The DFA $\mathcal{A}_i$ follows a sequence of transitions back to the start state if the following are satisfied:

1. if $i = h_j$, then $h = h_{j+1}$, $h = h_j + d_j$, $1 \leq h_j, h_{j+1} \leq n$, and $x[h_j] = r$
2. if $i = h'_j$, then $h' = h'_{j+1}$, $h' = h'_j + d'_j$, and $1 \leq h'_j, h'_{j+1} \leq n$

Since $(q, d, d', h, h', r, r', w)$ can be encoded by a bit string of length $O(\log(n))$ and we need at most a constant number of branches from the start state back to itself in $\mathcal{A}_i$'s state diagram, there are at most $O(\log(n))$ states and the DFA has constant cutwidth.

For each $i \in [n]$, DFA $\mathcal{A}_{n+i}$ reads in a sequence of extended space-n instructions. On each instruction, if the instruction is satisfactory, then the DFA cycles back to either the 0-state or the 1-state. The start state is the 0-state and both the 0-state and the 1-state are accepting states. Suppose that the DFA $\mathcal{A}_{n+i}$ is currently at the b_j-state for $b_j \in \{0, 1\}$ and reads in an extended space-n instruction $(q, d, d', h, h', r, r', w)$. The DFA $\mathcal{A}_{n+i}$ follows a sequence of transitions back to b_{j+1}-state for $b_{j+1} \in \{0, 1\}$ if the following is satisfied:

1. if $i = h'_j$, then $r' = b_j$ and $w = b_{j+1}$.

Since $(q, d, d', h, h', r, r', w)$ can be encoded by a bit string of length $O(\log(n))$ and we need at most a constant number of branches from b_j-state to b_{j+1}-state for each pair $b_j, b_{j+1} \in \{0, 1\}$ in $\mathcal{A}_{n+i}$'s state diagram, there are at most $O(\log(n))$ states and the DFA has constant cutwidth.

The DFA $\mathcal{A}_{2n+1}$ has states that represent each (q_j, h_j, h'_j) which is a subsequence of a space-n configuration. The start state represents $(q_0, 1, 1)$ where q_0 is a start state of M. The accepting states represent (q_j, h_j, h'_j) where q_j is an accepting state of M. Suppose that the DFA $\mathcal{A}_{2n+1}$ is currently at a state representing (q_j, h_j, h'_j) and reads in an extended space-n instruction $(q, d, d', h, h', r, r', w)$. The DFA $\mathcal{A}_{2n+1}$ transitions to a state representing $(q_{j+1}, h_{j+1}, h'_{j+1})$ if $(q, d, d', w) \in \delta(q_j, r, r')$ and $q = q_{j+1}$. The DFA $\mathcal{A}_{2n+1}$ has $O(1)$ states and cutwidth because the machine M is fixed.

Collectively the DFA's determine whether the input string encodes an accepting extended space-n instruction sequence for M at x. Therefore, the Turing machine M accepts x if and only if there exists an accepting extending space-n instruction sequence for M at x if and only if $\bigcap_{i=1}^{2n+1} \mathcal{L}(\mathcal{A}_i) \neq \emptyset$. ◀

By combining the results of Corollary 5, Theorem 6, and the nondeterministic space hierarchy theorem [26], we obtain the following unconditional space complexity lower bound.

▶ **Corollary 6.** DFA-NEI *can not be solved in space* $o(n/\log(n))$ *where* n *denotes the length of the binary encoding of the input for* DFA-NEI.

3 Emptiness of Intersection and Conditional Lower Bounds

In this section we apply results obtained in Theorem 1 and Corollary 2 to show that even a slight improvement in running time of the classic algorithm for non-emptiness of intersection of finite automata would yield faster than state of the art algorithms for satisfiability of Boolean formulas and Boolean circuits. Further applying Williams' *algorithms vs lower bounds* framework, we are able to show that faster algorithms for non-emptiness of intersection of finite automata implies non-uniform circuit lower bounds.

3.1 Satisfiability for Boolean Formulas

In the satisfiability problem for Boolean formulas (SAT), we are given a Boolean formula. The goal is to determine if there exists an assignment that satisfies the formula. It is common to restrict the inputs for SAT to formulas in conjunctive normal form (CNF-SAT). Further, it is common to restrict the inputs for SAT to formulas in conjunctive normal form with clause width at most k (k-CNF-SAT) for some number k.

