Synchronization and Diversity of Solutions

Emmanuel Arrighi[1], Henning Fernau[2], Mateus de Oliveira Oliveira[1,3], and Petra Wolf[1,2]

[1]University of Bergen, Bergen, Norway
[2]University of Trier, Trier, Germany
[3]University of Stockholm, Stockholm, Sweden

Abstract

A central computational problem in the realm of automata theory is the problem of determining whether a finite automaton A has a synchronizing word. This problem has found applications in a variety of subfields of artificial intelligence, including planning, robotics, and multi-agent systems. In this work, we study this problem within the framework of diversity of solutions, an up-and-coming trend in the field of artificial intelligence where the goal is to compute a set of solutions that are sufficiently distinct from one another.

We define a notion of diversity of solutions that is suitable for contexts were solutions are strings that may have distinct lengths. Using our notion of diversity, we show that for each fixed $r \in \mathbb{N}$, each fixed finite automaton A, and each finite automaton B given at the input, the problem of determining the existence of a diverse set $\{w_1, w_2, \ldots, w_r\} \subseteq L(B)$ of words that are synchronizing for A can be solved in polynomial time. Finally, we generalize this result to the realm of conformant planning, where the goal is to devise plans that achieve a goal irrespectively of initial conditions and of non-determinism that may occur during their execution.

1 Introduction

A word w is said to be synchronizing for a deterministic finite automaton (DFA) A if there is some state q of A such that any state q' is sent to q by w. This concept has found numerous applications across several subfields of computer science and artificial intelligence, such as circuit testing [46, 54, 75], multi-agent systems [16, 17, 71], robotics [67], game theory [63], among others.

The most central problem in the field of synchronization is the one to determine whether a given DFA has a synchronizing word. Note that this problem can be decided in polynomial time [79]. Nevertheless, in several applications, one is interested in finding a synchronizing word satisfying certain additional constraints [30, 88, 83]. Here, the complexity landscape of the problem changes drastically: even the problem of determining the existence of a synchronizing word satisfying certain additional regularity constraints is NP-hard. For instance, it is NP-hard to determine whether a given DFA A has a synchronizing word that belongs to the regular language ab^*a [30], or whose length is bounded by a given integer [73, 28].

Diversity of solutions is a trend that has been calling substantial attention of the artificial intelligence community during the past years [44, 7, 70, 6, 31, 51, 5]. Here, the goal is to find not a single solution to a given combinatorial problem, but rather a *small* set of solutions that are sufficiently diverse from each other. One of the motivations for this framework is that it can be applied in situations where certain side constraints are difficult, or even impossible to formalize. In this case, the user will have the opportunity to select a solution that she deems to be best

for her application at hand. Intuitively, the diversity requirement tells us that the solutions that are given as an output are a good representative of the space of solutions. See [24] for further discussions of the idea of selecting a diverse set of solutions versus the idea of selecting a so-called representative set of solutions. On the other hand, small sets of solutions make sense because one does not want to overwhelm the user with an excessive number of solutions.

While for many combinatorial problems, notions of solution diversity based on the Hamming distance between pairs of solutions are sufficient, in the context of synchronization, these notions are not so appropriate. First, distinct solutions may have distinct length, and it is not clear how positions should be aligned in order to define an appropriate notion of Hamming distance. Additionally, even strings of the same size that are very similar to each other may have very large Hamming distance. For instance, the Hamming distance between the string $w = abab...ab = (ab)^n$ and the string $w' = baba...ba = (ba)^n$ is $2n$, while w can be transformed into w' with two modifications: first delete the first symbol of w and then append the symbol a to the resulting string.

We circumvent the issue described above by basing our diversity measure in the notion of edit distance between strings [86, 57]. This is a well studied metric for strings that has many nice properties and applications in a wide variety of fields [14, 59, 18].

Another issue is that sets of solutions in which any two of them are far apart from each other may still not capture solution diversity in the context of synchronization. The problem is that if w is a synchronizing word, then any superword of w is also synchronizing. Therefore, any sequence of superwords $w_1, w_2, \ldots, w_k$ of w of substantially distinct lengths would have large diversity if only edit distance were taken into consideration. To circumvent this issue, we require that each word in the set of solutions is subsequence-minimal with respect to the synchronization requirement. The subsequence minimality requirement combined with edit distance not only guarantees that solutions in any given subset are genuinely distinct, but also provides a way of tackling diverse synchronization problems using the machinery of finite automata theory. On the one hand, Higman's lemma [47] (see Lemma 8), a classical tool in automata theory, implies that the set of subsequence-minimal synchronizing words in the language of an automaton is always finite. On the other hand, the computation of the edit distance between two words is a process that can be simulated using finite automata. More specifically, it is possible to construct finite automata accepting a suitable encoding of pairs of words that are far apart from each other.

Note that subsequence-minimal synchronization problems involving a single DFA A are already hard. First, subsequence-minimal synchronizing words for a DFA A may have exponential length on the number of states of A (Proposition 24). Second, determining if a given word w is subsequence-minimal among all synchronizing words in the language of a DFA A is coNP-hard (Theorem 22). Third, determining if a DFA A has two distinct subsequence-minimal synchronizing words is NP-hard (Theorem 28). Finally, the problem of enumerating the set of subsequence-minimal synchronizing words is #P-hard (Theorem 29).

In order to cope with the inherent intractability of synchronization problems, we leverage on the framework of parameterized complexity theory [25]. In particular, we show that for each fixed value of r, interesting computational problems requiring a diverse set with r subsequence-minimal synchronizing words can be solved in time that is fixed parameter tractable with respect to the size of the synchronizing automaton A. Previously, algorithms with an FPT dependence in $|A|$ were unknown even for $r = 2$! Using our approach we also show that given a DFA A with state set Q over an alphabet Σ, and a word $w \in \Sigma^*$ one can determine in time $O(f(|\Sigma|, |Q|) \cdot |w|)$, for some function f, if some subsequence-minimal synchronizing word for A is a subsequence of w, and we can construct such a subsequence in case the answer is affirmative (Theorem 17). As mentioned above, the unparameterized version of this problems is already

coNP-hard. Our main result (Theorem 18) states that given numbers $r, k \in \mathbb{N}$, a DFA A, and an possibly nondeterministic finite automaton B over an alphabet Σ, the problem of computing a subset $\{w_1, \ldots, w_r\} \subseteq L(B)$ of subsequence-minimal synchronizing words for A, with pairwise edit distance of at least k, can be solved in time $O(f_A(r,k) \cdot |B|^r \log(|B|))$ for some suitable function f depending only on A, r and k. Intuitively, the automaton A is a specification of a system which we want to synchronize (or reset), and B is a specification of the set of words that are allowed to be used as synchronizing sequences. As stated in the beginning of this section, the unparameterized version of this problem is NP-hard even if we are interested in finding a single solution and the language of the automaton B is as simple as ab^*a. As a consequence of our main result, given a word $w \in \Sigma^*$, the problem of determining whether there exist r subsequence-minimal synchronizing words for A that are subsequences of w and that are at least k apart from each other can be solved in time $O(f_A(r,k) \cdot |w|^r \log(|w|))$ (Corollary 19).

It turns out that our notion of diversity of solutions is general enough to be applied in other contexts where solutions are strings whose sizes may have vary. In particular, we generalize our framework to the realm of conformant planning (Theorem 33), where the goal is to design plans that achieve goals irrespectively of initial conditions and of nondeterminism that may occur during the execution of these plans [4, 12, 20, 68]. In Section 9, we describe quite a number of further applications of our approach within artificial intelligence. This ranges from the design of autonomous production lines over the interactionless synchronization of robots and agents in general to questions in game theory. We also shortly describe a possible application in the area of molecular computing.

Related Work. Diversity of solutions is a concept that has found applications in several subfields of artificial intelligence, such as information search and retrieval [38, 1], mixed integer programming [36, 21, 69], binary integer linear programming [40, 82], constraint programming [44, 45], SAT solving [66], recommender systems [2], routing problems [76], answer set programming [27], decision support systems [58, 42], genetic algorithms [34, 87], planning [7], computational social choice [5], and in many other fields. Recently, in the context of combinatorial optimization, there has been an increasing interest of analysing the notion of diversity of solutions from the perspective of parameterized complexity theory [6, 5, 31]. Nevertheless, the computational problems and parameterizations addressed in these works are substantially distinct from the ones studied here. In the vast majority of these contexts diversity is formalized using some notion of Hamming distance between solutions represented as a binary vector.

Finally, it is worth noting that the framework of diversity of solutions differs in spirit from the framework of knowledge compilation, where the goal is to succinct representations of a large space of solutions for a posteriori processing, [22, 29, 60], and from the framework of enumeration, where the purpose is to count or list a large number of solutions of a given combinatorial problem, usually all minimal or maximal solutions with respect to a certain partial order on the space of solutions [32, 39, 52, 53, 81].

2 Preliminaries

Let $\mathbb{N}$ denote the set of non-negative integers, while $\mathbb{N}_{>0}$ is the set of positive integers. For an integer $k \in \mathbb{N}$, we denote by $[k]$ the set $\{1, 2, \ldots, k\}$. Hence, $[0] = \emptyset$.

For $n \in \mathbb{N}, \Sigma^n$ denotes the set of all words of length n. We denote $\Sigma^+ = \bigcup_{n \in \mathbb{N}_{>0}} \Sigma^n$ and $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$, where ε denotes the *empty word* which is the unique word of length zero. We also use the notation $\Sigma^{<k} = \bigcup_{n=0}^{k-1} \Sigma^n$. Hence, Σ^* is the groundset of the free monoid generated by Σ; its binary operation is known as *concatenation*. Usually, concatenation is denoted by juxtaposition, but sometimes we explicitly write the concatenation operation as $\cdot$. Given a

word $w \in \Sigma^*$, we let $|w|$ denote the *length* of w. For each $i \in 1, \dots, |w|$, we let $w[i]$ denote the ith symbol of w. For $i, j \in 1, \dots, |w|$, we let $w[i..j]$ denote the *infix* $w[i]w[i+1] \dots w[j]$ of w.

A *deterministic finite automaton* (DFA) A is a tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $\delta \colon Q \times \Sigma \to Q$ is a total function called the *transition function* of A, $q_0 \in Q$ is the *start state*, and $F \subseteq Q$ is the set of *final states*. For convenience, in the following, we will sometimes omit start and final states of DFAs. When speaking about the complexity of algorithms that work with a DFA A, we sometimes use $|A|$ to denote the number of bits needed to specify A; for convenience, let $|A| = |Q|\,|\Sigma| \log(|Q|)$ (for DFAs). Often, $|Q|$ is sufficient as a size estimate. We generalize δ to words by setting $\delta(q, \varepsilon) = q$ and $\delta(q, w) = \delta(\delta(q, w[1]), w[2..|w|])$. We further generalize δ to sets of states $S \subseteq Q$ and to sets of input letters $\Gamma \subseteq \Sigma$ as $\delta(S, \Gamma) = \{\delta(s, \gamma) \colon s \in S, \gamma \in \Gamma\}$, or to words w as $\delta(S, w) = \{\delta(s, w) \colon s \in S\}$.

We assume basic knowledge of automata theory on side of the reader. In particular, the subset and the product construction for finite automata should be known. Also, we make use of nondeterministic finite automata (or NFA for short). Recall that with NFAs, the transition function is replaced by a transition relation. More precisely, a *nondeterministic finite automaton* (NFA) A is a tuple $A = (Q, \Sigma, \delta, Q_0, F)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $\delta \subseteq Q \times \Sigma \to Q$ is the *transition relation* of A, $Q_0 \subseteq Q$ is the set of *start state*, and $F \subseteq Q$ is the set of *final states*. We generalize δ to words by setting $(q, \varepsilon, q) \in \delta$ and $(q, w, q') \in \delta$ iff there is a state p such that $(q, w[1], p) \in \delta$ and $(p, w[2..|w|], q') \in \delta$. Alternatively, we can view δ as a function, mapping $2^Q \times \Sigma$ to 2^Q, by setting $\delta(S, w) = \{q \colon \exists s \in S((s, w, q) \in \delta)\}$. This also explains the well-known powerset automaton construction, as now $2^A = (2^Q, \Sigma, \delta, Q_0, F')$ is a DFA equivalent to A, setting $F' = \{G \subseteq Q \colon F \cap G \neq \emptyset\}$.

We will discuss several partial orderings on Σ^* in this this paper. See [11] for a review of these and other concepts. In particular, x is a *prefix* of y, written $x \sqsubseteq y$, if $y \in x\Sigma^*$, x is a *suffix* of y if $y \in \Sigma^* x$ and x is an *infix* of y, written $x \preceq y$, if $y \in \Sigma^* x \Sigma^*$. We say that x is a *subsequence* of y, written $x \leq y$,[1] if there exists a sequence of indices $i_1 < i_2 < \ldots < i_{|x|}$ such that for each $j \in [|x|]$, $y_{i_j} = x_j$. If we add the word *proper*, we exclude the possibility that $x = y$ in each of the previous definitions. In our notations, we then write $\sqsubset$, $\prec$, or $<$, respectively. A non-empty word x can be split into the prefix $x[1]$ of length 1 and the suffix of length $|x| - 1$ that we call (reminiscent of Prolog) the *tail* of x, denoting it by $\mathrm{tail}(x)$. Hence, $x = x[1]\mathrm{tail}(x)$.

