Consensus String Problem for Multiple Regular Languages

Yo-Sub Han[1], Sang-Ki Ko[2], Timothy Ng[3], and Kai Salomaa[3]

[1] Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
emmous@yonsei.ac.kr
[2] Department of Computer Science, University of Liverpool
Liverpool L69 3BX, UK
sangkiko@liverpool.ac.uk
[3] School of Computing, Queen's University
Kingston, Ontario K7L 3N6, Canada
{ng,ksalomaa}@queensu.ca

Abstract. The consensus string (or center string, closest string) of a set S of strings is defined as a string which is within a radius r from all strings in S. It is well-known that the consensus string problem for a finite set of equal-length strings is NP-complete. We study the consensus string problem for multiple regular languages. We define the consensus string of languages $L_1, \ldots, L_k$ to be within distance at most r to some string in each of the languages $L_1, \ldots, L_k$. We also study the complexity of some parameterized variants of the consensus string problem. For a constant k, we give a polynomial time algorithm for the consensus string problem for k regular languages using additive weighted finite automata. We show that the consensus string problem for multiple regular languages becomes intractable when k is not fixed. We also examine the case when the length of the consensus string is given as part of input.

Keywords: consensus string problem, computational complexity, regular languages, edit-distance

1 Introduction

In bioinformatics, the multiple sequence alignment is a process of finding an optimal alignment from its multiple reads to find a correct biological sequence [10, 21]. See Fig. 1 for example. Moreover, it is a very important task to find the *consensus sequence* for detecting data commonalities from a set of strings in many practical applications such as coding theory [5, 7], data compression [8], and so forth.

There have been several definitions of a consensus string for a set of strings. Frances and Litman defined the consensus string based on the concept of the *radius* [7]. The radius of a string w with respect to a set S of strings is the smallest number r such that the distance between w and any string in S is bounded by r.

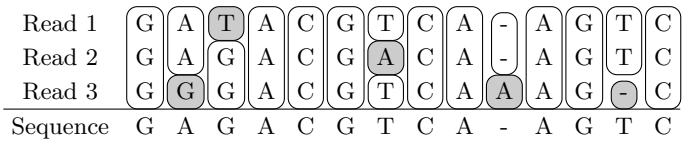

Fig. 1. The correct biological sequence can be estimated as the consensus sequence derived from the multiple sequence alignment from its reads

It is known that the consensus string problem based on radius is NP-complete even when the strings are given over the binary alphabet [7]. Sim and Park considered a different consensus string problem where they try to minimize the sum of distances between w and all strings in S, which is the *consensus error*. They showed that the consensus string problem based on the consensus error is NP-complete when the penalty matrix is a metric [25].

Amir et al. [1] examined the consensus string problem by considering both distance sum and radius, where the distance sum is the sum of distances from the strings in the given set to the consensus string and the radius is the largest distance from the set to the consensus string. They presented efficient polynomial time algorithms for the set of three strings. Amir et al. [2] studied the consensus string problem for other string metrics such as the swap metric and the reversal metric and showed that the problem is NP-hard for the swap metric and APX-hard for the reversal metric.

Since there is no polynomial-time solution for the consensus string problem unless P=NP, many researchers studied approximation algorithms and fixed-parameter tractability [10, 16]. Stojanovic et al. [26] proposed a linear-time algorithm when the radius is one. Gramm et al. [9, 10] designed the first fixed-parameter algorithm whose time complexity is $O(kl + kr^{r+1})$ where k is the number of strings, l is the length of strings, and r is the radius, which yields a linear-time algorithm for a constant r. For more details on fixed-parameter tractability of the consensus string problem, we refer the reader to [10, 16, 21].

We consider the consensus string problem for multiple regular languages, that is, to compute the consensus string which is within a given radius to the given regular languages. We use the Levenshtein distance (edit-distance) as a distance function for strings instead of the Hamming distance since the regular languages may be infinite and thus containing strings of unequal length.[4]

Given k regular languages given by NFAs or DFAs (nondeterministic or deterministic finite-state automata) and a radius r, we define the consensus string problem which decides the existence of a consensus string of radius r. We show that the problem is PSPACE-complete in general, and also when the radius r is fixed. We also establish that the problem becomes polynomial-time decidable when k is fixed using additive weighted finite automata. Lastly, we study a special case when the length l of the consensus string is given as input.