The *Exponential Time Hypothesis* (ETH) asserts that for some $\varepsilon > 0$, 3-CNF-SAT cannot be solved in time $(1+\varepsilon)^n$ [14]. The *strong exponential time hypothesis* (SETH) asserts that for every $\varepsilon > 0$, there is a large enough integer k such that k-CNF-SAT cannot be solved in time $(2-\varepsilon)^n$ [14, 15, 8]. ETH has been used to rule out the existence of subexponential algorithms for many decision problems [14], parameterized problems [9, 21], approximation problems [23], and counting problems [10]. On the other hand, SETH has been useful in establishing tight lower bounds for many problems in P such as Edit Distance [4], k-Dominating Set[24], non-emptiness of intersection for deterministic finite automata [32] and many other problems [3, 29].

▶ **Lemma 7.** *Satisfiability for* n*-input Boolean formulas of size* s *is solvable by a non-deterministic* 2*-tape Turing machine with binary alphabet using at most* $n + O(\log(s))$ *bits on the work tape.*[7]

Proof. The machine uses n tape cells to guess an assignment $x \in \{0,1\}^n$ to the input variables. Then, using an additional of $O(\log s)$ tape-cells, the machine evaluates[8] the Boolean formula from the input tape on the assignment from the work tape. ◀

By combining the reduction from Theorem 1 with the Turing machine from Lemma 7, we obtain the following theorem.

▶ **Theorem 8.** *If there exists* $k \geq 2$ *and* $\varepsilon > 0$ *such that* k-DFA-NEI *can be solved in time* $O(n^{k-\varepsilon})$*, then SAT is solvable in time* $\text{poly}(s) \cdot 2^{n(1-\delta)}$ *for some* $\delta > 0$.

[7] In addition, both QBF and satisfiability for n-input non-deterministic branching programs are solvable by nondeterministic 2-tape Turing machines with binary alphabet using at most $n + O(\log(s))$ bits on the work tape.

[8] This evaluation problem is referred to as the Boolean formula value problem (BFVP)[22, 6, 7].

It was shown in [33] that if there exists some $k \geq 2$ and $\varepsilon > 0$ such that k-DFA-NEI can be solved in in time $O(n^{k-\varepsilon})$, then SETH is false. The following corollary states that an improvement in the running time of the standard algorithm for intersection of a constant number of DFAs would indeed have much stronger consequences.

▶ **Corollary 9.** *If there exists $k \geq 2$ and $\varepsilon > 0$ such that k-DFA-NEI can be solved in time $O(n^{k-\varepsilon})$, then satisfiability for n-input fan-in-2 Boolean formulas of size $2^{o(n)}$ can be solved in time $2^{n(1-\delta)}$ for some $\delta > 0$.*

Note that while CNFs of bounded width are a very weak computational model, formulas of sub-linear depth and sub-exponential size can already simulate any circuit in the class NC. Therefore the consequence of Corollary 13 would contradict the NC-SETH hypothesis, a more robust version of SETH which states that satisfiability of circuits of polynomial size and polylogarithmic depth cannot be solved in time $2^{n(1-\delta)}$ for any $\delta > 0$ [2]. In the next subsection we show that the existence of an algorithm running in time $n^{o(k)}$ for k-DFA-NEI would imply faster satisfiability algorithms for even larger classes of circuits.

3.2 Satisfiability for Boolean Circuits

In the CIRCUIT VALUE problem (CV), we are given an n-input fan-in-2 Boolean circuit C and a string $x \in \{0,1\}^n$. The goal is to determine whether the circuit $C(x)$, obtained by initializing the input variables of C according to x, evaluates to 1. Let the size of C denote the number of gates of C. The next lemma, which is a classic result in complexity theory [5], states that the circuit value problem for circuits of depth d and size s can be solved in space $O(d) + O(\log s)$ on a 2-tape Turing machine.

▶ **Lemma 10** (Borodin [5])**.** *There is a deterministic 2-tape Turing machine M over the alphabet $\{0,1\}$, that takes as input a pair $\langle C, x\rangle$ where x is a string in $\{0,1\}^n$ and C is an n-input fan-in-2 Boolean circuit of depth d and size s, and determines, using at most $O(d) + O(\log s)$ work-tape cells, whether $C(x)$ evaluates to 1.*

In the satisfiability problem for Boolean circuits, we are given an n-input fan-in-2 Boolean circuit C. The goal is to determine whether there exists a string $x \in \{0,1\}^n$ such that $C(x)$ evaluates to 1. As a consequence of Lemma 10, we have that satisfiability for circuits of depth d can be decided by a nondeterministic 2-tape Turing machine using at most $n + O(d) + O(\log s)$ tape cells.