Let Σ be some alphabet not containing the symbol $\Box$ that we will now use as a blank symbol. For $k \in \mathbb{N}_{>0}$, define $(\Sigma \cup \{\Box\})^{\times k} = (\Sigma \cup \{\Box\})^k \setminus \{(\Box, \dots, \Box)\}$ as a new alphabet with $(|\Sigma| + 1)^k - 1$ many symbols that we also call *compound characters*, consisting of k letters. Of particular importance will be the case $k = 2$. For this, we now define a specific construction, called *convolution*, that takes two words $u, v \in \Sigma^*$ and constructs a unique word $w = u \otimes v$ over the alphabet $(\Sigma \cup \{\Box\})^{\times 2}$, in a recursive fashion as follows.

$$u \otimes v = \begin{cases} \varepsilon, & \text{if } u = v = \varepsilon \\ (u[1], \Box) \cdot (\mathrm{tail}(u) \otimes \varepsilon), & \text{if } u \neq \varepsilon, v = \varepsilon \\ (\Box, v[1]) \cdot (\varepsilon \otimes \mathrm{tail}(v)), & \text{if } u = \varepsilon, v \neq \varepsilon \\ (u[1], v[1]) \cdot (\mathrm{tail}(u) \otimes \mathrm{tail}(v)), & \text{if } u \neq \varepsilon, v \neq \varepsilon \end{cases}$$

For instance, the convolution of word $u = ababa$ with $v = abb$ is the word

$$u \otimes v = (a, a)(b, b)(a, b)(b, \Box)(a, \Box).$$

This operation can be generalized to the convolution of $k > 2$ words in a straightforward manner.

[1] In the literature, the naming of these relations is not unique. Hence, what we call an infix is also known as a factor or a subword, while what we call a subsequence is also known as subword or scattered subword. We are using these notions also in order to avoid confusion.

Lemma 1. *The language of all words over the alphabet* $(\Sigma \cup \{\Box\})^{\times k}$ *that can be obtained as the convolution of* k *words over* Σ *can be accepted by a DFA with* 2^k *many states.*

Proof. We give a rather intuitive explanation of the DFA A only in the following. The state set is $\{0, 1\}^k\}$, with $(1, \dots, 1)$ being a trap state of A. A state $(b_1, \dots, b_k)$ is understood as follows. If $b_i = 0$, then no blank symbol was seen reading letters from the ith component of compound characters. Hence, $(0, \dots, 0)$ is the initial state and, as $(\Box, \dots, \Box) \notin (\Sigma \cup \{\Box\})^{\times k}$, $(1, \dots, 1)$ is a rejecting trap state. If $b_i = 0$ and a blank symbol is seen in the ith component of the current compound character (to be read from the input), then the DFA moves to a state with $b_i = 1$, and this is true for all $i \in [k]$. If $b_i = 0$ and a non-blank symbol is seen in the ith component of the current compound character (to be read from the input), then the DFA moves to a state with $b_i = 0$, and this is true for all $i \in [k]$. If $b_i = 1$ and a blank symbol is seen in the ith component of the current compound character (to be read from the input), then the DFA moves to a state with $b_i = 1$, and this is true for all $i \in [k]$. If $b_i = 1$ and a non-blank symbol is seen in the ith component of the current compound character (to be read from the input), then the DFA moves to the trap state $(1, \dots, 1)$. All states but the trap state $(0, \dots, 0)$ are accepting. □

The previous construction can be integrated in the classical product automaton construction to give the following result.

Corollary 2. *Let* $A = (Q, \Sigma, \delta, q_0, F)$ *be a DFA. Let* A^k *be a DFA accepting the set of all strings of the form* $u_1 \otimes u_2 \otimes \cdots \otimes u_k$ *where for each* $l \in [k]$, $u_l \in L(A)$*. Then,* A^k *needs to have no more than* $(2|Q|)^k$ *many states. A similar statement holds for NFAs.*

The synchronization problem for deterministic finite automata (DFA) asks for a given DFA $A = (Q, \Sigma, \delta)$ whether there exists a word $w \in \Sigma^*$ (called *synchronizing word*) that maps each state of A to the same state, i.e., for which $|\delta(Q, w)| = 1$ holds. This problem can be solved in polynomial time [75, 84], but it becomes NP-complete if an upper bound on the length of the sought synchronizing word is added to the input [73, 28]. Notice that the classical complexity picture changes drastically if one considers synchronization problems for partial DFA (i.e., the transition function may be partial), or for (variations of) NFA, or also if one ask to synchronize a given set of states, to mention only few variants, all of which turn out to be PSPACE-complete; see [61, 62, 74, 75].

3 Diversity in Synchronization

In some applications of synchronizing words, it is relevant to construct a synchronizing word that satisfies additional constraints. Nevertheless, such constraints may either not be completely well-defined or may yield a much harder synchronization problem. Examples of this phenomenon, where computing a synchronizing word satisfying simple constraints become PSPACE-complete can be found in [30, 48, 88]. Therefore, instead of specifying such side constraints formally, one alternative is to compute several synchronizing words for a given automaton from which the designer can pick the most suitable one. To have a good variety of choices, the solutions should be quite *diverse*. In this section we formulate a notion of solution diversity that suits the context of synchronization.

We will base our notion of diversity on the notion of edit distance between strings, as it provides a suitable way of measuring dissimilarity between strings with possibly distinct lengths. Let $u = u[1]u[2] \dots u[n]$ be a string in Σ. We define the following elementary edit operations on u; see [57, 86].

1. Insertion: for some $i \in \{0, \ldots, n\}$, insert a letter $a \in \Sigma$ after position i, obtaining in this way the string $u[1..i]\, a\, u[i+1..n]$.

2. Deletion: for some $i \in \{1, \ldots, n\}$, delete the entry $u[i]$ from u, obtaining in this way the string $u[1..i-1]\, u[i+1..n]$.

3. Replacement:[2] for some $i \in \{1, \ldots, n\}$, replace $u[i]$ with some letter $a \in \Sigma$ distinct from $u[i]$, obtaining in this way the string $u[1..i-1]\, a\, u[i+1..n]$.

We say that a string $u \in \Sigma^*$ can be edited to a string $w \in \Sigma^*$ in s steps if there is some sequence $u_0, u_1, \ldots, u_s$ of strings in Σ^* such that for each $j \in [r]$, u_j is obtained from u_{j-1} by the application of one of the three elementary edit operations above and $u = u_0$, $w = u_s$.

Definition 3 (Edit Distance)**.** *The edit distance between u and w, denoted by $\Delta(u, w)$, is defined as the minimum s such that u can be edited to w in s steps.*

Intuitively, the edit distance between u and w is the minimum number of insertions, deletions and replacements necessary to transform u into w. By classical results, this distance (and related ones) can be computed efficiently; see [86, 85].
Nevertheless, there is still a problem that needs to be overcome, stemming from the next fact.

Fact 4. *Let $A = (Q, \Sigma, \delta)$ be a DFA. Let $w \in \Sigma^*$ be a synchronizing word for A. Then, each word in $\Sigma^* w \Sigma^*$ synchronizes A [79, 80].*

Fact 4 implies that if we only take edit distance into consideration, then one can obtain a diverse set $\{w_1, ..., w_r\}$ by simply computing a single synchronizing word for A and then by appending to it sufficiently large suffixes. Although well-defined mathematically, this solution does not capture the intuitive notion of diversity in an appropriate way. In order to circumvent this issue, we could instead require that the words in the set are minimal under the infix ordering. The problem is that the set of infix-minimal synchronizing words of an automaton may still be infinite, because the infix ordering is not a well-ordering, and therefore, it becomes a challenging task to even define what good representatives for the space of solutions are.

Remark 5. *In general, for a given DFA A, the set of all infix-minimal synchronizing words for A can be infinite, i.e., if there is an infix-minimal synchronizing word w such that for some i with $0 < i < |w|$, some letter σ is a permutation on $\delta(Q, w[1..i])$, then all words in $w[1..i]\sigma^* w[i+1..|w|]$ are infix-minimal synchronizing words for A.*

We circumvent the issue discussed above by adopting the notion of subsequence minimality. We say that a word w is a *subsequence-minimal synchronizing word* for a DFA A if w is synchronizing for A and no proper subsequence w' of w is synchronizing for A. On the one hand, one can ensure that the set of subsequence-minimal synchronizing words for an automaton is finite (see Section 4). On the other hand, subsequence minimality imposes a higher level of dissimilarity between two words in a prospective subset of solutions, increasing in this way the representativeness of the set.

Definition 6 (Synchronization Diversity)**.** *Let $A = (Q, \Sigma, \delta)$ be a DFA. We say that a set of words $\{w_1, \ldots, w_r\}$ has synchronization diversity k if the following two conditions are satisfied.*

1. *For each $i, j \in \{1, ..., r\}$ with $i \neq j$, $\Delta(w_i, w_j) \geq k$.*

2. *For each $i \in \{1, ..., r\}$, w_i is a subsequence-minimal synchronizing word for A.*

[2] This operation is also sometimes called *change* or *substitution* in the literature. In the case of binary alphabets, it was introduced as *reversal* in [57], but this wording is quite ambiguous in the literature on Formal Languages and hence avoided here.

4 Subsequence Minimality

The next well-known lemma, whose proof can be found in [41, Corollary 18], states that, given an NFA A, one can construct a finite automaton Subseq(A) accepting precisely the subsequences of words in $L(A)$. As shown in [41, Lemma 19], this bound cannot be improved in general.

Lemma 7 (Subsequence Automaton)**.** *Let $A = (Q, \Sigma, \delta)$ be an NFA. One can construct in time $O(|A|)$ an NFA* Subseq(A) *with $|Q|$ states with $L(\text{Subseq}(A)) = \{u : \exists w \in L(A),\ u \leq w\}$. If A has a trap state,* Subseq(A) *has $|Q| - 1$ states.*

A classic result in partial order theory, known as Higman's Lemma, states that for each finite set Σ, the set Σ^* of finite words over Σ is well-ordered under the subsequence order [47]. An interesting consequence of this lemma is the fact that for each language $L \subseteq \Sigma^*$, the set of subsequence-minimal words in L is finite. For a more language-theoretic treatment, refer to [43].

Lemma 8 (Higman's Lemma)**.** *Let Σ be an alphabet and $L \subseteq \Sigma^*$. There is a unique finite set $S \subseteq L$ satisfying the following properties.*

1. *For each word $w \in L$, some word $u \in S$ is a subsequence of w.*
2. *Each word $u \in S$ is subsequence-minimal for L.*

It is worth noting that Lemma 8 holds with respect to any language $L \subseteq \Sigma^*$, irrespectively of whether L is regular or not. Nevertheless, if L is indeed regular and we are given an automaton A accepting L, then we can construct a DFA MinSubseq(A) accepting the set of subsequence-minimal words for L.

Lemma 9 (Subsequence-Minimal Words [11])**.** *Let $A = (Q, \Sigma, \delta)$ be a DFA. One can construct in time $O(2^{|Q|} \cdot |A|)$ a DFA* MinSubseq(A) *with at most $|Q| \cdot 2^{|Q|}$ many states accepting the language*

$$L(\text{MinSubseq}(A)) = \\ \{u : u \in L(\text{Subseq}(A)) \text{ and } \forall w \in L(\text{Subseq}(A)), w \not< u\}.$$

Interestingly, the exponential blow-up in the number of states of the input automaton A is unavoidable, even if one allows MinSubset(A) to be an NFA. Here, we refer to Example 3.18, Facts 3.19 and 3.20 in [11].

The next lemma states that given an automaton A, one can construct a DFA Sync(A) accepting precisely the words in Σ^* that are synchronizing for A. It is shown by using the classical subset construction, now on a DFA A; see [75, 84].

Lemma 10 (Synchronizing Words)**.** *Let $A = (Q, \Sigma, \delta)$ be a DFA. One can construct in time $O(2^{|Q|} \cdot |A|)$ a DFA* Sync(A) *with at most $2^{|Q|}$ states such that*

$$L(\text{Sync}(A)) = \{u : u \text{ is synchronizing for } A\}.$$

It was unknown until very recently [49] if this exponential blow-up is necessary, but it is indeed unavoidable. By combining Lemma 10 with Lemma 9, we have the following corollary stating that, given a DFA A, one can construct a DFA SyncMinSubseq(A) of state complexity at most double-exponential in the state complexity of A accepting the set of subsequence-minimal synchronizing words for A. It seems to be an interesting open question if this double-exponential blow-up is necessary.

Corollary 11. *Let* $A = (Q, \Sigma, \delta)$ *be a DFA. One can construct in time* $O(2^{2^{|Q|}} \cdot 2^{|A|})$ *a DFA* SyncMinSubseq(A) *with at most* $2^{2^{|Q|}+|Q|}$ *many states accepting the language*

$$L(\text{SyncMinSubseq}(A)) = \\ \{u \;:\; u \text{ is a subsequence-minimal synchronizing word for } A\}.$$

Proof. Namely, given the DFA $A = (Q, \Sigma, \delta)$, we first construct an automaton Sync(A) accepting all synchronizing words of A according to Lemma 10 in time $O(2^{|Q|} \cdot |A|)$. Sync$(A) = (Q', \Sigma, \delta')$ has at most $2^{|Q|}$ many states. By Lemma 9, from Sync(A) we can construct SyncMinSubseq$(A) = (Q'', \Sigma, \delta'')$ in time $O(2^{|Q'|} \cdot |\text{Sync}(A)|) = O(2^{2^{|Q|}} \cdot |\text{Sync}(A)|)$ with at most $|Q'| \cdot 2^{|Q'|} = 2^{|Q|} \cdot 2^{2^{|Q|}}$ many states. □

5 Edit Distance vs Diversity of Solutions

Wagner and Fischer [86] gave a dynamic-programming algorithm that efficiently computes the edit distance between two given strings. Further details can be found in [85]. This and similar questions are an active area of study until today, as exemplified by [13]. On an intuitive level—ignoring technicalities—the following lemma can be interpreted as an automata-theoretic counterpart of such computations.