[4] The Hamming distance is only for strings of equal length

2 Preliminaries

In the following Σ is always a finite alphabet, Σ^* is the set of strings over Σ, and ε is the empty string. The length of a string w is $|w|$. When there is no danger of confusion, a singleton set $\{w\}$ is denoted simply as w. The set of non-negative integers is $\mathbb{N}_0$. For $n \in \mathbb{N}_0$, $\mathrm{bin}(n) \in \{0,1\}^*$ is the string giving the binary representation of n.

A *nondeterministic finite automaton* (NFA) is a tuple $A = (Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of states, Σ is an alphabet, δ is a multi-valued transition function $\delta : Q \times \Sigma \to 2^Q$, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of final states. We extend the transition function δ to a function $Q \times \Sigma^* \to 2^Q$ in the usual way. A string $w \in \Sigma^*$ is *accepted* by A if $\delta(q_0, w) \cap F \neq \emptyset$ and the language recognized by A consists of all strings accepted by A. The automaton A is a *deterministic finite automaton* (DFA) if, for all $q \in Q$ and $a \in \Sigma^*$, $\delta(q, a)$ either consists of one state or is undefined.

The reader can find more details on finite automata and regular languages in the text by Shallit [24] or the survey by Yu [27].

2.1 Distance measures and neighbourhoods

Intuitively, a distance measure on strings is a numerical description of how far apart two strings are. We view a distance on strings as a function from $\Sigma^* \times \Sigma^*$ to the nonnegative rationals that has value zero only for two identical strings, is symmetric, and satisfies the triangle inequality [6]. For our purposes it is sufficient to consider *integral distances* [20] where the values are integers. We define that a function $d : \Sigma^* \times \Sigma^* \to \mathbb{N}_0$ is a *distance* if it satisfies, for all $x, y, z \in \Sigma^*$,

(i) $d(x, y) = 0$ if and only if $x = y$,
(ii) $d(x, y) = d(y, x)$,
(iii) $d(x, z) \leq d(x, y) + d(y, z)$.

The *neighbourhood* of radius r of a language L is the set

$$E(L, d, r) = \{x \in \Sigma^* : (\exists y \in L)\ d(x, y) \leq r\}.$$

A distance d is said to be *finite* if the neighbourhood of any given radius of an individual string with respect to d is finite. A distance d is *additive* [3] if for every factorization $w = w_1 w_2$ and radius $r \geq 0$,

$$E(w, d, r) = \bigcup_{r_1 + r_2 = r} E(w_1, d, r_1) \cdot E(w_2, d, r_2).$$

Additive distances preserve regularity in the sense that the neighbourhood of a regular language is always regular [3].

The Levenshtein distance [15], a.k.a. the *edit-distance* d_e [22, 11, 13, 4] is an example of an additive distance. By the elementary edit operations on strings we mean an operation that replaces an alphabet symbol by another alphabet

symbol (substitution), a deletion of an alphabet symbol or an insertion of an alphabet symbol into a string. The edit-distance between strings s_1 and s_2 is the smallest number of elementary edit operations that transform s_1 into s_2. More generally, we can associate a non-negative cost to each elementary edit operation and then define the edit-distance between s_1 and s_2 as the smallest cost of all sequences of elementary edit operations that transform string s_1 into string s_2.

Unless otherwise mentioned, in the following we assume that elementary edit operations are all associated a unit cost. Our results would not change significantly if we use more general costs for the elementary edit operations.

We recall a result on the nondeterministic state complexity of edit-distance neighbourhoods. The result is originally due to Povarov [23] who states it for the Hamming distance neighbourhoods. However, exactly the same construction works for a general edit-distance where the costs of elementary edit operations are non-negative integers [20]. We add to the statement of the result a time bound needed for the construction.

Proposition 1 ([20, 23]). *Let A be an NFA with n states and $r \in \mathbb{N}$. The neighbourhood of $L(A)$ of radius r with respect to the edit-distance d_e can be recognized by an NFA B with $n \cdot (r+1)$ states. The NFA B can be constructed in time that depends polynomially on n and r.*

2.2 Additive weighted finite automata

An additive weighted finite automaton model has been used by Ng et al. [19] to recognize the neighbourhood of an NFA language. Since we need to recognize an intersection of neighbourhoods we extend the model to a multi-component weighted finite automaton where the states are k-tuples and transition weights are k-tuples of integers. The original definition allows real weights but integer weights will be sufficient for our purposes.