▶ **Lemma 11.** *There is a nondeterministic 2-tape Turing machine M over the alphabet $\{0,1\}$, that takes as input an n-input fan-in-2 Boolean circuit C of depth d and size s, and determines, using at most $n + O(d) + O(\log s)$ tape cells, whether C is satisfiable.*

Combining the reduction from Theorem 1 with the preceding Lemma, we obtain the following.

▶ **Theorem 12.** *If we can solve k-DFA-NEI in time $n^{o(k)}$, then we can solve satisfiability for fan-in-2 Boolean circuits of size s and depth d in time $\mathrm{poly}(s) \cdot 2^{o(d)}$.*

It was shown in [11, 33] that if k-DFA-NEI can be solved in time $n^{o(k)}$ then satisfiability of CNF formulas with n variables and polynomially-many clauses can be solved in time $2^{o(n)}$. In other words, they have shown that k-DFA-NEI in time $n^{o(k)}$ would falsify the exponential-time hypothesis (ETH). The next corollary strengthens substantially this consequence, by showing that such a faster algorithm for k-DFA-NEI would indeed imply that satisfiability of circuits of linear depth and sub-exponential size could be solved in sub-exponential time.

▶ **Corollary 13.** *If we can solve* k-DFA-NEI *in time* $n^{o(k)}$*, then we can solve satisfiability for fan-in-2 Boolean circuits of size* $s = 2^{o(n)}$ *and depth* $O(n)$ *in time* $2^{o(n)}$*.*

3.3 Circuit Lower Bounds

Recall the following Theorems from the literature on circuit lower bounds [35, 2].

▶ **Theorem 14** ([35, 2])**.** *Suppose that there is a satisfiability algorithm for bounded fan-in formulas of size* n^k *running in* $O(2^n/n^k)$ *time, for all constant* $k > 0$*. Then* NTIME$[2^{O(n)}]$ *is not contained in non-uniform* NC^1*.*

Combining Theorem 8 with Theorem 15, we obtain the following.

▶ **Theorem 15.** *If there exists* $k \geq 2$ *such that* k-DFA-NEI *in time* $O(n^k/\log^c n)$ *for every constant* $c > 0$*, then* NTIME$[2^{O(n)}]$ *does not have non-uniform* NC^1 *circuits.*

In [2], it is shown that faster algorithms for Edit Distance or LCS lead to circuit lower bounds. Within their arguments, they show something more general. They show that faster algorithms for satisfiability of formulas of sub-exponential size imply non-uniform circuit lower bounds. Using our reduction from formula-SAT to k-DFA-NEI provided in Theorem 8 (and Corollary 9), we obtain the following conditional lower bounds analog to Corollary 1 of [2].

▶ **Theorem 16.** *If there exists* $k \geq 2$ *and* $\varepsilon > 0$ *such that* k-DFA-NEI *in time* $O(n^{k-\varepsilon})$*, then the complexity class* E^{NP} *does not have*[9]*:*

1. *non-uniform* $2^{o(n)}$*-size Boolean formulas and*
2. *non-uniform* $o(n)$*-depth circuits of bounded fan-in.*

Furthermore, NTIME$[2^{O(n)}]$ *is not in non-uniform* NC*.*

Below, we show that an algorithm running in time $n^{o(k)}$ for k-DFA-NEI would imply non-uniform lower bounds for Boolean circuits of linear depth and sub-exponential size.

▶ **Theorem 17** ([35], Theorem 8 of [2])**.** *Let* $S(n)$ *be a time constructible and monotone non-decreasing function such that* $n \leq S(n) \leq 2^{o(n)}$*. Let* $\mathcal{C}$ *be a class of circuits. Suppose there is an SAT algorithm for* n*-input circuits which are arbitrary functions of three* $O(S(n))$*-size circuits from* $\mathcal{C}$*, that runs in time* $O(2^n/(n^{10} \cdot S(n)))$*. Then* E^{NP} *does not have* $S(n)$*-size circuits from* $\mathcal{C}$*.*

Note that if we take three circuits of linear depth and sub-exponential size and combine their outputs using an arbitrary 3-bit gate, then the resulting circuit still have linear depth and sub-exponential size. Therefore a satisfiability algorithm running in time $2^{o(n)}$ for fan-in-2 Boolean circuits of linear depth and sub-exponential size would imply that there are functions in E^{NP} which cannot be computed by such circuits. Note that this would be a remarkable consequence in complexity theory since to date it is not even known whether all functions in E^{NP} can be computed by non-uniform circuits of linear size. In view of our discussion, Theorem 18 below is a corollary of Theorem 17 and Corollary 13.