Lemma 12. *Let* Σ *be an alphabet and* $k \in \mathbb{N}_{>0}$*. There is an NFA* Edit$^<(\Sigma, k)$ *with* $2^{O(k \log |\Sigma|)}$ *many states accepting the following language.*

$$L(\text{Edit}^<(\Sigma, k)) = \left\{ u \otimes w \in \left((\Sigma \cup \{\Box\})^{\times 2}\right)^* \;:\; \Delta(u, w) < k \right\}. \tag{1}$$

Proof. Let u and w be strings in Σ^+. If $\Delta(u, w) < k$, then it should be clear that $||u| - |w|| \leq k - 1$, since otherwise, we would need to insert and delete a total of at least k symbols to make the strings have the same length. In order to define an automaton that accepts precisely those strings of the form $u \otimes w$ for which $\Delta(u, w) < k$, we define an NFA that simulates the behavior of a two-head automaton, one reading the string u (the first string) and the other reading the string w (the second string). We keep track of the edit operations on u that are necessary to turn u into w. In other words, we keep track of the changes on u and just check against w.

At each step, if the first head is at position m and the second head is at position m'. If $m' > m$, then the automaton keeps in memory the string $x = u[m..m']$ comprising of the symbols of the first string from position m to position m'. On the other hand, if $m' < m$, then the automaton keeps track of the string $x = w[m..m']$ comprising the symbols of the second string from positions m to position m'. If $m = m'$, we have $x = \varepsilon$. In any case, we have that $|x|$ is always smaller than k. We proceed with a formal definition of our automaton, where we store not only x in the states, but also a bit b telling in which of the two cases we are, as well as a counter keeping track of the number of edit operations.

We let Edit$^<(\Sigma, k) = (Q, (\Sigma \cup \{\Box\})^{\times 2}, \delta, F, Q_0)$ be the nondeterministic finite automaton with set of states $Q = \Sigma^{<k} \times \{0, 1\} \times \{0, ..., k-1\}$, set of initial states $Q_0 = \{(\varepsilon, 0, 0)\}$, and set of final states $F = \{\varepsilon\} \times \{0, 1\} \times \{0, \ldots, k-1\}$. We will describe next the elements of the transition relation δ. Altogether, we define four types of transitions leaving each state $(x, b, j) \in Q$, depending on whether $b = 0$ or $b = 1$.

1. *No Edits.* If $b = 0$ and $x \neq \varepsilon$, then we add (for each $a, a' \in \Sigma$ with $x[1] = a'$) a transition of the form $[(x, b, j), (a, a'), (\text{tail}(x)a, b, j)]$. Intuitively, if the symbol being read by the first head (i.e., the symbol $x[1]$ of the memory vector) is equal to the symbol being read by the second head (i.e., the symbol a'), then no edit operation is being performed, and

hence, the counter remains unchanged. Additionally, the new memory vector is obtained by dropping its first symbol and then appending the symbol a to it. A similar reasoning can be applied if $b = 1$, with exception of the fact that in this case, the memory vector records positions of the second string, and therefore we add a transition of the form $[(x, b, j), (a, a'), (\text{tail}(x)a', b, j)]$ for each $a, a' \in \Sigma$ with $x[1] = a$. In the special case when $x = \varepsilon$, irrespectively of b, we have transitions of the form $[(\varepsilon, b, j), (a, a), (\varepsilon, b, j)]$ for each $a \in \Sigma$.

2. *Replacement.* If $b = 0$, $j < k-1$ and $x \neq \varepsilon$, then we add (for each $a, a' \in \Sigma$ with $x[1] \neq a'$) a transition of the form $[(x, b, j), (a, a'), (\text{tail}(x)a, b, j+1)]$. Intuitively, if the symbol being read by the first head (i.e., the symbol $x[1]$ of the memory vector) is different from the symbol being read by the second head (i.e., the symbol a'), then a replacement operation is being performed (transforming $x[1]$ into a'), and hence, the counter is increased by 1. As in the previous case, the new memory vector is obtained by dropping its first symbol and then appending the symbol a to it. A similar reasoning can be applied if $b = 1$. As in the previous item, in this case we add a transition of the form $[(x, b, j), (a, a'), (\text{tail}(x)a', b, j)]$ for each $a, a' \in \Sigma$ with $a \neq x[1]$. In the special case when $x = \varepsilon$, irrespectively of b, we have transitions of the form $[(\varepsilon, b, j), (a, a'), (\varepsilon, b, j + 1)]$ for each $a, a' \in \Sigma$ with $a \neq a'$.

3. *Deletion.* If $b = 0$, $j < k-1$ and $x \neq \varepsilon$, then we consider the compound input letter (a, a') for each $a, a' \in \Sigma$ and check if there is some $i \leq |x|$ with $x[i] = a'$. In this case, assuming $j+i-1 < k$, we can add a transition of the form $[(x, 0, j), (a, a'), x[i+1..|x|]a, 0, j+i-1)]$. This means that we delete $i-1$ symbols from the first string and match the ith symbol of x against the current input letter a' of the second string. Notice that there might be multiple occurrences of a' within x, and for each of this occurrences, we add a transition, assuming that we can "afford" this number of deletion operations. It might be also the case that we (nondeterministically) decide to delete all of x. This may be an option (in particular) if a' does not occur in x at all. Clearly, the transitions described in the following are only added if $j + |x| < k$. Now, two cases may occur: either, $a' = a$, in which case we add the transition $[(x, 0, j), (a, a), (\varepsilon, 0, j + |x|)]$ (which means that we delete all of x from the first string and then read the compound letter (a, a) in the sense of *no edits*), or $a' \neq a$, which adds the transitions $[(x, 0, j), (a, a'), (a', 1, j + |x| + 1)]$ (which means to delete all of x from the first string, plus the following letter a) and $[(x, 0, j), (a, a'), (\varepsilon, 0, j + |x| + 1)]$ (which means to delete all of x from the first string, plus replacing the letter a with a'), assuming $j + |x| + 1 < k$ in both subcases. Here we have an asymmetry with respect to the case $b = 1$, since we assume that the edit operations are being done in the first string. In this case, since the first head is moving right while the second head stays still, the memory vector gets extended by one position. More specifically, we add a transition of the form $[(x, b, j), (a, a'), (\text{tail}(x)a', b, j + 1)]$ for each $a, a' \in \Sigma$. This also happens if $b = 0$ and $x = \varepsilon$, leading to transitions of the form $[(\varepsilon, 0, j), (a, a'), (a', 1, j + 1)]$.

4. Insertion. If $b = 0$ and $j < k - 1$, then we add (for each $a, a' \in \Sigma$) a transition of the form $[(x, b, j), (a, a'), (xa, b, j + 1)]$. Intuitively, inserting a symbol at the first string corresponds to keeping the first head still, while moving the second head one step to the right. As a consequence, the size of the memory vector increases by one. On the other hand, as in the previous item, there is an asymmetry with respect to the case $b = 1$. In this case, we have to analyze the string x kept in memory. If there is some i with $x[i] = a$, then we can insert $x[1..i - 1]$ into u. This leads to a transition of the form $[(x, 1, j), (a, a'), (x[i + 1..|x|]a', b, j + i - 1)]$ for each $a, a' \in \Sigma$, assuming $j + i - 1 < k$. In the extreme case, we could also decide to insert all that is kept in memory into u. If $a = a'$,

then we add a transition of the form $[(x, 1, j), (a, a'), (\varepsilon, 0, j + |x|)]$, assuming $j + |x| < k$. This means that the whole of x is inserted into u, and moreover, as the compound symbol (a, a) is read, both heads reading u and w advance. If $a \neq a'$, then we add transitions of the form $[(x, 1, j), (a, a'), (a, 0, j+|x|+1)]$ and $[(x, 1, j), (a, a'), (\varepsilon, 0, j+|x|+1)]$, assuming $j + |x| + 1 < k$. Notice that these more complex rules combine inserting all of x into u with either inserting a' into u and hence having to memorize a, or replacing a with a'.

We remark that the Deletion case with $b = 0$ is dual to the Insertion case with $b = 1$. Similarly, the Deletion case with $b = 1$ is dual to the Insertion case with $b = 0$. This duality stems from the fact that when it comes to edit distance, deleting one symbol in one of the strings is equivalent to inserting the corresponding symbol into the other string.

There are actually further cases when one of the components of the compound letter that is read equals $\square$. We leave the details of formulating these corner cases to the reader.

Now, we prove that the automaton defined above really accepts precisely the strings of the form $u \otimes w$ where $\Delta(u, w) < k$. Observe that $\Delta(u, w) < k$ if and only if u can be transformed into w by applying less than k insertions, deletions and replacements. We can describe these operations by special letters. To this end, consider the new alphabet $\Gamma = \Sigma \cup \Sigma' \cup \Sigma'' \cup \Sigma \times \Sigma$, where Σ' (or Σ'', respectively) contains primed (or double-primed, respectively) variants of all letters from Σ. Furthermore, define the morphisms $h_1 : \Gamma^* \to \Sigma^*$ and $h_2 : \Gamma^* \to \Sigma^*$ by

$$h_1(a) = \begin{cases} a, & \text{if } a \in \Sigma \\ b, & \text{if } a = (b, c) \in \Sigma \times \Sigma \\ b, & \text{if } a = b' \in \Sigma' \\ \varepsilon, & \text{if } a = b'' \in \Sigma'' \end{cases} \quad \text{and} \quad h_2(a) = \begin{cases} a, & \text{if } a \in \Sigma \\ c, & \text{if } a = (b, c) \in \Sigma \times \Sigma \\ \varepsilon, & \text{if } a = b' \in \Sigma' \\ b, & \text{if } a = b'' \in \Sigma'' \end{cases}$$

Now, any word $z \in \Gamma^*$ describes how to change $u = h_1(z)$ into $w = h_2(z)$ by using as many edit operations as there are letters from $\Sigma' \cup \Sigma'' \cup \Sigma \times \Sigma$ in z. As we can formally associate a transducer τ_A to the NFA A described above that simply outputs the protocol of edit operations it performed on u, with the additional property that the counter of the NFA is incremented if and only if an edit operation from $\Sigma' \cup \Sigma'' \cup \Sigma \times \Sigma$ is output at this point by the transducer, we can see that the NFA A accepts word $u \otimes w \ : \ \Delta(u, w) < k$. Conversely, any word $z \in \Gamma^*$ with less than k letters from $\Sigma' \cup \Sigma'' \cup \Sigma \times \Sigma$ can be viewed as the output of the constructed transducer τ_A, protocolling less than k insertions, deletions or replacements, so that $h_1(z) \otimes h_2(z) \in L(A)$ holds. A detailed inductive proof is now a tedious but easy exercise. $\square$

It is helpful for understanding the previous reasoning by looking at a simple example: Let $u = ababbaca$ and $w = aabbabbba$. Define $z = ab''abba(c, b)b'b'a$. Here, red letters are deleted from u, blue letters should be inserted into u, and green letter pairs indicate replacements. Then, $h_1(z) = u$ and $h_2(z) = w$. The NFA will sequentially execute the following transitions when digesting

$$u \otimes w = (a, a)(b, a)(a, b)(b, b)(b, a)(a, b)(c, b)(a, b)(\square, a) \ :$$

$$\begin{array}{lll} [(\varepsilon, 0, 0), (a, a), (\varepsilon, 0, 0)] & [(\varepsilon, 0, 0), (b, a), (a, 1, 1)] & [(a, 1, 1), (a, b), (b, 1, 1)] \\ [(b, 1, 1), (b, b), (b, 1, 1)] & [(b, 1, 1), (b, a), (a, 1, 1)] & [(a, 1, 1), (a, b), (b, 1, 1)] \\ [(b, 1, 1), (c, b), (b, 1, 2)] & [(b, 1, 2), (a, b), (a, 0, 4)] & [(a, 0, 4), (\square, a), (\varepsilon, 0, 4)] \end{array}$$

The last transition also illustrates how to deal with reading $\square$, at least in this special case of *no edits*.

As a corollary of Lemma 12, by first determinizing the automaton of the previous lemma and then (basically) complementing final states, plus checking (by a product automaton construction) that words come from convolutions of words, using Lemma 1, we get the following result.

Lemma 13. *Let Σ be an alphabet and $k \in \mathbb{N}_{>0}$. There is a DFA* $\text{Edit}^{\geq}(\Sigma, k)$ *with* $2^{2^{O(k \log |\Sigma|)}}$ *states accepting the following language.*

$$L(\text{Edit}^{\geq}(\Sigma, k)) = \left\{ u \otimes w \in \left((\Sigma \cup \{\square\})^{\times 2}\right)^* \ : \ \Delta(u, w) \geq k \right\}. \tag{2}$$

Let $W \subseteq \Sigma^*$ be a finite set of strings. The *min-diversity* of W, denoted by $\text{MinDiv}(W)$ is defined as the minimum edit distance among any pair of strings in W.

$$\text{MinDiv}(W) = \min_{u,w \in W} \Delta(u, w) \tag{3}$$

The following lemma states that given DFA A, one can construct DFA $\text{MinDiv}(A, k)$ whose language is a suitable encoding of all r-tuples of words in $L(A)^r$ with diversity at least k.