Definition 2. *An* additive k-component weighted finite automaton *(additive k-WFA) is a 6-tuple $A = (Q, \Sigma, \gamma, \omega, q_0, F)$ where $Q = P_1 \times \cdots \times P_k$, $k \in \mathbb{N}$, and each P_i is a finite set of states, Σ is an alphabet, $\gamma : Q \times (\Sigma \cup \{\varepsilon\}) \to 2^Q$ is the transition function, $\omega : Q \times (\Sigma \cup \{\varepsilon\}) \times Q \to \mathbb{N}_0^k$ is a partial weight function where $\omega(q_1, a, q_2)$ is defined if and only if $q_2 \in \gamma(q_1, a)$, $(a \in \Sigma \cup \{\varepsilon\})$, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is the set of accepting states.*

Strictly speaking, the transitions of γ are also determined by the domain of the partial function ω. By a transition of an additive k-WFA A on symbol $a \in \Sigma$ we mean a triple (q_1, a, q_2) such that $q_2 \in \gamma(q_1, a)$, $q_1, q_2 \in Q$. A computation path α of the additive k-WFA A along a string $w = a_1 a_2 \cdots a_m$, $a_i \in \Sigma$, $i = 1, \ldots, m$, from state s_1 to s_2 is a sequence of transitions that spells out the string w,

$$\alpha = (q_0, a_1, q_1)(q_1, a_2, q_2) \cdots (q_{m-1}, a_m, q_m), \tag{1}$$

where $s_1 = q_0$, $s_2 = q_m$, and $q_i \in \gamma(q_{i-1}, a_i)$, $1 \leq i \leq m$.

In the following, we use componentwise notation for addition and inequality for k-tuples in $\mathbb{N}_0^k$. Let $(x_1, \ldots, x_k), (y_1, \ldots, y_k) \in \mathbb{N}_0^k$. Then, $(x_1, \ldots, x_k) + (y_1, \ldots, y_k) = (x_1 + y_1, \ldots, x_k + y_k)$, and $(x_1, \ldots, x_k) \leq (y_1, \ldots, y_k)$ if and only if $x_i \leq y_i$, $i = 1, \ldots, k$.

The weight of the computation path α as in Equation (1) is

$$\omega(\alpha) = \sum_{i=1}^{m} \omega(q_{i-1}, a_i, q_i) \in \mathbb{N}_0^k.$$

We let $\Theta(s_1, w, s_2)$ denote the set of all computation paths along a string w from state s_1 to state s_2. The *language recognized by the additive k-WFA A within the weight bound $r \geq 0$* is the set of strings for which there exists a computation path that is accepted by A and each component of the weight is at most r. Formally, we define

$$L(A, r) = \{w \in \Sigma^* : (\exists f \in F)(\exists \alpha \in \Theta(q_0, w, f)) \ \ \omega(\alpha) \leq (r, \ldots, r)\}.$$

3 Consensus String for Multiple Regular Languages

We define that the consensus string of languages $L_1, \ldots, L_k$ is a string that is within a given edit-distance r from a string in each of the languages.

Definition 3 (Consensus string of multiple languages). *Let $L_i \subseteq \Sigma^*, 1 \leq i \leq k$ be k languages over an alphabet Σ. Then, a string w is a radius r consensus of the languages $L_1, L_2, \ldots, L_k$, if $\min\{d_e(w, w_i) \mid w_i \in L_i\} \leq r$ for all $1 \leq i \leq k$.*

The following lemma is an immediate consequence of the definition.

Lemma 4. *Let $r, k \in \mathbb{N}$. Languages $L_1, \ldots, L_k$ have a radius r consensus string if and only if $\bigcap_{i=1}^{k} E(L_i, d_e, r) \neq \emptyset$.*

We will consider the algorithmic problem of determining the existence of a consensus string of radius r for given k regular languages. We consider also uniform variants of the problem where the values k and/or r will be given as part of the input. The terminology for talking about different variants of the problem is defined next.

Definition 5 (consensus string problem for regular languages). *Let $k, r \in \mathbb{N}_0$ be fixed. The (k, r)-consensus string problem for NFAs (respectively, for given DFAs) is the problem of determining, for given NFAs (respectively, for DFAs) $A_1, \ldots, A_k$*

$$\textit{whether or not } L(A_1), \ldots, L(A_k) \textit{ have a radius } r \textit{ consensus string.} \quad (2)$$

For a fixed $r \in \mathbb{N}_0$, the $(*, r)$-consensus string problem for NFAs is the problem of determining whether Equation (2) holds for given $k \in \mathbb{N}$ and NFAs $A_1, \ldots, A_k$. For a fixed $k \in \mathbb{N}$, the $(k, *)$-consensus string problem for NFAs asks whether Equation (2) holds for given $r \in \mathbb{N}$ and NFAs $A_1, \ldots, A_k$, and the $(*, *)$-consensus problem for NFAs is the same problem where both k and r are part of the input.