▶ **Theorem 18.** *If* k-DFA-NEI *can be solved in time* $n^{o(k)}$ *then* E^{NP} *does not have non-uniform circuits of depth* $O(n)$ *and size* $2^{o(n)}$*.*

[9] Recall that E^{NP} is the class of functions that can be computed by Turing machines that operate in time $2^{O(n)}$ with help of an NP oracle. Note that the strings that are passed to each call of the oracle may have size up to $2^{O(n)}$.

4 Hardness within Polynomial Time

Hardness within polynomial time has been a recent focus within fine-grained complexity theory. Problems such as triangle finding and 3SUM have made their place as canonical hard problems in polynomial time [3]. We connect the hardness of 2-DFA-NEI and 3-DFA-NEI to the hardness of triangle finding and 3SUM. This further reinforces the suggestion from [20] that 2-DFA-NEI is a hard problem in polynomial time.

4.1 Non-Emptiness of Intersection for Two DFA's

We have seen in Sections 2 and 3 that solving 2-DFA-NEI in time faster than $n^{2-\varepsilon}$ would imply that NSPACE$[n] \subseteq$ DTIME$[2^{\delta \cdot n}]$ for some $\delta < 1$, and would imply non-uniform circuit lower bounds that are much sharper than those that are currently known. We strengthen these results by providing a reduction from triangle finding to non-emptiness of intersection for two DFA's over a binary input alphabet.

▶ **Theorem 19.** *There exists a fine-grained reduction from triangle finding for a graph with n vertices and m edges to non-emptiness of intersection for two DFA's where the first DFA has $m \log(n)$ states and the second DFA has $n \log(n)$ states.*

As a result, we get that faster algorithms for non-emptiness of intersection lead to faster algorithms for triangle finding. For example, if we could solve the preceding non-emptiness of intersection problem in $n^{\frac{3}{4}} m^{\frac{3}{4}}$ time, then we could solve triangle finding in $n^{2.25}$ time which would be better than any currently known upper bound.

Since non-emptiness of intersection for two DFA's over a binary input alphabet is hard and we don't have any new approaches for solving it, we pursue a restricted version of the problem. In particular, we restrict the DFA's to a unary input alphabet. A DFA over a unary alphabet has a simple structure that some may describe as a pan[10] consisting of a segment[11] and a cycle. The segment is simply a sequence of states one following another in a linear fashion. A DFA over a unary alphabet can be thought of as a disjunction of numeric constraints where an input string is a number represented in unary that either satisfies some constraint (i.e. is accepted by the automaton) or does not satisfy any of the constraints (i.e. is rejected by the automaton).

Let n denote a natural number in unary that will act as a possible input for the DFA. Let l_s and l_c denote the segment and cycle lengths, respectively. The final states of the DFA represent the constraints. If the final state is along the segment at the ith position, then the constraint is $n = i - 1$. If the final state is within the cycle at the jth position, then the constraint is $n \geq l_s \ \wedge \ n - l_s \equiv i \ mod \ l_c$. Notice that all of the final states along the segment represent constraints that are only satisfied by fixed values while all of the final states within the cycle represent constraints on the remainder modulo the cycle length.

In general, the non-emptiness of intersection problem for DFA's over a binary alphabet is $PSPACE$-complete while the non-emptiness of intersection problem for DFA's over a unary alphabet is NP-complete[12] [27]. This suggests that there may exist subquadratic time algorithms for non-emptiness of intersection for two DFA's over a unary alphabet. In fact,

[10] This intuition was communicated to the authors from Michael Blondin.
[11] The segment may be empty.
[12] Recently, additional fine-grained hardness results for problems related to unary finite automata were shown in [11].

we show that the non-emptiness of intersection problem for two DFA's over a unary alphabet is solvable in near linear time.

▶ **Theorem 20.** *Non-Emptiness of Intersection for two DFA's over a unary alphabet is solvable in $O(n \log(n))$ time.*

4.2 Non-Emptiness of Intersection for Three DFA's

Further, we consider the non-emptiness of intersection problem for three DFA's. Following the previous results [33, 11], it is known that a subcubic time algorithm for non-emptiness of intersection for three DFA's would imply that the strong exponential time hypotheses is false. We strengthen these hardness results by providing a reduction from 3SUM to non-emptiness of intersection for three DFA's over a binary input alphabet.