Lemma 14. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be DFA over Σ. For each $r, k \in \mathbb{N}^+$, one can construct a DFA* $\text{MinDiv}(A, r, k)$ *with* $2^{r^2 \cdot 2^{O(k \cdot \log |\Sigma|)}} \cdot |Q|^r$ *many states accepting the language*

$$\{u_1 \otimes \cdots \otimes u_r \in (\Sigma \cup \{\square\})^{\times r} \ : \ \forall i \in [r](u_i \in L(A)) \wedge \text{MinDiv}(\{u_1, \ldots, u_r\}) \geq k\}.$$

Its construction takes $O\left(2^{r^2 \cdot 2^{O(k \log |\Sigma|)}} \cdot |A|^r\right)$ *time.*

Proof. Let $\binom{[r]}{2} = \{\{i, j\} \ : \ i, j \in [r], i \neq j\}$. For each pair $\{i, j\} \in \binom{[r]}{2}$, let $\text{Edit}^{\geq}_{i,j}(\Sigma, k)$ be the automaton that accepts all strings of the form $u_1 \otimes u_2 \otimes \cdots \otimes u_r$ where for each $l \in [r]$, $u_l \in \Sigma^+$ and $\Delta(u_i, u_j) \geq k$, based on Lemma 13. Let A^r be the automaton accepting the set of all strings of the form $u_1 \otimes u_2 \otimes \cdots \otimes u_r$ where for each $l \in [r]$, $u_l \in L(A)$. Then, $\text{MinDiv}(A, r, k)$ can be defined as the automaton that accepts the intersection of the languages $L(A^r)$ with all the languages $L(\text{Edit}^{\geq}_{i,j}(\Sigma, k))$ for $\{i, j\} \in \binom{[r]}{2}$. This implies that the number of states in $\text{MinDiv}(A, r, k)$ is at most the product of the number of states of A^r with the number of states of $\text{Edit}^{\geq}_{i,j}(\Sigma, k))$ for all $\{i, j\} \in \binom{[r]}{2}$. According to Corollary 2, the DFA A^r has at most $(2|Q|)^r + 1$ states. Its construction costs $O(|A|^r)$ time. We claim below that one can define $\text{Edit}^{\geq}_{i,j}(\Sigma, k))$ in such a way that it has at most $2^{2^{O(k \log |\Sigma|)}}$ states. As a consequence, one can construct $\text{MinDiv}(A, r, k)$ in such a way that it has at most $((2|Q|)^r + 1) \cdot \left(2^{2^{O(k \log |\Sigma|)}}\right)^{\binom{[r]}{2}} = 2^{r^2 \cdot 2^{O(k \log |\Sigma|)}} \cdot |Q|^r$ many states. Its construction takes $O(2^{r^2 \cdot 2^{O(k \log |\Sigma|)}} \cdot |A|^r)$ time.

Let $\rho_{i,j} : \Sigma^{\times k} \to \Sigma^{\times 2}$ be the map that sends each tuple $(a_1, a_2, \ldots, a_r)$ to the pair (a_i, a_j). Then, the automaton $\text{Edit}^{\geq}_{i,j}(\Sigma, k)$ can be defined as the automaton that accepts the inverse homomorphic image of $L(\text{Edit}^{\geq}(\Sigma, k))$ under $\rho_{i,j}$. More specifically, the automaton $\text{Edit}^{\geq}_{i,j}(\Sigma, k)$ is constructed from $\text{Edit}^{\geq}(\Sigma, k)$ by replacing each transition $(q, (a, b), q')$ with the set of transitions $\{(q, (a_1, \ldots, a_r), q') \ : \ (a_1, \ldots, a_r) \in \Sigma^{\times r}, \ a_i = a, \ a_j = b\}$. Note that since no new states are created, the number of states in $\text{Edit}^{\geq}_{i,j}(\Sigma, k)$ is equal to the number of states in $\text{Edit}^{\geq}(\Sigma, k)$, that is, $2^{2^{O(k \log |\Sigma|)}}$ (Lemma 13). $\square$

6 Algorithmic Results

In this section we state and prove our main results, starting with two simple facts.

Proposition 15 (Single-Word Automaton)**.** *Let Σ be an alphabet, $w \in \Sigma^*$ and $A(w)$ be the minimal deterministic finite automaton accepting $\{w\}$. Then, $A(w)$ has $|w| + 2$ states.*

Proof. Let $A(w) = (Q, \Sigma, \delta, q_0, F)$ be the DFA defined by setting $Q = \{q_0, q_1, \ldots, q_{|w|+1}\}$, $F = \{q_{|w|}\}$, and $\delta = \{(q_{i-1}, w[i], q_i) : i \in [|w|]\} \cup \{(q_{i-1}, a, q_{|w|+1}) : i \in [|w|], w[i] \neq a\} \cup \{(q_{|w|+1}, a, q_{|w|+1}) : a \in \Sigma\}$. Then, it should be clear that $A(w)$ accepts w and no other string. Minimality follows from the fact that the language $\{w\}$ splits the set Σ^* into $|w| + 2$ Myhill-Nerode equivalence classes. □

In combination with Lemma 7, we obtain the following result.

Corollary 16. *Let $w \in \Sigma^*$. Then, the NFA* $\mathrm{Subseq}(A(w))$ *has $|w| + 1$ states.*

After these preparatory results, let us move on to questions that are more directly related to our discussions on diversity in synchronization. The next algorithmic result is interesting, because we prove intractability of the corresponding decision problem in Theorem 22 below. We can interpret the following theorem also as a statement about FPT-membership of this problem with respect to the parameter $|Q|$, the number of states of the given DFA A.

Theorem 17. *Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Given a word $w \in \Sigma^+$, one can determine in time $O(2^{|Q|} \cdot |w| \cdot (|A| + \log |w|))$ whether w has a subsequence that is synchronizing for A. If yes, in time $O(2^{|Q|} \cdot |w|^2 \cdot (|A| + \log |w|))$, or alternatively, in time $O(2^{2^{|Q|}} \cdot 2^{|A|} \cdot |w| \cdot \log(|w|))$, one can even construct a subsequence-minimal synchronizing word for A that is a subsequence of w.*

Notice that the two algorithmic variants allow us to trade-off long words w against relatively large automata A.

Proof. Let $\mathrm{Subseq}(A(w))$ be the NFA of Corollary 16 with $|w| + 1$ states accepting all subsequences of w. Let $\mathrm{Sync}(A)$ be the automaton accepting all words that are synchronizing for A. By Lemma 10, this DFA has $O(2^{|Q|})$ states and can be constructed in time $O(2^{|Q|}|A|)$. Then, w has some subsequence that is synchronizing for A if and only if $L(\mathrm{Subseq}(A(w))) \cap L(\mathrm{Sync}(A)) \neq \emptyset$, a condition that can be verified in time $O(2^{|Q|}|w|(|A| + \log |w|))$ by first constructing a product automaton (here, an NFA), and then by performing a reachability test. In the same time bound, we can find some $u \in L(\mathrm{Subseq}(A(w))) \cap L(\mathrm{Sync}(A))$ (if it exists).

For the last part, suppose that the intersection above is non-empty. Then, w also contains a subsequence-minimal synchronizing word for A. Using Lemma 9, we can construct an automaton $\mathrm{MinSync}(A)$ that accepts the language of all synchronizing words of A that do not contain any subsequence that is also synchronizing for A. This construction takes time $O(2^{2^{|Q|}} \cdot |\mathrm{Sync}(A)|) = O(2^{2^{|Q|}} \cdot 2^{|A|})$. Observe that $\mathrm{MinSync}(A)$ has at most $O(2^{2^{|Q|}} \cdot 2^{|Q|})$ states. Hence, it is enough to output any word in the language $L(\mathrm{MinSync}(A)) \cap L(\mathrm{Subseq}(A(w)))$. Since $\mathrm{MinSync}(A)$ has $O(2^{2^{|Q|}} \cdot 2^{|Q|})$ states and $A(w)$ has $|w| + 1$ states, we have that one can find a word in the intersection of the languages accepted by these automata in time $O(2^{2^{|Q|}} \cdot 2^{|Q|} \cdot |w| \log(|w|))$.

Alternatively, we can obtain an algorithm running in time $O(2^{|Q|} \cdot |w|^2 \cdot (|A| + \log |w|))$ as follows. Above, we described how to test whether $L(\mathrm{Subseq}(A(w))) \cap L(\mathrm{Sync}(A))$ is non-empty. If the intersection is empty, then w has no subsequence that is synchronizing for A. In the other case, such a subsequence exists. Let $w^{\downarrow i} = w[1..i-1]w[i+1..|w|]$. If $L(\mathrm{Subseq}(A(w^{\downarrow i}))) \cap L(\mathrm{Sync}(A)) = \emptyset$ for every $i \in [|w|]$, then we know that w is a subsequence-minimal synchronizing word for A. Otherwise, if this intersection is non-empty for some i, then we know that $w^{\downarrow i}$ contains a subsequence-minimal synchronizing word for A. We then update w to $w^{\downarrow i}$, and repeat the process described in this paragraph. The algorithm operates in time $O(2^{|Q|} \cdot |w|^2 \cdot (|A| + \log |w|))$, since we need to delete at most $|w|$ letters, and at each deletion, one needs $O(2^{|Q|} \cdot |w| \cdot (|A| + \log |w|))$ steps to determine if $L(\mathrm{Subseq}(A(w^{\downarrow i}))) \cap L(\mathrm{Sync}(A)) \neq \emptyset$. □

In our next result we deal with the problem of finding a sufficiently diverse set W of subsequence-minimal synchronizing words for an automaton A that satisfy an additional constraint. More specifically, we require that each word in W belongs to the language of an automaton B given at the input. In this setting, A may be regarded as the specification of a system (say a robot) that interacts with an environment. Here, B models the set of all sequences of actions that are legal in the environment. The requirement that $W \subseteq L(B)$ ensures that the synchronizing words under consideration correspond to sequences of actions that are legal in the environment. Note that it makes sense to assume that A is fixed (say, a robot of a certain model), and that the environment may vary (e.g., the environment where the robot will be deployed). The next theorem analyzes the computational complexity of this problem parameterized by the diversity parameters r and k. Since A is assumed to be fixed, in the statement of the theorem we hide the dependencies on the DFA A in the function f_A.

Theorem 18. *Let $A = (Q, \Sigma, \delta)$ be a DFA and $B = (Q', \Sigma, \delta', Q'_0, F')$ be an NFA. One can determine in time $O(f_A(r, k) \cdot |Q'|^r \log(|Q'|))$ whether there is a set $W \subseteq L(B)$ with r strings such that each word in W is subsequence-minimal synchronizing for A and* $\mathrm{MinDiv}(W) \geq k$.

Proof. By combining Corollary 11 with Lemma 14, we can construct a DFA A' accepting the language of all compound words $u_1 \otimes \cdots \otimes u_r \in (\Sigma \cup \{\square\})^{\times r}$ such that for each $i \in [r]$, u_i is a subsequence-minimal synchronizing word for A, and $\mathrm{MinDiv}(\{u_1, \ldots, u_r\}) \geq k$. The DFA A' has $2^{r^2 \cdot 2^{O(k \cdot \log|\Sigma|)}} \cdot (2^{|Q|})^r$ many states and can be constructed in time $2^{r^2 \cdot 2^{O(k \cdot \log|\Sigma|)}} \cdot (2^{|Q|} \cdot |\Sigma| \cdot |Q|)^r$. Conversely, again by an $(r+1)$-fold product automaton construction, also using Lemma 1 and Corollary 2, an NFA B' with $(2|Q'|)^r + 1$ many states can be constructed that accepts the language

$$\{u_1 \otimes u_2 \cdots \otimes u_r : \forall i \in [r](u_i \in L(B))\} .$$

Checking if the product automaton C of A' and B' accepts any compound words solves the proposed problem. This again amounts to reachability testing. This final check takes time linear in the size of C, which is hence

$$O\left(2^{r^2 \cdot (|Q| + 2^{O(k \cdot \log|\Sigma|)})} \cdot |Q'|^r \cdot \left(r \log(|Q'|) + (r^2 \cdot (|Q| + 2^{O(k \cdot \log|\Sigma|)})) + |\Sigma|^r\right)\right) .$$

Sorting these terms, we get the claim with a suitably defined function $f_A(r, k)$. ☐

Actually, the previous result can be viewed as a diversity result concerning *synchronization under regular constraints* as introduced in [30]. This variation of the classical synchronization theme comes in with the constraint automaton B, but by setting $L(B) = \Sigma^*$, we get back to the classical theme.

Furthermore, as a direct consequence of Theorem 18 and Corollary 16 in combination with Lemma 7, we have that the problem of finding a set of sufficiently diverse subsequences of a word w that are subsequence-minimal synchronizing words for A can be solved by an algorithm with an FPT dependency on the parameters A (i.e., $|Q|$ and $|\Sigma|$) and k and an XP dependency on the parameter r.

Corollary 19. *Let $A = (Q, \Sigma, \delta)$ be a DFA and w be a word in Σ^*. One can determine in time $O(f_A(r, k) \cdot |w|^r \log(|w|))$ whether there is a set $W = \{w_1, \ldots, w_r\}$ of subsequences of w such that each word in W is subsequence-minimal synchronizing for A and* $\mathrm{MinDiv}(W) \geq k$.

Remark 20. *The reader might have wondered why we allowed a certain asymmetry in the formulation of Theorem 18 between the automata A and B, requiring the first one to be a DFA and allowing the second one to be an NFA. There are at least two reasons for doing so: (a) We consider Corollary 19 as being quite interesting, and here B being an NFA is important because*

of Lemma 7 that is the basis of Corollary 16, as here an NFA accepting all subsequences of w is produced, basically by nondeterministically guessing which parts of the word w are read. (b) Turning A into an NFA is possible in principle, in particular, as we consider the automaton A to be fixed in the interpretation of our results from the viewpoint of fixed-parameter tractability. However, there are some subtleties concerning the very definition of synchronizing words in the case of NFA; we only refer to [35, 50]. We wanted to avoid diving into these details in this paper.

7 Hardness Results

Below we show that the very basic problem of determining whether a given word is subsequence-minimal synchronizing for a given automaton A is already coNP-hard.

Definition 21 (Min-SubSequence-SW)**.**
Given: *DFA $A = (Q, \Sigma, \delta)$ and a word $w \in \Sigma^*$ synchronizing A.*
Question: *Is w a minimal synchronizing word with respect to the subsequence order?*

Theorem 22. Min-SubSequence-SW *is coNP-complete, even for DFAs over a binary input alphabet.*

This hardness result (and even more the hardness result for the counting class #P that we prove as our third main result of this section below in Theorem 29) explains why we have to develop exponential-time algorithms for the suggested diversity problems.

We prove the hardness claim by a reduction from Hitting Set to the complementary problem of Min-SubSequence-SW which asks for the existence of a smaller subsequence of w which is synchronizing.