The $(*, r)$- (respectively, $(k, *)$-, $(*, *)$-) consensus string problem for DFAs is defined analogously by restricting the input automata to be DFAs.

3.1 The $(*, r)$-consensus string problem for NFAs

Using Proposition 1 it is easy to see that the (k, r)-consensus string problem for NFAs can be solved in polynomial time. In Section 3.2, more generally, we show that the $(k, *)$-consensus string problem for NFAs has a polynomial time algorithm. On the other hand, the $(*, 0)$-consensus string problem for DFAs is the standard DFA intersection emptiness problem and is known to be PSPACE-complete [12, 14]. Below we show that, for any $r \in \mathbb{N}$, the $(*, r)$-consensus string problem is PSPACE-complete.

Lemma 6. *For $r \in \mathbb{N}$, the $(*, r)$-consensus string problem for DFAs is PSPACE-complete.*

Proof. For given k and DFAs $A_1, \ldots, A_k$, by Proposition 1, we can construct in polynomial space NFAs B_i recognizing the neighbourhood $E(L(A_i), d_e, r)$, $i = 1, \ldots, k$, since r is a constant. Then, the intersection emptiness of the NFAs B_i can be decided in polynomial space.

For the hardness result, we reduce the DFA intersection emptiness problem to the $(*, r)$-consensus string problem for DFAs. Let $C_1, \ldots, C_k$ be DFAs over an alphabet Σ. We construct DFAs $D_1, \ldots, D_k$ such that $D_1, \ldots, D_k$ have a radius r consensus string if and only if $\bigcap_{i=1}^{k} L(C_i) \neq \emptyset$.

Let $h : \Sigma^* \to \Sigma^*$ be a morphism defined by the condition $h(\sigma) = \sigma^{2r+1}$ for all $\sigma \in \Sigma$. Let D_i, $1 \leq i \leq k$, be a DFA for the language $h(L(C_i))$. If $\bigcap_{i=1}^{k} L(C_i) = \emptyset$, $L(D_1), \ldots, L(D_k)$ cannot have a radius r consensus because a block of $2r + 1$ symbols cannot be converted to a different block in r edit steps. Conversely, if $w \in \bigcap_{i=1}^{k} L(C_i)$, then $h(w)$ is a radius 0 consensus string for $L(D_1), \ldots, L(D_k)$. □

The first part of the proof works equally well for NFAs and we have

Corollary 7. *For $r \in \mathbb{N}$, the $(*, r)$-consensus string problem for NFAs is PSPACE-complete.*

The proof of Lemma 6 explicitly constructs NFAs for the neighbourhoods $E(L(A_i), d_e, r)$. In the case where the radius r is part of the input this construction cannot be completed in polynomial space and the proof does not imply that the $(*, *)$-consensus string problem is in PSPACE. In section 3.2, we give a different PSPACE algorithm for the $(*, *)$-consensus string problem.

It is known that the intersection emptiness for unary NFAs or DFAs is NP-complete [12] and this implies, as in the proof of Lemma 6, that for all $r \in \mathbb{N}$, the $(*, r)$-consensus string problem for unary DFAs is NP-hard.

Theorem 8. *For $r \in \mathbb{N}$, the $(*, r)$-consensus string problem for unary NFAs is NP-complete.*

Proof. We first show that the $(*, r)$-consensus string problem for unary NFAs over $\Sigma = \{0\}$ is in NP. Suppose that we have k unary NFAs from A_1 to A_k and $A_i, 1 \leq i \leq k$ has m_i states. If there is a radius r consensus string for unary

NFAs $A_1, \dots A_k$, then the intersection of the neighbourhood $\cap_{i=1}^{k} E(L(A_i), d_e, r)$ is not empty. Since we can construct the neighbourhood $E(L(A_i), d_e, r)$ of unary NFA A_i with $m_i + r$ states, we have an upper bound m on the number of states of $\cap_{i=1}^{k} E(L(A_i), d_e, r)$ as follows: $m = \prod_{i=1}^{k}(m_i + r)$.