▶ **Theorem 21.** *Let a natural number k be given. There exists a fine-grained reduction from 3SUM for a set of n numbers in the range $[-n^k, n^k]$ to non-emptiness of intersection for three DFA's over a binary alphabet where each DFA has at most $kn \log(n)$ states.*

Next, we consider the non-emptiness of intersection for three DFA's operating on a unary alphabet. Surprisingly, we were able to exactly characterize the complexity of triangle finding. In particular, we were able to show that the complexity of intersection non-emptiness for three DFA's over a unary alphabet is the square root of the complexity of triangle finding. In other words, we show a fine-grained equivalence that precisely characterizes the relationship between the exponents expressed in the complexities of non-emptiness of intersection for three DFA's over a unary alphabet and triangle finding.[13]

▶ **Theorem 22.** *Let a real number $\alpha > 2$ be given. If we can solve triangle finding in n^α time, then we can solve non-emptiness of intersection for three DFA's over a unary alphabet in $n^{\frac{\alpha}{2}}$ time.*

▶ **Theorem 23.** *Let a real number $\alpha > 2$ be given. If we can solve non-emptiness of intersection for three DFA's over a unary alphabet in $n^{\frac{\alpha}{2}}$ time, then we can solve triangle finding in n^α time.*

References

1. Deciding emptiness of intersection of regular languages in subquadratic time. `https://cstheory.stackexchange.com/questions/29142`. Accessed: 2018-02-13.
2. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: Or: a polylog shaved is a lower bound made. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing*, STOC '16, pages 375–388, New York, NY, USA, 2016. ACM. URL: `http://doi.acm.org/10.1145/2897518.2897653`, `doi:10.1145/2897518.2897653`.
3. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 434–443. IEEE, 2014.
4. Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing*, pages 51–58. ACM, 2015.

[13] The authors would especially like the thank Joseph Swernofsky for advice and feedback that helped in obtaining this result as well as Ronald Fagin for some early feedback and encouragement.

5 Allan Borodin. On relating time and space to size and depth. *SIAM journal on computing*, 6(4):733–744, 1977.

6 S. R. Buss. The boolean formula value problem is in alogtime. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 123–131, New York, NY, USA, 1987. ACM. URL: http://doi.acm.org/10.1145/28395.28409, doi:10.1145/28395.28409.

7 Samuel R. Buss. Algorithms for boolean formula evaluation and for tree contraction. In *Arithmetic, Proof Theory and Computational Complexity*, pages 96–115. Oxford University Press, 1993.

8 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *International Workshop on Parameterized and Exact Computation*, pages 75–85. Springer, 2009.

9 Jianer Chen, Benny Chor, Mike Fellows, Xiuzhen Huang, David Juedes, Iyad A Kanj, and Ge Xia. Tight lower bounds for certain parameterized np-hard problems. *Information and Computation*, 201(2):216–231, 2005.

10 Holger Dell, Thore Husfeldt, Dániel Marx, Nina Taslaman, and Martin Wahlén. Exponential time complexity of the permanent and the tutte polynomial. *ACM Transactions on Algorithms (TALG)*, 10(4):21, 2014.

11 Henning Fernau and Andreas Krebs. Problems on finite automata and the exponential time hypothesis. In Yo-Sub Han and Kai Salomaa, editors, *Implementation and Application of Automata: 21st International Conference, CIAA 2016, Seoul, South Korea, July 19-22, 2016, Proceedings*, pages 89–100, Cham, 2016. Springer International Publishing. URL: http://dx.doi.org/10.1007/978-3-319-40946-7_8, doi:10.1007/978-3-319-40946-7_8.

12 John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *J. ACM*, 24(2):332–337, April 1977. URL: http://doi.acm.org/10.1145/322003.322015, doi:10.1145/322003.322015.

13 Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complex.*, 14(2):257–299, June 1998. URL: http://dx.doi.org/10.1006/jcom.1998.0476, doi:10.1006/jcom.1998.0476.

14 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62:367–375, 2001.

15 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

16 George Karakostas, Richard J Lipton, and Anastasios Viglas. On the complexity of intersecting finite state automata and nl versus np. *Theoretical Computer Science*, 302(1):257–274, 2003.