Definition 23 (Hitting Set)**.**
Given: *Collection C of non-empty subsets of a finite set S and integer k.*
Question: *Is there a subset $S' \subseteq S$ with $|S'| \leq k$ such that for all $c \in C$, $S' \cap c \neq \emptyset$?*

Proof. For membership in coNP, we also consider the complementary problem. Hence, given w, we first check if w is synchronizing at all. If not, it cannot be minimal synchronizing. If it is, we guess a subsequence of w and check again if it is synchronizing. This proves memberhip in NP of the complementary problem.

For the coNP-hardness, we argue as follows. Let $C = \{c_0, c_1, \ldots, c_{m-1}\}$ be a collection of subsets of $S = \{s_0, s_1, \ldots, s_{n-1}\}$, and $k \in \mathbb{N}$. We construct from C, S, k a DFA $A = (Q, \{a, b\}, \delta)$ as follows. We set $Q = C \times (S \cup \{s_n\}) \times [k+1] \cup q_{\text{sync}}$. To underline the set-theoretic nature of all states but q_{sync}, we also talk about state-tuples in the following. The DFA A will consist of $|C|$ many chains leading to the synchronizing state q_{sync} which are not connected to each other. Each chain consists of k many listings of the elements in S such that for each element $s \in S$ which appears in the corresponding subset c, it is possible to shortcut the chain at the nodes corresponding to s. More formally, we define δ as follows. For $c_i \in C$, $s_j \in S$ and $l \in [k+1]$ we set $\delta((c_i, s_j, \ell), a) = (c_i, s_{j+1}, \ell)$ if $j < n$ and $\delta((c_i, s_j, \ell), a) = (c_i, s_j, \ell)$ if $j = n$. For the letter b we set $\delta((c_i, s_j, \ell), b) = q_{\text{sync}}$ if $s_j \in c_i$. Otherwise, we set $\delta((c_i, s_j, \ell), b) = (c_i, s_0, \ell)$ if $j < n$ and $\delta((c_i, s_j, \ell), b) = (c_i, s_0, \ell + 1)$ if $j = n$ and $\ell \leq k$. For $j = n$ and $\ell = k + 1$ we set $\delta((c_i, s_j, \ell), b) = q_{\text{sync}}$. For the state q_{sync} we set $\delta(q_{\text{sync}}, a) = \delta(q_{\text{sync}}, b) = q_{\text{sync}}$.

As the input synchronizing word, we set $w = (a^n b)^{k+1}$.

Clearly, w is a synchronizing word for A as it simply traverses through the chains in parallel without the attempt of taking a shortcut. Note that before the first symbol b is read, each segment listing the set S has multiple active states from which some have s_n in the second

entry of the state-tuple. Hence, the first letter b cannot bring all active states of a chain to the synchronizing state q_{sync} and can only have the effect of resetting all chains to state-tuples with element s_0 in the second component.

If w contains a shorter synchronizing subsequence, we can find one by removing a's. In fact, we can only find a synchronizing subsequence with fewer b's if the instance specified by C, S even admits a hitting set of size $< k$.

First, assume $S' = \{s_{i_1}, s_{i_2}, \ldots, s_{i_\ell}\}$ is a hitting set for C of size $|S'| \leq k$. Then, the word $w_s = ba^{i_1}ba^{i_2}b \ldots a^{i_\ell}b$ is a synchronizing word for A and a subsequence of w with $|w_s| < |w|$. After reading a b (and especially after reading the initial b) the active states of all chains have s_0 in the second position of their state-tuple. Hence, the word w_s maps the active states of some chain corresponding to the set c_i to some states $\{(c_i, s_j, h) \colon h \leq k+1\}$ for which $s_j \in c_i$ and $s_j \in S'$. These states are then mapped to q_{sync} with a letter b.

For the other direction, assume w_s is a subsequence of w and a synchronizing word for A with $|w_s| < |w|$. W.l.o.g., we can assume that w_s starts with b, as the first a's are only making noticeable progress towards q_{sync} if we do not try to make shortcuts with the letter b, because every shortcut throws the active states of chains, which cannot use this shortcut, back to the last state with s_0 in the second component. Hence, note that if we take *any* shortcut for *some* chain to reach the sync-state, then *all* states must be mapped to the sync-state via a shortcut. Since w is the shortest synchronizing word which does not use a shortcut, we conclude that w_s uses shortcuts and hence all chains must reach q_{sync} via shortcuts. As argued above, the first b cannot map all active states of a chain to the sync-state. Collecting the second components of active states before each succeeding letter b then gives a hitting set for C, as each chain must be left via a shortcut. □

Recall that above, in Theorem 17 we proved that Min-SubSequence-SW, parameterized by the number of states of the input DFA, belongs to FPT.

The following result explains also to some extent why the problems that we consider in this paper are computationally hard ones. Note that in the classical setting, length-minimal synchronizing words (if existent at all) are of polynomial size only [84]. Requiring subsequence-minimality instead of length-minimality changes the picture drastically, as then some synchronizing words with this additional property can be of exponential length.

Proposition 24. *Some subsequence-minimal synchronizing words can be of exponential length, even for DFAs with a ternary input alphabet.*

Proof. Let $P = \{p_1, p_2, \ldots, p_k\}$ be a set containing the first k prime numbers and let A be a unary DFA over the letter a consisting of a synchronizing state q_{sync} and k independent cycles where the ith cycle have length p_i. Then, enhance A with two more letters b and c, where b maps all states of a cycle to one single state on this cycle and c takes the predecessor under the letter a of this state to some designated synchronizing state and is the identity on all other states. Figure 1 represents A for $k = 3$. More formally, $A = (Q, \{a, b, c\}, \delta)$ is defined as follows: $Q = \{q_{\text{sync}}\} \cup \{q_{i,j} \colon i \in [k], j \in [p_i]\}$, for all $i \in [k]$ and $j \in [p_i]$, $\delta(q_{i,j}, a) = q_{i,(j+1) \bmod p_i}$, $\delta(q_{i,j}, b) = q_{i,1}$, $\delta(q_{\text{sync}}, a) = \delta(q_{\text{sync}}, b) = \delta(q_{\text{sync}}, c) = q_{\text{sync}}$, for $i \in [k]$ and $j \in [p_i - 1]$, $\delta(q_{i,j}, c) = q_{i,j}$ and $\delta(q_{i,p_i}, c) = q_{\text{sync}}$. By definition of δ, A is deterministic. Let $P_k = \prod_{i=1}^{k} p_i$. Then, $w = ba^{P_k - 1}c$ is a synchronizing word for A. As q_{sync} is a trap state, it must be the synchronizing state of A and hence any possibly synchronizing subsequence of w must end in c. By the choice of the prime numbers, all states on all cycles can only be synchronized after P_k steps, and in order to have a clear starting point, we need the letter b at the beginning, i.e., no proper subsequence of w is synchronizing. □

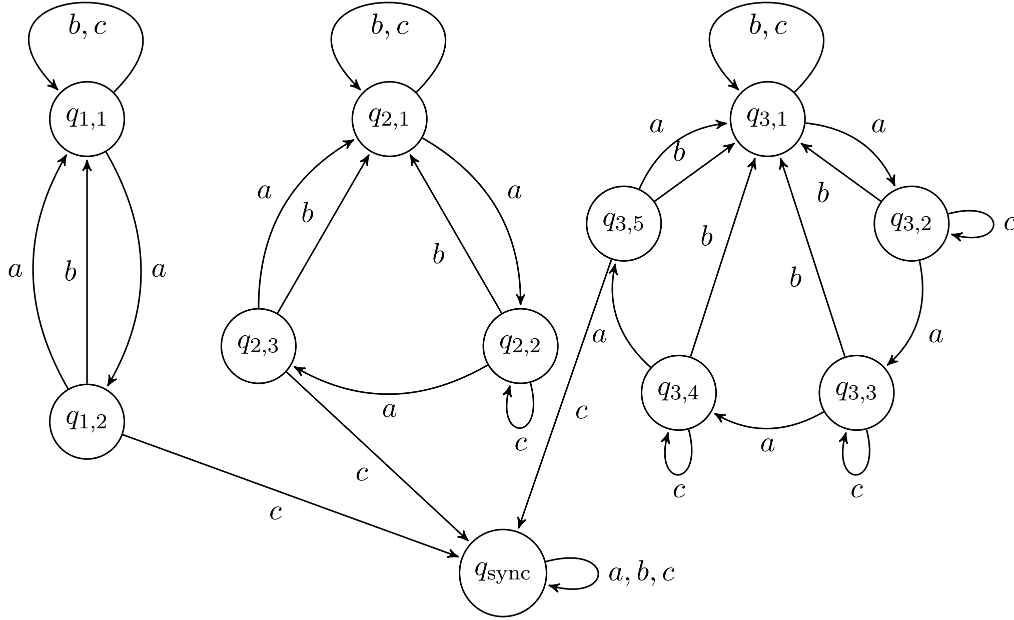

Figure 1: Automaton A constructed in the proof of Proposition 24 for $k = 3$.

Remark 25. *The previous construction also implies that infix-minimal synchronizing words can be of exponential size.*

For binary input alphabets, we do not have an example, while for unary alphabets, observe that subsequence-minimality is the same as length-minimality, so that we cannot expect to find subsequence-minimal synchronizing words of exponential length in that case.

Next, we define two further combinatorial questions, which are in fact the central problems of our study. In the last section, we showed some algorithmic results that can be viewed as exponential-time algorithms for these problems. The fact that these algorithms exceed polynomial time is (with hindsight) justified by the hardness results that we will prove in the following.

Definition 26 (Diversity-Sync)**.**
Given: *DFA $A = (Q, \Sigma, \delta)$ and $k \in \mathbb{N}$, encoded in binary.*
Question: *Is there a set $W = \{w_1, w_2, \ldots, w_r\}$ of subsequence-minimal synchronizing words for A such that $\sum_{1 \leq i < j \leq r} \Delta(w_i, w_j) \geq k$?*

A natural variant of the problem above is obtained when we ask for a set of subsequence-minimal synchronizing words of some minimal *cardinality* and neglect the diversity of the set. Note that the condition that the words are subsequence-minimal implies that the words have a pairwise distance greater than one.

Definition 27 (Card-Sync)**.**
Given: *DFA $A = (Q, \Sigma, \delta)$ and $r \in \mathbb{N}$, encoded in binary.*
Question: *Is there a set $W = \{w_1, w_2, \ldots, w_r\}$ of subsequence-minimal synchronizing words for A?*

With a slight adaption of the previous construction, we show in the following that the problems Diversity-Sync and Card-Sync are NP-hard.

As we have shown in Theorem 22, verifying the subsequence minimality of a synchronizing word is coNP-hard and hence the problem is unlikely to be contained in NP, because we cannot

use a guess & check approach. Further, for binary encoded parameters k and r, we obtain coNP-hardness from the reduction in Theorem 22 and hence should at least climb to Δ_2^P in the polynomial-time hierarchy for a membership attempt. Note that the NP-hardness shown below does not require binary parameters k and r, whereas coNP-hardness does (so far).

Theorem 28. *The problems* Diversity-Sync *and* Card-Sync *are NP-hard.*

Proof. We give a reduction from the Hitting Set problem and adapt the construction of Theorem 22. We add to each chain c_i n new states $(c_i, [j])$ with $j \leq n$. For these states, we set $\delta((c_i, j), a) = (c_i, j+1)$ for $j < n$ and $\delta((c_i, n), a) = (c_i, s_n, 1)$. For the letter b we set $\delta((c_i, j), b) = (c_i, j)$. From the previously defined transitions we redefine the transitions for b if $s_j \neq s_n$ to the identity, i.e., we set $\delta((c_i, s_j, \ell), b) = (c_i, s_j, \ell)$. We further introduce a new letter c which acts as follows. For $s_j \neq s_n$ we set for $s_j \in c_i$, $\delta((c_i, s_j, \ell), c) = q_{\text{sync}}$ and for $s_j \neq s_n$, $s_j \notin c_i$ we set $\delta((c_i, s_j, \ell), c) = (c_i, j+1)$. For all other states, c acts as the identity.

The critical observation is that synchronizing words which use shortcuts will have the form $a^n b a^{i_1} c a^{n-i_1} b a^{i_2} c a^{n-i_2} b \ldots a^{i_\kappa} c$ and contain $w = (a^n b)^{k+1}$ as a subsequence if $\kappa > k$ and are hence not minimal. Therefore, setting the diversity lower bound to one and requiring $r = 2$ (i.e., two elements should be contained in the diverse set) concludes the reduction as the instance C, S has a hitting set of size at most k if and only if there are at least two subsequence-minimal synchronizing words for the constructed automaton, namely a listing $a^n b a^{i_1} c a^{n-i_1} b a^{i_2} c a^{n-i_2} b \ldots a^{i_\ell} c$ of the hitting set and the word $w = (a^n b)^{k+1}$. □

We show in the next theorem that it is #P-hard to compute the maximal diversity of the set of subsequence-minimal synchronizing words for a given DFA A.

Theorem 29. *Let A be a DFA. Then, on input A, computing the diversity of the set of all subsequence-minimal synchronizing words of A is #P-hard.*

Proof. We give a reduction from 3-CNF SAT to the subsequence-minimal synchronization problem and show that the number of satisfying variable assignments for some 3-CNF formula φ can be extracted from the diversity of a maximal set of subsequence-minimal synchronizing words for the constructed automaton. Hence, by computing the diversity of the set LMinSync(A) of all subsequence-minimal synchronizing words of A, we would also be able to compute the number of satisfying assignments for φ. The reduction is an adaption of the construction for the short synchronizing word problem by Eppstein and Rystsov [28, 73].