This implies that if there is a radius r consensus string for unary NFAs $A_1, \dots A_k$, then there exists a consensus string whose length is bounded by m. Hence, we can nondeterministically guess a string $w = 0^n, n \leq m$ which is a radius r consensus string of unary NFAs. We can test whether or not $w \in E(L(A_i), d_e, r), 1 \leq i \leq k$ in time bounded by a polynomial in $|\text{bin}(n)|$ by successively squaring and multiplying matrices for the transition function. Since $|\text{bin}(n)| \leq |\text{bin}(m)| \in O(\log m)$, we can verify that the unary string w is a radius r consensus string of unary NFAs in time bounded by a polynomial in the size of NFAs.

For the hardness, we reduce the intersection emptiness of unary NFAs to the $(*, r)$-consensus string problem for unary NFAs as in the proof of Lemma 6. □

Finally, we note that since the emptiness problem for two context-free languages is undecidable, similarly as in Lemma 6, it follows that for all $k \geq 2$ and $r \geq 0$ the (k, r)-consensus problem for context-free languages is undecidable.

Corollary 9. *For all $k \geq 2$ and $r \geq 0$ the (k, r)-consensus problem for context-free languages is undecidable.*

3.2 The $(k, *)$-consensus string problem for NFAs

In this section we consider the consensus string problem where the number of NFAs is fixed. Proposition 1 yields "small" NFAs for radius r neighbourhoods of regular languages. However, the NFAs are still too big to yield a polynomial time algorithm if the size of the input is counted to be the length of the binary representation of r. On the other hand, if the distance r is given in unary, then the above approach yields polynomial time algorithm and we consider this case first.

By the $(k, *\text{unary})$-consensus string problem, we mean a variant of the $(k, *)$-consensus string problem for NFAs where the radius r is given in unary notation,

Lemma 10. *For $k \in \mathbb{N}$, the $(k, *\text{unary})$-consensus string problem for NFAs can be decided in polynomial time.*

Proof. Consider NFAs A_i with m_i states, $i = 1, \dots, k$. By Proposition 1, the neighbourhood $E(L(A_i), d_e, r)$ has an NFA B_i with $m_i \cdot (r + 1)$ states, $1 \leq i \leq k$, and B_i can be constructed in time that depends polynomially on m_i and r. An NFA C for $\bigcap_{i=1}^{k} L(B_i)$ can be obtained from the NFAs B_i, $1 \leq i \leq k$, using the standard cross-product construction and emptiness of $L(C)$ can be tested in polynomial time. The claim follows by Lemma 4. □

Note that if the radius r is given in binary the proof of Lemma 10 does not yield a polynomial time algorithm because the NFAs B_i are too large to

write down. Next we extend the polynomial time algorithm of the standard $(k, *)$-consensus string problem for NFAs. Given NFAs $A_1, \ldots, A_k$ and r, we again, roughly speaking, need to decide non-emptiness of $\bigcap_{i=1}^{k} E(L(A_i), d_e, r)$. However, instead of NFAs (with a number of states that depends exponentially on the size of the representation of r) we use a weighted finite automaton where the number of states is independent of r.

We obtain a construction for the weighted finite automaton by modifying the proof of the following lemma from [19].

Lemma 11 ([19]). *Let $N = (Q, \Sigma, \delta, q_0, F)$ be an NFA with n states, d an additive quasi-distance, and $R \geq 0$ is a constant. There exists an additive WFA A with n states such that for any $0 \leq r \leq R$,*

$$L(A, r) = E(L(N), d, r)$$

Furthermore, the WFA A can be constructed in time $O(n^3)$.

Note that Lemma 11 assumes that the radius is a constant and for this reason we have to be a little careful to check that the construction can be done in time that depends polynomially on the length of the binary representation of r. Furthermore, Lemma 11 deals only with one component WFAs (the case $k = 1$) but changing the construction for k components is straightforward. On the other hand, the construction for Lemma 11 allows the use of a general additive quasi-distance, and we are dealing with the simpler case of edit-distance where a neighbourhood of a string is always finite.