17 Takumi Kasai and Shigeki Iwata. Gradually intractable problems and nondeterministic log-space lower bounds. *Mathematical systems theory*, 18(1):153–170, 1985. URL: http://dx.doi.org/10.1007/BF01699467, doi:10.1007/BF01699467.

18 Dexter Kozen. Lower bounds for natural proof systems. In *FOCS*, volume 77, pages 254–266, 1977.

19 Klaus-Jörn Lange and Peter Rossmanith. The emptiness problem for intersections of regular languages. In *MFCS*, volume 629 of *LNCS*, pages 346–354, 1992.

20 R. J. Lipton. On the intersection of finite automata. *Gödel's Lost Letter and P=NP*, August 2009. URL: http://rjlipton.wordpress.com/2009/08/17/on-the-intersection-of-finite-automata/.

21 Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Slightly superexponential parameterized problems. In *Proc. of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 760–776, 2011.

22 Nancy Lynch. Log space recognition and translation of parenthesis languages. *J. ACM*, 24(4):583–590, October 1977. URL: http://doi.acm.org/10.1145/322033.322037, doi:10.1145/322033.322037.

23 Dániel Marx. On the optimality of planar and geometric approximation schemes. In *Foundations of Computer Science, 2007. FOCS'07. 48th Annual IEEE Symposium on*, pages 338–348. IEEE, 2007.

24 Mihai Pătraşcu and Ryan Williams. On the possibility of faster sat algorithms. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1065–1075. SIAM, 2010.

25 Narad Rampersad and Jeffrey Shallit. Detecting patterns in finite regular and context-free languages. *Information Processing Letters*, 110(3):108–112, 2010.

26 Richard Edwin Stearns, Juris Hartmanis, and Philip M Lewis. Hierarchies of memory limited computations. In *Switching Circuit Theory and Logical Design, 1965. SWCT 1965. Sixth Annual Symposium on*, pages 179–190. IEEE, 1965.

27 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9, 1973.

28 H. Todd Wareham. The parameterized complexity of intersection and composition operations on sets of finite-state automata. In Shen Yu and Andrei Păun, editors, *Implementation and Application of Automata: 5th International Conference, CIAA 2000 London, Ontario, Canada, July 24–25, 2000 Revised Papers*, pages 302–310, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. URL: http://dx.doi.org/10.1007/3-540-44674-5_26, doi:10.1007/3-540-44674-5_26.

29 Joshua R. Wang and Richard Ryan Williams. Exact algorithms and strong exponential time hypothesis. In *Encyclopedia of Algorithms*, pages 657–661. 2016.

30 Michael Wehar. Intersection emptiness for finite automata. Honors thesis, Carnegie Mellon University, 2012.

31 Michael Wehar. Hardness results for intersection non-emptiness. In *International Colloquium on Automata, Languages, and Programming*, pages 354–362. Springer, 2014.

32 Michael Wehar. Intersection non-emptiness for tree shaped finite automata. *manuscript*, 2016.

33 Michael Wehar. *On the Complexity of Intersection Non-Emptiness Problems*. PhD thesis, University at Buffalo, 2016.

34 Ryan Williams. Finding paths of length k in $o^*(2^k)$ time. *Inf. Process. Lett.*, 109(6):315–318, 2009.

35 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

5 Proofs of Results from Section 4

Restatement of Theorem 19 . *There exists a fine-grained reduction from triangle finding for a graph with n vertices and m edges to non-emptiness of intersection for two DFA's where the first DFA has $m\log(n)$ states and the second DFA has $n\log(n)$ states.*

Proof. Let a graph G with n vertices and m edges be given. We construct DFA's D_1 and D_2 over a binary input alphabet such that D_1 has at most $m\log(n)$ states, D_2 has at most $n\log(n)$ states, and G contains a triangle if and only if D_1 and D_2 have a non-empty intersection.

We construct D_1 so that it reads in a sequence of four vertices encoded as bit strings. The DFA D_1 reads the first two vertices and checks if they actually form an edge in G. Since

there are at most m edges in G, there are only $2m$ possible paths each of length at most $2\log(n)$. If the first two vertices form an edge in G, then the DFA remembers the second vertex by transitioning to a state corresponding to this vertex. All of the $2m$ paths with this second vertex go to the same state. Then, the DFA reads the third vertex checking if it has an edge with the second vertex. Similarly, if such an edge does exist in G, then the DFA transitions to a state corresponding to the third vertex. And, so on checking that the third and fourth vertices form an edge in G.