Let $\varphi = \{c_1, c_2, \ldots, c_m\}$ be a Boolean formula in 3-CNF over the set of variables $V = \{x_1, x_2, \ldots, x_n\}$. The idea of the construction is to represent each clause by a chain of states leading into a single sink-state. Each subsequence-minimal synchronizing word will contain a subsequence of 0's and 1's which corresponds to a variable assignment of the variables $x_1, x_2, \ldots, x_n$, in this order. If a clause is satisfied by the assignment of some variable, then the letter corresponding to this assignment will shortcut this chain into the sink-state. Therefore, subsequence-minimal synchronizing words that correspond to satisfying assignments will be shorter than those corresponding to unsatisfying assignments. In order to avoid that a satisfying assignment might be a subsequence of a synchronizing word corresponding to an unsatisfying assignment, we separate the 0's and 1's corresponding to variable assignments in a synchronizing word by a sequence of n many letters a, each. This will allow us to extract the number of satisfying assignments from the diversity of LMinSync(A).

We now give the details of the construction. It might be beneficial to consider Figure 2 while following the details. We construct from φ a DFA $A = (Q, \{0, 1, a\}, \delta)$ where Q is defined as $Q = \{q_{\text{sync}}\} \cup (\varphi \cup \{\emptyset\}) \times (V \cup \{x_{n+1}\}) \times \{0, 1, \ldots, n\} \setminus \{\emptyset\} \times \{x_{n+1}\} \times \{n\}$. For the transition

function δ, we define $\delta(q_{\text{sync}}, \sigma) = q_{\text{sync}}$ for all $\sigma \in \{0, 1, a\}$. For the other states, we define

$$\delta((c_i, x_j, k), 0) = \begin{cases} (c_i, x_j, k) & \text{if } k \neq n, \\ (c_i, x_{j+1}, 0) & \text{if } k = n \wedge j \leq n \wedge \neg x_j \notin c_i, \\ q_{\text{sync}} & \text{otherwise} \end{cases}$$

$$\delta((c_i, x_j, k), 1) = \begin{cases} (c_i, x_j, k) & \text{if } k \neq n, \\ (c_i, x_{j+1}, 0) & \text{if } k = n \wedge j \leq n \wedge x_j \notin c_i, \\ q_{\text{sync}} & \text{otherwise} \end{cases}$$

$$\delta((c_i, x_j, k), a) = \begin{cases} (c_i, x_j, k) & \text{if } k = n, \\ (c_i, x_j, k+1) & \text{if } k \neq n \wedge (c_i, x_j, k) \neq (\emptyset, x_{n+1}, n-1) \\ q_{\text{sync}} & \text{otherwise} \end{cases}$$

Note that due to the chain corresponding to the artificial clause $\emptyset$, each synchronizing word must have length at least $(n+1)^2 - 1$. Further, note that every word in the language $(a^n\{0,1\})^n a^n$ maps all states of A into a subset of $\{q_{\text{sync}}\} \bigcup_{c_i \in \varphi} \{(c_i, x_{n+1}, n)\}$. Hence, every word in the language $(a^n\{0,1\})^{n+1}$ is synchronizing but not necessarily subsequence-minimal. In contrast, every *synchronizing* word in $(a^n\{0,1\})^n a^n$ is a subsequence-minimal synchronizing word.

After the prefix a^n all states of A are mapped into the set $\{q_{\text{sync}}\} \bigcup_{c_i \in \varphi, j \leq n+1} \{(c_i, x_j, n)\}$. Hence, the only states which might not be mapped to q_{sync} by a word from $(a^n\{0,1\})^n a^n$ are the $n+1$ leftmost states of each chain (corresponding to c_i), which are mapped to the state (c_i, x_1, n) after reading a^n.

If $\beta \colon V \to \{0,1\}$ is a variable assignment that *satisfies* a clause $c_i \in \varphi$, then the word $w = a^n\beta(x_1)a^n\beta(x_2)\ldots a^n\beta(x_n)a^n$ maps *all* states in the chain corresponding to the clause c_i to the sink-state q_{sync} as then, the sub-word $\beta(x_1)a^n\beta(x_2)\ldots a^n\beta(x_n)a^n$ maps the state (c_i, x_1, n) directly to q_{sync} via a short-cut transition corresponding to the first satisfied literal in c_i. Therefore, w is a subsequence-minimal synchronizing word for A if and only if β is a variable assignment that satisfies *each* clause in φ.

If β does *not* satisfy some clause c_i, then w is not a synchronizing word for A, since the state $(c_i, x_1, 0)$ is mapped to (c_i, x_{n+1}, n) by w and not to q_{sync}, as no short-cut transition was taken in the chain c_i by w. Then, both words $w0$ and $w1$ are *subsequence-minimal* synchronizing words for A as *deleting* some letters from w cannot change the variable assignment encoded in w, since two positions encoding variable assignments in w are n letters apart and deleting more than one letter from $w0$ or $w1$ would result in a word of length less than $(n+1)^2 - 1$ which is hence not synchronizing.

Next, we consider the diversity of the set of all subsequence-minimal synchronizing words $\text{LMinSync}(A)$ of A. Note that $\text{LMinSync}(A) \subseteq (a^n\{0,1\})^n a^n\{0,1,\epsilon\}$ as any other synchronizing word u would cause the automaton to stall somewhere, i.e., u would contain a position i such that $\delta(Q, u[1..i]) = \delta(Q, u[1..i+1])$, implying that the letter at position $i+1$ could be removed from u, thus yielding a shorter synchronizing word being a subsequence of u. Hence, we can split the set $\text{LMinSync}(A)$ into the disjoint sets $\text{sat-LMinSync}(A) \subseteq (a^n\{0,1\})^n a^n$ and $\text{unsat-LMinSync}(A) \subseteq (a^n\{0,1\})^{n+1}$. As discussed earlier, if β is an unsatisfying variable assignment, then both words

$$a^n\beta(x_1)a^n\beta(x_2)\ldots a^n\beta(x_n)a^n0 \qquad \text{and} \qquad a^n\beta(x_1)a^n\beta(x_2)\ldots a^n\beta(x_n)a^n1$$

are subsequence-minimal synchronizing words. Further, each string in $(a^n\{0,1\})^n a^n$ will appear as a prefix of some word in $\text{LMinSync}(A)$.

Note that the edit distance of two words w_1 and w_2 in LMinSync(A) is at most $n+1$ and that w_1 and w_2 can only differ at the positions $i \cdot (n+1)$ for $1 \leq i \leq n+1$. Due to the block of a's of length n between any of those positions, a shortest edit sequence that transforms w_1 into w_2 consists of changing directly the letters at the positions $i \cdot (n+1)$ only. Any other type of edit sequence would demand to delete at least one block of a's (costing n edit operations) and inserting again a block of a's at some other position (costing n edit operations).

As each string in $(a^n\{0,1\})^n a^n$ will appear as a prefix of some word in LMinSync(A), we first compute the sum of Hamming distances of the set of binary strings of length exactly n. There are 2^n many binary strings of length n. For each position $1 \leq i \leq n$, exactly half of the strings have a 0 at position i and the other half have a 1 at position i. Each of the $2^n/2$ strings with a 0 at position i adds to the total Hamming distance for this position a weight of value 1 for each of the $2^n/2$ many strings with a 1 at position i. Summing over all n positions, this gives us a total Hamming distance of $\frac{2^n}{2} \cdot \frac{2^n}{2} \cdot n = n \cdot (2^{n-1})^2 = n \cdot 2^{2n-2}$. As in the set $(a^n\{0,1\})^n a^n$ between each adjacent 0's or 1's, there are n many letters a, the total diversity of the set $(a^n\{0,1\})^n a^n$ is the total Hamming distance of the set $\{0,1\}^n$, as the blocks of a's do not contribute to the minimal edit-distance and *shifting* a 0 or 1 over a block of a's would be too expensive, as discussed above.

Now, observe that only the non-satisfying assignments have an additional letter at the end of the word. Hence, every word $w \in$ unsat-LMinSync(A) contributes an additional weight of 1 for each word in sat-LMinSync(A). It further contributes an additional weight of 1 for each unsatisfying assignment of φ, namely, for those words in unsat-LMinSync(A) that have the complementary value of the last position of w as their last position. Note that the words in unsat-LMinSync(A) only add a 1 for each unsatisfying assignment and not for each word in unsat-LMinSync(A). Hence, the total diversity of LMinSync(A) is

$$n \cdot 2^{2n-2} + |\,\text{unsat-LMinSync}(A)| \cdot 2^n.$$

Therefore, if the diversity of LMinSync(A) is d, we can obtain the number of satisfying assignments #sat as

$$\#\text{sat} = 2^n - \left(d - n \cdot 2^{2n-2}\right)/2^n = 2^n - \frac{d}{2^n} + n \cdot 2^{n-2}.$$

Hence, computing the diversity d of the set of subsequence-minimal synchronizing words of A would also allow us to compute the number of satisfying assignments for φ in polynomial time. □

8 Conformant Planning

Conformant planning [37, 78] is the task of finding a sequence of actions for a planning problem that ensures that the goal will be achieved regardless of the initial state and of the nondeterminism of the planning domain. The essence of many planning problems can be abstracted using the framework of automata theory [19, 20]. In this section, we use this point of view to define a suitable notion of diversity of solutions in the context of conformant planning. Apart from some notational differences, our terminology is borrowed from [19].

A *planning domain* can be abstracted as a 4-tuple $D = (Q, \Sigma, \delta, P)$, where Q is a set of *states*, Σ is a set of *actions*, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, and P is a function that assigns a set $P(q)$ of *propositions* (or *beliefs*) to each state $q \in Q$. Intuitively, $P(q)$ is the set of beliefs that are known to hold at state q, and a transition $(q, a, q') \in \delta$ indicates that the set of beliefs $P(q)$ should be updated to $P(q')$ if action a is taken from state q. Note that that the relation δ may be nondeterministic, meaning that for some state $q \in Q$ and some action $a \in \Sigma$,

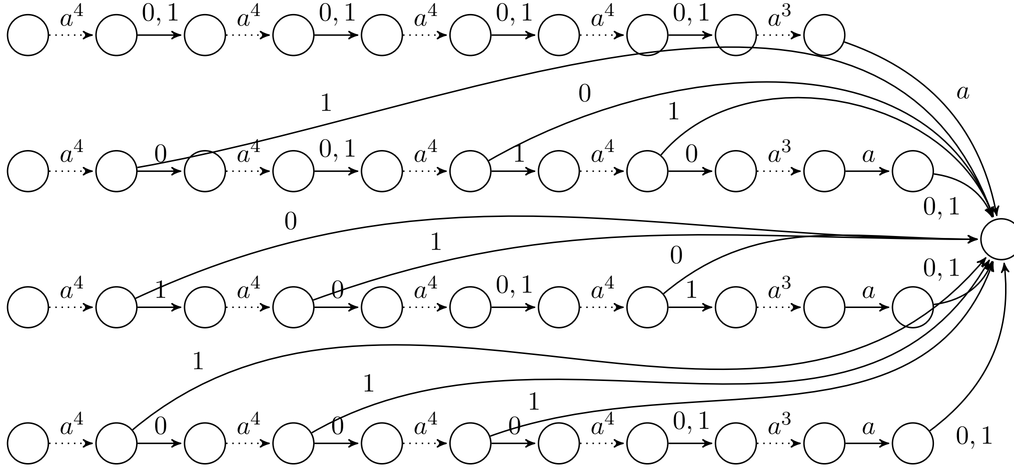

Figure 2: Automaton constructed in the proof of Theorem 29. As an example, the automaton for the formula $\varphi = \{c_1, c_2, c_3\}$ with the clauses $c_1 = \{x_1, \neg x_3, x_4\}$, and $c_2 = \{\neg x_1, x_2, \neg x_4\}$, $c_3 = \{x_1, x_2, x_3\}$ is drawn ($n = 4$). Dotted transitions labeled with words represent a chain labeled by the letters of the word in order where all other letters are realized as self-loops. All not explicitly drawn transitions are realized as self-loops.

there may be two states q' and q'' such that both (q, a, q') and (q, a, q'') belong to δ. In this case, the result of taking the action a at state q is regarded as being undetermined. A *planning problem* is a triple (D, I, G) where $D = (Q, \Sigma, \delta, P)$ is a planning domain, $I \subseteq Q$ is a set of *initial states*, and $G \subseteq Q$ is a set of *goal states*.

A word $u \in \Sigma^*$ is called a *plan*. An action a is said to be *applicable* in a state q if there is some state q' such that $(q, a, q') \in \delta$. Such an action a is applicable in a set of states S if it is applicable in *every* state of S. A plan u is said to be *applicable* in a set of states S if either $u = \varepsilon$ and S is non-empty, or $u = au'$ for some action a and some plan $u' \in \Sigma^*$, a is applicable on S and u' is applicable in $\delta(S, a)$. The notion of a conformant plan is captured by the following definition.

Definition 30 (Conformant Plan)**.** *Let (D, I, G) be a planning problem with planning domain $D = (Q, \Sigma, \delta, P)$. A plan $u \in \Sigma^*$ is* conformant *for (D, I, G) if the following conditions are satisfied:*

1. u is applicable in I, and

2. $\delta(I, u) \subseteq G$.

Intuitively, the two conditions guarantee that a conformant plan u achieves a goal regardless of the initial state, and of the nondeterministic actions that may occur during the execution of u. The following lemma, which is an analogue of Lemma 10 in the context of conformant planning, gives an upper bound on the number of states in a DFA accepting all conformant words for a given planning problem (D, I, G).