Lemma 12. *Let $k \in \mathbb{N}$ be fixed. Let $B_i = (S_i, \Sigma, \delta_i, s_{i,0}, F_i), i = 1, \ldots, k$ be an NFA with m_i states and $r \in \mathbb{N}$. There exists an additive k-WFA A with $\prod_{i=1}^{k} m_i$ states such that*

$$L(A, r) = \bigcap_{i=1}^{k} E(L(B_i), d_e, r) \tag{3}$$

Furthermore, the additive k-WFA A can be constructed in time that depends polynomially on the sizes of the NFAs B_i, $1 \leq i \leq k$, and $|\mathrm{bin}(r)|$.

Proof. We define an additive k-WFA $A = (Q, \Sigma, \gamma, \omega, q_0, F)$, where $Q = S_1 \times \cdots \times S_k$, $q_0 = (s_{1,0}, \ldots, s_{k,0})$, $F = F_1 \times \cdots \times F_k$ and the transitions of γ and the weight function are defined as follows.

The transition function γ is defined by setting, for $(s_1, \ldots, s_k) \in Q$, $b \in \Sigma \cup \{\varepsilon\}$,

$$\begin{aligned} &\gamma((s_1, \ldots, s_k), b) = \\ &\quad \{(s'_1, \ldots, s'_k) : (\exists x \in \Sigma^*)\ s'_i \in \delta_i(s_i, x), 1 \leq i \leq k, \text{ and } d_e(b, x) \leq r\}. \end{aligned} \tag{4}$$

That is, for each two states of A we add a transition on $b \in \Sigma \cup \{\varepsilon\}$ if there is a string $x \in \Sigma^*$ within edit-distance at most r from the symbol b that in the NFAs B_i transform the components of the first state to the components of the second state. The transition $(\mathbf{s}, b, \mathbf{s}')$, $\mathbf{s} = (s_1, \ldots, s_k)$, $\mathbf{s}' = (s'_1, \ldots, s'_k)$, in A has weight

$$\omega((\mathbf{s}, b, \mathbf{s}')) = \min_{x \in \Sigma^*} \{d_e(b, x) : s'_i \in \delta(s_i, x), i = 1, \ldots, k\}. \tag{5}$$

The correctness of the construction (i.e., that equation (3) holds) is verified as in [19] and we only need to check that the construction, for non-constant r, can be done in time that is polynomial in the m_i's and $|\text{bin}(r)|$ (whereas in [19] the radius was treated as a constant).

For $s_i, s_i' \in S_i$ and $b \in \Sigma \cup \{\varepsilon\}$, to check whether there exists a string x such that $s_i' \in \delta_i(s_i, x)$ and $d_e(x, b) \leq r$ we determine whether the shortest path in B_i from s_i to s_i' has length at most r, or length at most $r+1$ if $b \in \Sigma$ and the path contains an occurrence of b. This can be done in time that depends polynomially on m_i and $|\text{bin}(r)|$. Then, the transitions of γ (4) can be computed by repeating this procedure $(\Pi_{i=1}^{k} m_i)^2$ times.

For each two tuples $\mathbf{s} = (s_1, \ldots, s_k)$ and $\mathbf{s}' = (s_1', \ldots, s_k')$ that will have a γ-transition on $b \in \Sigma \cup \{\varepsilon\}$, the process finds the string x which transforms components of $\mathbf{s}$ into components of $\mathbf{s}'$ in the NFAs B_i and minimizes the distance $d_e(b, x)$, and this allows us to compute also the weights in (5). Note that although the length of x may be superpolynomial in $|\text{bin}(r)|$, the length is upper bounded by the values m_i which are the sizes of the input NFAs. □

We need one more technical lemma.

Lemma 13. *Let $k \in \mathbb{N}$ be a constant. Given an additive k-WFA $A = (Q, \Sigma, \gamma, \omega, q_0, F)$ with n states and $r \in \mathbb{N}$, we can decide in time that is polynomial in n and in $|\text{bin}(r)|$ whether or not $L(A, r) = \emptyset$.*

Proof. We use a polynomial time graph reachability algorithm to check whether a final state of A is reachable from the initial state, and the computation keeps track of the smallest cumulative weight of the path traversed up to that point. Recall that each state of A is a k-tuple of states.

In stage one, the algorithm marks the states of A reachable from $q_0 = (q_{0,1}, \ldots, q_{0,k})$ in a single transition. Each of these states $q = (q_1, \ldots, q_k)$ is marked by a k-tuple $(w_1, \ldots, w_k)$, where w_i is the smallest weight of a transition that reaches q_i from $q_{0,i}$. The marking process is then repeated $n-1$ times as follows.