We construct D_2 so that it reads the first vertex and remembers it. Then, the DFA reads the second and third vertices ignoring them. This requires a straight path of states of length $2\log(n)$. Finally, the DFA reads the fourth vertex checking that it is the same as the first vertex. ◀

Restatement of Theorem 20 . *Non-Emptiness of Intersection for two DFA's over a unary alphabet is solvable in $O(n\log(n))$ time.*

Proof. Let two DFA's D_1 and D_2 over a unary alphabet be given. Consider all of the final states of D_1 and D_2. If any of the final states fall on the segment of D_1 or D_2, we can quickly (in linear time) check to see if any of these states represent strings that are in the intersection. If none of them do, then we continue on.

We get the cycle length for D_1's cycle and D_2's cycle. We denote these cycle lengths by c_1 and c_2, respectively. Each final state along the cycle (offsetted by the handle) represents a remainder modulo the cycle length. Then, determining if there is a unary string in the intersection is equivalent to determining if there is a remainder r_1 computed from a final state of D_1 and a remainder r_2 computed from a final state of D_2 such that there exists x satisfying $x \equiv r_1 \ mod \ c_1$ and $x \equiv r_2 \ mod \ c_2$.

Next, we compute the greatest common divisor d of c_1 and c_2. We go through each final state of D_1 along the cycle and compute the remainder modulo d. We create a set of all possible remainders S_1. Similarly, we create a set S_2 of all possible remainders modulo d for D_2.

Finally, S_1 intersect S_2 is non-empty if and only if D_1's language and D_2's language have a non-empty intersection. As a result, we can apply a standard approach for checking set intersection. Using a RAM machine we can accomplish this in $n\log(n)$ time while it would take $n\log^2(n)$ time for a two tape Turing machine. The blow-up in time in the latter case comes from applying merge sort which makes $n\log(n)$ comparisons that each take $\log(n)$ time. ◀

Restatement of Theorem 21 . *Let a natural number k be given. There exists a fine-grained reduction from 3SUM for a set of n numbers in the range $[-n^k, n^k]$ to non-emptiness of intersection for three DFA's over a binary alphabet where each DFA has at most $kn\log(n)$ states.*

Proof. Let a natural number k be given. Let a set of n numbers in the range $[-n^k, n^k]$ be given. We construct DFA's D_1, D_2, and D_3 as follows. The DFA's will read in a binary string that encodes three numbers, but this encoding will not be in sequence. Rather, the binary encodings of the three numbers will be interleaved one bit at a time to make the input string. Also, the numbers are represented in binary such that the left most bit is the least significant bit.

The first DFA will verify that the first number (corresponding to the bit positions with remainder 0 mod 3) is a number in the set. While the second DFA will verify that the second number (corresponding to the bit positions with remainder 1 mod 3) is a number in the set

and the third DFA will verify that the third number (corresponding to the bit positions with remainder 2 mod 3) is a number in the set. Further, one of the DFA's is designated to also add the first two bits and check if it equals the third while carrying a remainder of 0 or 1 with it. If all three numbers are in the set and the first two numbers add up to the third number, then there exists a three sum. Otherwise, there does not. ◀

Restatement of Theorem 22 . *Let a real number $\alpha > 2$ be given. If we can solve triangle finding in n^α time, then we can solve non-emptiness of intersection for three DFA's over a unary alphabet in $n^{\frac{\alpha}{2}}$ time.*

Proof. Let DFA's over a unary alphabet D_1, D_2, and D_3 be given. We may ignore the DFA's segments if we first check to make sure there is no small number in their intersection. Next, we consider the segment lengths s_1, s_2, and s_3 along with cycle lengths c_1, c_2, and c_3 for DFA's D_1, D_2, and D_3, respectively. We compute the greatest common divisor d of c_1, c_2, and c_3. Let $c_1' := \frac{c_1}{d}$, $c_2' := \frac{c_2}{d}$, and $c_3' := \frac{c_3}{d}$.

For each $i \in [d-1] \cup \{0\}$, we construct a tripartite graph G_i. Let such a natural number i be given. The graph G_i consists of three groups of vertices of sizes $g_1 := gcd(c_1', c_2')$, $g_2 := gcd(c_2', c_3')$, and $g_3 := gcd(c_3', c_1')$, respectively. The vertices in the first group are label from 0 to $g_1 - 1$. The vertices in the second group are label from 0 to $g_2 - 1$. Similarly, the vertices in the third group are label from 0 to $g_3 - 1$.