Lemma 31 (Conformant Words)**.** *Let $D = (Q, \Sigma, \delta, P)$ be a planning domain, and (D, I, G) be a planning problem. One can construct in time $|\Sigma| \cdot 2^{O(|Q|)}$ a DFA* $\mathrm{Conf}(D, I, G)$ *with $2^{|Q|}$*

states such that

$$L(\mathrm{Conf}(D, I, G)) = \{u \ : \ u \textit{ is conformant for } (D, I, G)\}.$$

Proof. We let $\mathrm{Conf}(D, I, G)$ be the DFA $\mathcal{A}$ whose set of states $\mathcal{Q}$ is the set of all subsets of Q, I is the unique initial state, and the set of final states $\mathcal{F}$ is the set of all non-empty subsets of G. The state $\emptyset$ acts as a trap state of $\mathrm{Conf}(D, I, G)$. The transition function $\Delta : \mathcal{Q} \times \Sigma \to \mathcal{Q}$ sends each pair $(S, a) \in \mathcal{Q} \times \Sigma$ to the state $\{q' \ : \ \exists q \in S, (q, a, q') \in \delta\}$ if a is applicable in every state of S, and to the state $\emptyset$ if a is not applicable from some state of S. Note that the automaton $\mathrm{Conf}(D, I, G)$ has $2^{|Q|}$ states and can be constructed in time $|\Sigma| \cdot 2^{O(|Q|)}$. Additionally, by construction, we have that a plan $u = a_1 a_2 \ldots a_n$ is accepted by $\mathrm{Conf}(D, I, G)$ if and only if the plan u is applicable in I and $\delta(I, u) \subseteq G$, or, in other words, if and only if u is conformant for (D, I, G). □

Notice that although the construction of the previous proof is reminiscent of the powerset construction explained in Section 2, starting with the NFA specified by $(Q, \Sigma, \delta, I, G)$, it is not the same, as the transition function has to be defined differently, because an action a is adimissible to a set of states S if *all* states in S have valid transitions using a. More precisely, if an action a is not applicable in some state of S, then $\Delta(S, a) = \emptyset$. Also, the set of final states is defined differently.

Let (D, I, G) be a planning problem and u be a conformant plan for (D, I, G). We say that u is subsequence minimal if no proper subsequence of u is a conformant plan for (D, I, G). The following corollary, which is an analogue of Corollary 11 in the realm of conformant planning, states that one can construct an automaton accepting precisely the set of subsequence-minimal conformant words for a given planning problem (D, I, G). The proof of this corollary follows straightforwardly by plugging the automaton $\mathrm{Conf}(D, I, G)$ of Lemma 31 into Lemma 9.

Corollary 32. *Let (D, I, G) be a planning problem where $D = (Q, \Sigma, \delta, P)$. One can construct in time $O(|\Sigma| \cdot 2^{2^{|Q|}+|Q|})$ a DFA $\mathrm{ConfMinSubseq}(A)$ with at most $2^{2^{|Q|}+|Q|}$ many states accepting the language*

$$L(\mathrm{ConfMinSubseq}(A)) = \{u : u \textit{ is subsequence-minimal conformant for } (D, I, G)\}.$$

The next theorem is an analogue of Theorem 18 in the context of conformant planning. In this theorem we consider the problem of computing a sufficiently diverse set W of subsequence-minimal conformant plans for a planning problem (D, I, G), with the property that each word in W belongs to the language of an automaton B given at the input. The intuition is that B is a specification of all legal plans that could occur in a given environment independent of (D, I, G). The proof of this theorem is identical to the proof of Theorem 18 with the exception that Corollary 32 is used instead of Corollary 11. Notice that it might well be interesting to produce several different conformant plans for the planner's decision (who might have background knowledge not formalized by the underlying logic), but clearly not too many not to confuse the planner.

Theorem 33. *Let (D, I, G) be a planning problem with planning domain $D = (Q, \Sigma, \delta, P)$. Given an NFA $B = (Q', \Sigma, \delta', Q'_0, F')$, one can determine in time $O(f_D(r, k) \cdot |Q'|^r \log(|Q'|))$ whether there is a set $W \subseteq L(B)$ with r plans such that each plan in W is subsequence-minimal conformant for (D, I, G) and $\mathrm{MinDiv}(W) \geq k$.*

9 Further Applications

In this section, we describe several applications of the notion of a synchronizing word in the context of artificial intelligence, different from conformant planning that was dealt with in the previous section. We also describe possible ways in which the notion of *solution diversity* may enhance these applications.

9.1 Part Orienters

Part orienters are a type of mechanical device used to reorient objects from an unknown initial direction to a predetermined direction. These devices, which are very common for instance, in assembly lines, provide a natural application for the notions of synchronizing automata and synchronizing words [3]. In a pioneering work in this area, Natarajan modeled part orienters as deterministic complete automata where each state corresponds to a possible direction (assumed to be finitely many), and each letter corresponds to the application of a direction modifier [67]. Intuitively, each synchronizing word in such an automaton A corresponds to the application of a sequence of modifications that sends a part to a predetermined direction (state) no matter what the initial direction (state) of the part at the beginning of the process.

The original motivation of designing a part orienters was revisited in [83] where Türker and Yenigün modelled the design of an assembly line, which again brings a part from an unknown orientation into a known orientation, where different modifiers have different costs. For example, a robot arm is much more expensive than a simple obstacle wall. Therefore, they enhance the alphabet Σ with a cost function $c : \Sigma \to \mathbb{N}$. It is natural then to ask for the existence of a synchronizing word of minimum cost. Or, for a given $\alpha \in \mathbb{N}$, to find a synchronizing word of cost at most α.

It is worth noting for each cost value $\alpha \in \mathbb{N}$, one can construct an automaton B_α with $O(\alpha)$ states that accepts a word $w \in \Sigma^*$ if and only if its cost is at most α. Here, states are the numbers $\{0, \ldots, \alpha\} \cup \{q_\dagger\}$, 0 is the initial state, and reading a symbol $a \in \Sigma$ from a state i causes the automaton to transition to state $i + c(a)$ if this value is at most α or to the sink state $q_\dagger$ if the value is greater than α. Therefore, finding a synchronizing word of cost at most α for a DFA A may be modeled as finding a synchronizing for A that also belongs to $L(B_\alpha)$. In this context, diversity may be desirable because some constraints such as power consumption, machine size, space availability, etc. may be difficult to formalize as a regular constraint, and therefore, it would be good to allow the engineers developing the part orienter to choose among such. Using Theorem 18, the task of finding a set with r subsequence-minimal synchronizing words of cost at most α and that are k apart from each other can be solved in time $O(f_A(r, k) \cdot \alpha^r \log(\alpha))$.

9.2 Synchronizing Robots

Ronald L. Rivest and Robert E. Schapire considered in [72] the idea of modeling the map of an environment as a finite automaton. In their setting, the environment was originally unknown to the robot(s) who had to explore it and then build (learn) a map. Due to this scenario, the automata were in fact finite automata with outputs, and the paper did not consider synchronizing words but rather homing sequences, which can be viewed as a more general concept, but still analogous to synchronizing words, for automata with outputs.

Abstracting away these differences between automata with and without outputs, at a certain level of abstraction, a map can be viewed as a finite automaton, where the 'input letters' are interpreted as commands that move the robot, bringing it thus in a different state, i.e., a different position on the map. To make this scenario more concrete, think of robots that help visitors of a big museum building, say, by telling them information about the items in a particular room

(this room could be then viewed as a 'state' of the automaton). The robots can move and hence serve the visitors as personal assistants. This model abstracts from several difficulties, for instance, of the problem of finding out in which room the robot is. Notice that although nowadays many of these problems can be solved by GPS, this might not be the case in a big concrete building, as there could be problems with receiving the signals. See [26, 55] for other attempts on this problem. For simplicity, each room might have at most four doors, to the North, East, South and West, so that a word over the alphabet $\{N, E, S, W\}$ can be interpreted as a sequence of movements of a robot; if some door is not present, the robot simply stays in the room where it is. In the evening, all robots need to be put back to a particular room where they stay overnight and their batteries are recharged. Obviously, a synchronizing word could help here, as it means that only one particular sequence of commands need to be stored in the robots' memories, irrespectively of where they are. The overnight room of the robots would then be the synchronizing state.

However, there are practical complications to this approach: Even if all robots start their evening journey in different rooms, after executing, say, half of the synchronizing word, all the robots will be in only few of the rooms, so that many robots will be in the same room. The problem is now that of congestion: there will be traffic jams, slowing down the 'evening reset', because dozens of robots cannot pass through the same door at the same time. How can we avoid this problem that many of us know from rush hour scenarios in our cities? If not all robots would follow the same synchronizing word, then there are good chances that some of the mentioned traffic jams are avoided. Hence, we suggest that the robots may store a small diverse set of synchronizing words. In the evening, each robot will pick one of the synchronizing words at random and follow its commands. In this context, earlier studies on dealing with stochastic aspects of finite automata might be of interest, as they also sometimes deal with the sketched robot orientation application, see [23].

9.3 A Robot Coming Home

As explained in the previous section, maps could be stored as state spaces / finite automata. Then, a synchronizing word is a sequence of elementary commands that tell a robot how to get home, assuming he got lost, or maybe just because it is too difficult to store a word that leads home for each possible position. However, if a robot executes such a synchronizing sequence each evening, then it becomes quite predictable when it will be at what position, in particular concerning the last positions of the journey. This in turn makes the robot amenable to attacks, say, from adversaries who notice this 'regular behavior', so that they could set up a trap. In order to avoid this (type of analysis), the robot could store a few diverse synchronizing words and flip a coin before starting its journey home, hence (seemingly) selecting a different tour each evening. This way, it could create a certain level of security against ambush attacks.

In fact, the ideas presented in this and in the preceding section could be also of practical help in the following scenario that goes beyond problems concerning synchronizing words, helping with traffic jams originating from typical rush hour scenarios in our cities, where often enough people work in one part W of the city but live in another part L, so that they have to move from L to W and back each day. Usually, there is not just one optimal route, but there are a few possible routes. Nonetheless, most people do not change their habits, and also navigation systems typically stick with one decision (in that case, clearly influenced by the current traffic situation). But we all know from personal experience what happens if we only follow our habits or also, if we follow the advice given by navigation systems that try to avoid jams by looking at the current traffic situation: we will end up in (possibly different) traffic jams. Why does this happen if we follow the advice of our navigation system? Just because many drivers do

this at the same time, and all navigation systems follow similar if not the same algorithm to determine an alternative route, this will lead to a new traffic jam on our route. In the future, this might become even worse due to autonomous driving cars. In order to avoid this situation, one possible simiple solution would be to randomly select from a diverse set of routes. On average, this simple procedure should help balance the traffic over all possible routes between L and W.

9.4 Putting Several Agents into Synchrony

Consider the setting where we have a large number of identical copies of an agent, whose interaction with the environment is modeled by a deterministic finite automaton. Even though the agents are identical, their behavior is dictated by the environment, and therefore, distinct agents may be in distinct states at any given time, depending on how they have reacted to the environment. Suppose one wishes to put all agents into synchrony, by making them behave in the same way for a certain period of time, before they start to interact with the environment again. This could be, for instance, imposing that a swarm of very simple robots move in the same direction. This can be achieved by broadcasting a synchronizing word. Diversity is relevant in this case because distinct synchronizing words may correspond to distinct behavioural patterns to be followed by the automata.

Notice that this approach is distinct from the classical *Firing Squad Synchronization Problem* that is typically considered in one- or more-dimensional cellular automata, because the corresponding solution strategies heavily rely on a known interconnection pattern between the finite automata that need to be synchronized, which is not the case in the scenario that we described above. While the 1-dimensional case was first solved by John McCarthy and Marvin Minsky in the 1950s, there is quite a body of literature trying to improve on the running times or on the number of states of the finite automata involved. We only refer to [65, 77].

9.5 Molecular Computing Machines

An interesting application of the notion of synchronization occurs in the realm of molecular computing machines [10]. More specifically, in [9, 8], Benenson et al. introduced the notion of DNA and ATP based automata, which are automata operating in a molecular level. They were able to run experiments consisting of $3 \cdot 10^{12}$ automata per μl and performing $6.6 \cdot 10^{10}$ transitions per second per μl with transition fidelity 99.9%. Since distinct copies of the automaton process distinct molecules, the state of each such copy may depend on the actual molecule being read at the moment. In particular, the state of the automaton at the end of the process is unknown. In order to be reusable, molecular automata must be reset. This can be achieved by adding a mix of molecules encoding synchronizing words to the automata soup. This causes all automata in the soup to reboot. Even though one synchronizing word is enough to reset the system, we suggest to explore the use of a diverse set of synchronizing sequences to improve the reset operation in practice.

9.6 Games with incomplete information

In [63], Bastien Maubert and Sophie Pinchimat made some connection between game theory and automaton synchronization. In their work, they reduce the existence of a synchronizing word for a complete NFA to the existence of a winning strategy in a game with opacity condition (see [64] for the definition). This means that also our hardness results translate to hardness results in the sketched game-theoretic application. Even more, in the mentioned papers, it is also shown how to make use of such game-theoretic hardness results in security applications. Hence,

we suggest that also our hardness results could be interpreted as tools for proving security of certain protocols. As in this context, NFA are of special interest, we point again to Remark 20.

10 Conclusion

In this work, we have introduced a suitable notion of diversity of solutions in the context of the theory of synchronizing automata. Using this framework, we showed that for each $r, k \in \mathbb{N}$, each DFA A, and each non-deterministic finite automaton B over an alphabet Σ, the problem of computing a subset $\{w_1, \ldots, w_r\} \subseteq L(B)$ of subsequence-minimal synchronizing words for A, with pairwise edit distance of at least k, can be solved in time $O(f_A(r,k) \cdot |B|^r \log(|B|))$ for some suitable function f depending only on A, r and k. Note that our algorithm has a fixed-parameter tractable dependency on the parameters $|A|$ and k and an XP dependency on the parameter r. Therefore, for each fixed r, our algorithm runs in FPT time when parameterized by $|A|$ and k. We note that the existence of such FPT-algorithms was not clear even when the number r of solutions was fixed to 2. We leave the problem of determining whether one can obtain an algorithm of the form $O(f_A(r,k) \cdot |B|^{O(1)})$ as an interesting direction of further research. We have also shown that similar results hold in the context of conformant planning.