Suppose states $p_1, \ldots, p_r$ are marked at the ith stage ($i \leq n-2$) where the cumulative weight of p_i is w_i. In stage $i+1$ the algorithm marks states $p_1', \ldots, p_s'$ reachable from a state among $p_1, \ldots, p_r$ with a single transition. Each state $p_i' = (p_{i,1}', \ldots, p_{i,k}')$ will have a weight tuple $w_i' = (w_{i,1}', \ldots, w_{i,k}')$ where for $1 \leq \ell \leq k$, we have

$$w_{i,\ell} = \min(\{w_{j,\ell} + \omega(p_{j,\ell}, b, p_{i,\ell}') \mid 1 \leq j \leq r, b \in \Sigma \cup \{\varepsilon\}\} \cup \{r+1\}).$$

Note that weight $r+1$ is used as an error value indicating that the computation path has failed.

If at some point a final state is marked with a weight where each component is at most r, the algorithm answers "yes". The algorithm is correct because $L(A, r) \neq \emptyset$ if and only if A has a computation of weight at most r leading to a final state on a string of length at most $n-1$. The number of steps of the marking algorithm is upper bounded by $O(kn^2)$ (when the size of the alphabet

Σ is viewed as a constant). The individual weight additions can be done in time linear in $|\mathrm{bin}(r)|$. □

Now we present a polynomial time algorithm for the $(k, *)$-consensus string problem.

Theorem 14. *For $k \in \mathbb{N}_0$, the $(k, *)$-consensus string problem for NFAs can be solved in polynomial time.*

As a corollary we extend the result of Lemma 6 for the $(*, *)$-consensus string problem.

Corollary 15. *The $(*, *)$-consensus string problem for NFAs (or DFAs) is PSPACE-complete.*

3.3 Finding a consensus string of given length

Interestingly, if we are given the length $l \in \mathbb{N}_0$ of a consensus string in unary, then the computational complexity of the problem becomes NP-complete. We use a reduction from the classical consensus string problem which is already proven to be NP-complete. First, we give the definition of the classical consensus string problem. Note that the distance function d_H denotes the Hamming distance.

Definition 16. *Let $S \subseteq \Sigma^l$ be a finite set of strings of length l. Then, a string w is a radius r consensus of S, if $d_H(w, s) \leq r$ for all $s \in S$.*

Then, given $\ell, r \in \mathbb{N}$ and a set $S \subseteq \Sigma^l$, it is NP-complete to decide whether or not there exists a radius r consensus of S [7]. The problem remains NP-complete even when $|\Sigma| = 2$.

However, because we are considering the edit-distance for our variant of the consensus string problem, first we need to reduce the Hamming distance to the edit-distance. Manthey and Reischuk [17] have shown that it is possible to reduce the Hamming distance to the edit-distance when only binary strings are considered. Here we first show that we can reduce the Hamming distance to the edit-distance in a much simpler way by introducing one more character.

Lemma 17. *Let w, w' be a strings of length l over Σ and $h : \Sigma^* \to (\Sigma \cup \{\$\})^*, \$ \notin \Sigma$ be a morphism defined by $h(\sigma) = \$^l\sigma$ for all $\sigma \in \Sigma$. Then, $d_H(w, w') = d_e(h(w), h(w'))$.*

The next lemma shows that the consensus string problem for multiple NFAs is NP-complete when the length ℓ of the consensus string is given in unary. The lemma is proved by a reduction from the classical consensus string problem and by relying on Lemma 17.

Lemma 18. *The consensus string problem for NFAs (or DFAs) is NP-complete if the length l of consensus string is given in unary.*

We also mention that the problem is still NP-complete if context-free languages are considered instead of regular languages since the edit-distance between a string and a context-free language described by a context-free grammar (CFG) can be computed in polynomial time [18].

Corollary 19. *The consensus string problem for context-free languages is NP-complete if the length l of consensus string is given in unary.*

However, the problem becomes PSPACE-complete if the length ℓ is given in binary.

Theorem 20. *The consensus string problem for NFAs (or DFAs) is PSPACE-complete if the length l of consensus string is given in binary.*

We also establish that the *fixed-length consensus problem*, where the length l of the consensus string is fixed, can be decided in polynomial time.

Corollary 21. *The fixed-length consensus string problem for NFAs (or DFAs) can be decided in $O(n \log n)$ time, where n is the size of NFAs (or DFAs).*

As a final note, we state that the fixed-length consensus string problem for context-free languages given by CFGs can be also decided in polynomial time.