Consider the edges between the first and second group of vertices of G_i. The vertices $x \in [g_1 - 1] \cup \{0\}$ and $y \in [g_2 - 1] \cup \{0\}$ from the first and second groups, respectively, are connected by an edge if there exists a final state along D_2's cycle at position l so that $l \equiv i \ mod \ d$, $l \equiv x \ mod \ g_1$, and $l \equiv y \ mod \ g_2$.

Similarly, edges are define between the second/third groups and third/first groups. Since $gcd(g_1, g_2, g_3) = 1$, we have that $g_1 \cdot g_2 \leq c_2'$, $g_2 \cdot g_3 \leq c_3'$, and $g_3 \cdot g_1 \leq c_1'$. As a result, there are at most c_2' edges between the first and second groups, at most c_3' edges between the second and third groups, and at most c_1' edges between the third and first groups.

We constructed graphs $\{G_i\}_{i \in [d-1] \cup \{0\}}$ so that there exists a string in the intersection of D_1, D_2, and D_3 if and only if there exists $i \in [d-1] \cup \{0\}$ such that G_i contains a triangle. Now, we can iteratively go through each of the d graphs searching for triangles. For a given graph G_i, we can apply an approach similar to fast rectangular matrix multiplication (as described in [13]).

We proceed as follows. We pick the smallest group of vertices from G_i. Without loss of generality, we choose the second group of vertices containing g_2 vertices. Since $g_2 = min(g_1, g_2, g_3)$ and $g_1 \cdot g_2 \leq c_2'$, we have that $g_2 \leq \sqrt{c_2'}$.

We reduce the triangle finding problem for G_i to solving triangle finding for $\frac{g_1 \cdot g_3}{g_2^2}$ subgraphs each with $O(g_2)$ vertices. We do this by splitting the g_1 vertices of the first group of G_i into $\frac{g_1}{g_2}$ subsets each with g_2 vertices and splitting the g_3 vertices of the third group of G_i into $\frac{g_3}{g_2}$ subsets each with g_2 vertices. There are $\frac{g_1 \cdot g_3}{g_2^2}$ combinations consisting of a subset of the first group and a subset of the third group. Each combination corresponds with a subgraph of G_i containing roughly $3 \cdot g_2$ vertices. Therefore, using the n^α time algorithm for triangle finding, we can solve triangle finding for all of these subgraphs in roughly $g_1 \cdot g_3 \cdot g_2^{\alpha-2}$ total time.

If n represents the total input size, then we have $g_1 \cdot g_3 \leq c_1' \leq \frac{n}{d}$ and $g_2 \leq \sqrt{c_2'} \leq \sqrt{\frac{n}{d}}$. Therefore, we can solve triangle finding for G_i in $(\frac{n}{d})^{\frac{\alpha}{2}}$ time. Since we need to carry out this procedure d times (once for each graph G_i), in total, the computation takes $d \cdot (\frac{n}{d})^{\frac{\alpha}{2}} \leq n^{\frac{\alpha}{2}}$ time. ◀

Restatement of Theorem 23 . *Let a real number $\alpha > 2$ be given. If we can solve non-emptiness of intersection for three DFA's over a unary alphabet in $n^{\frac{\alpha}{2}}$ time, then we can*

solve triangle finding in n^{α} time.

Proof. Let a graph G with n vertices be given. We first choose three prime numbers p_1, p_2, and p_3 greater than or equal to n. We correspond each vertex in G to a number in $[n]$. Then, we construct three DFA's over a unary alphabet so that they read in a number whose remainders mod $p_1 \cdot p_2$, $p_2 \cdot p_3$, and $p_3 \cdot p_1$ represent choices for the first, second, and third edges in a possible triangle, respectively.

The DFA D_1 will be a cycle of length $p_1 \cdot p_2$ that checks if the input number's remainder mod $p_1 \cdot p_2$ corresponds with an edge of G. The DFA D_2 will be a cycle of length $p_2 \cdot p_3$ that checks if the input number's remainder mod $p_2 \cdot p_3$ corresponds with an edge of G. And, similarly, the DFA D_3 will be a cycle of length $p_3 \cdot p_1$ that checks if the input number's remainder mod $p_3 \cdot p_1$ corresponds with an edge of G. Then, there exists a number satisfying all three DFA's if and only if there exists a triangle in G. ◀