Corollary 22. *The fixed-length consensus string problem for CFGs can be decided in $O(n \log n)$ time, where n is the size of CFGs.*

References

1. A. Amir, G. M. Landau, J. C. Na, H. Park, K. Park, and J. S. Sim. Efficient algorithms for consensus string problems minimizing both distance sum and radius. *Theoretical Computer Science*, 412(39):5239–5246, 2011.
2. A. Amir, H. Paryenty, and L. Roditty. On the hardness of the consensus string problem. *Information Processing Letters*, 113(10–11):371–374, 2013.
3. C. S. Calude, K. Salomaa, and S. Yu. Additive distances and quasi-distances between words. *Journal of Universal Computer Science*, 8(2):141–152, 2002.
4. C. Choffrut and G. Pighizzini. Distances between languages and reflexivity of relations. *Theoretical Compututer Science*, 286(1):117–138, 2002.
5. G. D. Cohen, I. S. Honkala, S. N. Litsyn, and P. Solé. Long packing and covering codes. *IEEE Transactions on Information Theory*, 43(5):1617–1619, 2006.
6. M. M. Deza and E. Deza. *Encyclopedia of Distances.* Springer Berlin Heidelberg, 2009.
7. M. Frances and A. Litman. On covering problems of codes. *Theory of Computing Systems*, 30(2):113–119, 1997.
8. R. L. Graham and N. J. A. Sloane. On the covering radius of codes. *IEEE Transactions on Information Theory*, 31(3):385–401, 2006.
9. J. Gramm, R. Niedermeier, and P. Rossmanith. Exact solutions for closest string and related problems. In *Proceedings of the 12th International Symposium on Algorithms and Computation*, pages 441–453. 2001.

10. J. Gramm, R. Niedermeier, and P. Rossmanith. Fixed-parameter algorithms for closest string and related problems. *Algorithmica*, 37:25–42, 2003.
11. Y.-S. Han, S.-K. Ko, and K. Salomaa. The edit-distance between a regular language and a context-free language. *International Journal of Foundations of Computer Science*, 24(7):1067–1082, 2013.
12. M. Holzer and M. Kutrib. Descriptional and computational complexity of finite automata—a survey. *Information and Computation*, 209(3):456–470, 2011.
13. S. Konstantinidis. Computing the edit distance of a regular language. *Information and Computation*, 205(9):1307–1316, 2007.
14. D. Kozen. Lower bounds for natural proof systems. In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 254–266, 1977.
15. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.
16. B. Ma and X. Sun. More efficient algorithms for closest string and substring problems. *SIAM Journal on Computing*, 39(4):1432–1443, 2010.
17. B. Manthey and R. Reischuk. The intractability of computing the Hamming distance. *Theoretical Computer Science*, 337(1–3):331–346, 2005.
18. G. Myers. Approximately matching context-free languages. *Information Processing Letters*, 54:85–92, 1995.
19. T. Ng, D. Rappaport, and K. Salomaa. Quasi-distances and weighted finite automata. In *Proceedings of the 17th International Workshop on Descriptional Complexity of Formal Systems*, pages 209–219, 2015.
20. T. Ng, D. Rappaport, and K. Salomaa. State complexity of neighbourhoods and approximate pattern matching. In *Proceedings of the 19th International Conference on Developments in Language Theory*, pages 389–400, 2015.
21. L. Palopoli, Z.-Z. Chen, B. Ma, and L. Wang. A three-string approach to the closest string problem. *Journal of Computer and System Sciences*, 78(1):164–178, 2012.
22. G. Pighizzini. How hard is computing the edit distance? *Information and Computation*, 165(1):1–13, 2001.
23. G. Povarov. Descriptive complexity of the Hamming neighborhood of a regular language. In *Proceedings of the 1st International Conference on Language and Automata Theory and Applications*, pages 509–520, 2007.
24. J. Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, New York, NY, USA, 1st edition, 2008.
25. J. S. Sim and K. Park. The consensus string problem for a metric is NP-complete. *Journal of Discrete Algorithms*, 1(1):111–117, 2003.
26. N. Stojanovic, P. Berman, D. Gumucio, R. Hardison, and W. Miller. A linear-time algorithm for the 1-mismatch problem. In *Proceedings of the 5th International Workshop on Algorithms and Data Structures*, pages 126–135. 1997.
27. S. Yu. Handbook of formal languages, vol. 1. chapter Regular Languages, pages 41–110. 1997.