For some simplified versions of our problems, we proved NP- or coNP-hardness results. Then, it is conceivable that our main problems are hard for higher levels of the polynomial hierarchy. Here, we want to point to recent papers [15, 33] that study hardness and membership of combinatorial problems with some flavor of minimality, placing them into low levels of the said hierarchy.

We also described the impact of our results in different areas of artificial intelligence. We already mentioned conformant planning above, an area that finds quite direct applications of our results. But the concept of synchronization plays also an important role concerning some aspects of designing automatic production assembly lines when it comes to the deployment of part orienters. This application would also motivate to further study costs associated to input letters and to ask for minimum-cost synchronizing words or to incorporate costs into the diversity context. We also described two applications of (diverse sets of) synchronizing words in the context of robotics. These scenarios can be generalized and abstracted towards putting several agents into synchrony. Furthermore, we suggested to use (diverse sets of) synchronizing words for molecular computing machines. Finally, we pointed to the connections between game theory and automaton synchronization and suggested to study diversity in this context.

In the context of finite automata with outputs, the notion of *homing sequences* plays an analogous role to the notion of synchronization for finite automata without outputs. We suggest extending our research on diverse sets of synchronizing words to homing sequences and related notions, as they were coined already in the beginning of automata theory. We refer to [54, 56, 75] as a survey of these notions. Notice that homing sequences are well-motivated in particular in the context of robot navigation.

Acknowledgements

Mateus de Oliveira Oliveira acknowledges support from the Research Council of Norway (project numbers 288761 and 326537) and from the Sigma2 Network (project number NN9535K).

References

[1] Zeinab Abbassi, Vahab S. Mirrokni, and Mayur Thakur. Diversity maximization under matroid constraints. In Inderjit S. Dhillon, Yehuda Koren, Rayid Ghani, Ted E. Senator, Paul

Bradley, Rajesh Parekh, Jingrui He, Robert L. Grossman, and Ramasamy Uthurusamy, editors, *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD*, pages 32–40. ACM, 2013.

[2] Gediminas Adomavicius and YoungOk Kwon. Optimization-based approaches for maximizing aggregate recommendation diversity. *INFORMS Journal on Computing*, 26(2):351–369, 2014.

[3] Dimitry S. Ananichev and Mikhail V. Volkov. Synchronizing monotonic automata. *Theoretical Computer Science*, 327(3):225–239, 2004.

[4] Ariel S. Anders. *Reliably arranging objects: a conformant planning approach to robot manipulation.* PhD thesis, Massachusetts Institute of Technology, USA, 2019.

[5] Emmanuel Arrighi, Henning Fernau, Daniel Lokshtanov, Mateus de Oliveira Oliveira, and Petra Wolf. Diversity in Kemeny rank aggregation: A parameterized approach. In Zhi-Hua Zhou, editor, *International Joint Conference on Artificial Intelligence, 30th IJCAI*, pages 10–16. ijcai.org, 2021.

[6] Julien Baste, Michael R. Fellows, Lars Jaffke, Tomás Masarík, Mateus de Oliveira Oliveira, Geevarghese Philip, and Frances A. Rosamond. Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. In Christian Bessiere, editor, *International Joint Conference on Artificial Intelligence, 29th IJCAI*, pages 1119–1125. ijcai.org, 2020.

[7] Julien Baste, Lars Jaffke, Tomáš Masařík, Geevarghese Philip, and Günter Rote. FPT algorithms for diverse collections of hitting sets. *Algorithms*, 12:254, 2019.

[8] Yaakov Benenson, Rivka Adar, Tamar Paz-Elizur, Zvi Livneh, and Ehud Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proceedings of the National Academy of Sciences (USA)*, 100(5):2191–2196, 2003.

[9] Yaakov Benenson, Tamar Paz-Elizur, Rivka Adar, Ehud Keinan, Zvi Livneh, and Ehud Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414(6862):430–434, 2001.

[10] Yaakov Benenson and Ehud Shapiro. Molecular computing machines. *Encyclopedia of Nanoscience and Nanotechnology*, pages 2043–2056, 2004.

[11] Jean-Camille Birget. Partial orders on words, minimal elements of regular languages and state complexity. *Theoretical Computer Science*, 119(2):267–291, 1993.

[12] Blai Bonet. Conformant plans and beyond: Principles and complexity. *Artificial Intelligence*, 174(3-4):245–269, 2010.

[13] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM Journal on Computing*, 48(2):481–512, 2019.

[14] Horst Bunke and János Csirik. Parametric string edit distance and its application to pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(1):202–206, 1995.

[15] Elisabet Burjons, Fabian Frei, Edith Hemaspaandra, Dennis Komm, and David Wehner. Finding optimal solutions with neighborly help. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *International Symposium on Mathematical Foundations of Computer Science, 44th MFCS*, volume 138 of *LIPIcs*, pages 78:1–78:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[16] Airlie Chapman and Mehran Mesbahi. On symmetry and controllability of multi-agent systems. In *IEEE Conference on Decision and Control, 53rd CDC*, pages 625–630. IEEE, 2014.

[17] Pierre-Yves Chevalier, Julien M. Hendrickx, and Raphaël M. Jungers. Reachability of consensus and synchronizing automata. In *IEEE Conference on Decision and Control, 54th CDC*, pages 4139–4144. IEEE, 2015.

[18] Sook-Ling Chua, Stephen Marsland, and Hans W. Guesgen. Unsupervised learning of patterns in data streams using compression and edit distance. In Toby Walsh, editor, *International Joint Conference on Artificial Intelligence, 22nd IJCAI*, pages 1231–1236. IJCAI/AAAI, 2011.

[19] Alessandro Cimatti and Marco Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, 2000.

[20] Alessandro Cimatti, Marco Roveri, and Piergiorgio Bertoli. Conformant planning via symbolic model checking and heuristic search. *Artificial Intelligence*, 159(1-2):127–206, 2004.

[21] Emilie Danna and David L. Woodruff. How to select a small set of diverse solutions to mixed integer programming problems. *Operations Research Letters*, 37(4):255–260, 2009.

[22] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[23] Thomas L. Dean, Dana Angluin, Kenneth Basye, Sean P. Engelson, Leslie Pack Kaelbling, Evangelos Kokkevis, and Oded Maron. Inferring finite automata with stochastic output functions and an application to map learning. *Machine Learning*, 18(1):81–108, 1995.

[24] Emir Demirovic and Nicolas Schwind. Representative solutions for bi-objective optimisation. In *National Conference on Artificial Intelligence, 34th AAAI*, pages 1436–1443. AAAI Press, 2020.

[25] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 1999.

[26] Tom Duckett and Ulrich Nehmzow. Performance comparison of landmark recognition systems for navigating mobile robots. In Henry A. Kautz and Bruce W. Porter, editors, *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, pages 826–831. AAAI Press / The MIT Press, 2000.

[27] Thomas Eiter, Esra Erdem, Halit Erdogan, and Michael Fink. Finding similar/diverse solutions in answer set programming. *Theory and Practice of Logic Programming*, 13(3):303–359, 2013.

[28] David Eppstein. Reset Sequences for Monotonic Automata. *SIAM Journal on Computing*, 19(3):500–510, 1990.

[29] Hélène Fargier and Pierre Marquis. Disjunctive closures for knowledge compilation. *Artificial Intelligence*, 216:129–162, 2014.

[30] Henning Fernau, Vladimir V. Gusev, Stefan Hoffmann, Markus Holzer, Mikhail V. Volkov, and Petra Wolf. Computational complexity of synchronization under regular constraints. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *International Symposium on Mathematical Foundations of Computer Science, 44th MFCS*, volume 138 of *LIPIcs*, pages 63:1–63:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[31] Fedor V Fomin, Petr A Golovach, Lars Jaffke, Geevarghese Philip, and Danil Sagunov. Diverse pairs of matchings. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *International Symposium on Algorithms and Computation, 31st ISAAC*, volume 181 of *LIPIcs*, pages 26:1–26:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[32] Fedor V. Fomin, Pinar Heggernes, Dieter Kratsch, Charis Papadopoulos, and Yngve Villanger. Enumerating minimal subset feedback vertex sets. *Algorithmica*, 69(1):216–231, 2014.

[33] Fabian Frei, Edith Hemaspaandra, and Jörg Rothe. Complexity of stability. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *International Symposium on Algorithms and Computation, 31st ISAAC*, volume 181 of *LIPIcs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[34] Thomas Gabor, Lenz Belzner, Thomy Phan, and Kyrill Schmid. Preparing for the unexpected: Diversity improves planning resilience in evolutionary algorithms. In *15th International Conference on Autonomic Computing, ICAC*, pages 131–140. IEEE Computer Society, 2018.

[35] Zsolt Gazdag, Szabolcs Iván, and Judit Nagy-György. Improved upper bounds on synchronizing nondeterministic automata. *Information Processing Letters*, 109(17):986–990, 2009.

[36] Fred Glover, Arne Løkketangen, and David L. Woodruff. Scatter search to generate diverse MIP solutions. In *Computing Tools for Modeling, Optimization and Simulation*, pages 299–317. Springer, 2000.

[37] Robert P. Goldman and Mark S. Boddy. Expressive planning and explicit knowledge. In Brian Drabble, editor, *International Conference on Artificial Intelligence Planning Systems, 3rd AIPS*, pages 110–117. AAAI, 1996.

[38] Sreenivas Gollapudi and Aneesh Sharma. An axiomatic approach for result diversification. In Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl, editors, *Proceedings of the 18th International Conference on World Wide Web, WWW*, pages 381–390. ACM, 2009.

[39] Petr A. Golovach, Pinar Heggernes, Mamadou Moustapha Kanté, Dieter Kratsch, Sigve Hortemo Sæther, and Yngve Villanger. Output-polynomial enumeration on graphs of bounded (local) linear mim-width. *Algorithmica*, 80(2):714–741, 2018.

[40] Peter Greistorfer, Arne Løkketangen, Stefan Voß, and David L. Woodruff. Experiments concerning sequential versus simultaneous maximization of objective function and distance. *Journal of Heuristics*, 14(6):613–625, 2008.

[41] Herrmann Gruber, Markus Holzer, and Martin Kutrib. The size of Higman-Haines sets. *Theoretical Computer Science*, 387(2):167–176, 2007.

[42] Tarik Hadžić, Alan Holland, and Barry O'Sullivan. Reasoning about optimal collections of solutions. In *Principles and Practice of Constraint Programming, 15th CP*, volume 5732 of *LNCS*, pages 409–423. Springer, 2009.

[43] Leonard H. Haines. On free monoids partially ordered by embedding. *Journal of Combinatorial Theory*, 6(1):94–98, 1969.

[44] Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. Finding diverse and similar solutions in constraint programming. In Manuela M. Veloso and Subbarao Kambhampati, editors, *National Conference on Artificial Intelligence, 20th AAAI*, pages 372–377. AAAI Press / The MIT Press, 2005.

[45] Emmanuel Hebrard, Barry O'Sullivan, and Toby Walsh. Distance constraints in constraint satisfaction. In Manuela M. Veloso, editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, pages 106–111, 2007.

[46] F. C. Hennie. Fault detecting experiments for sequential circuits. In *5th Annual Symposium on Switching Circuit Theory and Logical Design, SWCT (FOCS)*, pages 95–110. IEEE Computer Society, 1964.

[47] Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society, Series 3*, 2(7):326–336, 1952.

[48] Stefan Hoffmann. Computational complexity of synchronization under regular commutative constraints. In Donghyun Kim, R. N. Uma, Zhipeng Cai, and Dong Hoon Lee, editors, *Computing and Combinatorics - 26th International Conference, COCOON*, volume 12273 of *LNCS*, pages 460–471. Springer, 2020.

[49] Stefan Hoffmann. Completely reachable automata, primitive groups and the state complexity of the set of synchronizing words. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications - 15th International Conference, LATA*, volume 12638 of *LNCS*, pages 305–317. Springer, 2021.

[50] Balázs Imreh and Magnus Steinby. Directable nondeterministic automata. *Acta Cybernetica*, 14(1):105–115, 1999.

[51] Linnea Ingmar, Maria Garcia de la Banda, Peter J. Stuckey, and Guido Tack. Modelling diversity of solutions. In *National Conference on Artificial Intelligence, 34th AAAI*, pages 1528–1535. AAAI Press, 2020.

[52] Mamadou Moustapha Kanté, Vincent Limouzy, Arnaud Mary, and Lhouari Nourine. On the enumeration of minimal dominating sets and related notions. *SIAM Journal on Discrete Mathematics*, 28(4):1916–1929, 2014.

[53] Mamadou Moustapha Kanté and Lhouari Nourine. Minimal dominating set enumeration. In *Encyclopedia of Algorithms*, pages 1287–1291. Springer, 2016.

[54] Zvi Kohavi and Niraj K. Jha. *Switching and Finite Automata Theory*. Cambridge University Press, 3rd edition, 2009.

[55] Benjamin Kuipers and Patrick Beeson. Bootstrap learning for place recognition. In Rina Dechter, Michael J. Kearns, and Richard S. Sutton, editors, *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence*, pages 174–180. AAAI Press / The MIT Press, 2002.

[56] David Lee and Mihalis Yannakakis. Testing finite-state machines: State identification and verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994.

[57] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, pages 707–710, 1966.

[58] Arne Løkketangen and David L. Woodruff. A distance function to support optimized selection decisions. *Decision Support Systems*, 39(3):345–354, 2005.

[59] Dimitrios P. Lyras, Kyriakos N. Sgarbas, and Nikolaos D. Fakotakis. Using the Levenshtein edit distance for automatic lemmatization: A case study for modern Greek and English. In *IEEE International Conference on Tools with Artificial Intelligence, 19th ICTAI*, volume 2, pages 428–435. IEEE Computer Society, 2007.

[60] Pierre Marquis. Existential closures for knowledge compilation. In Toby Walsh, editor, *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI*, pages 996–1001. IJCAI/AAAI, 2011.

[61] Pavel V. Martyugin. Synchronization of automata with one undefined or ambiguous transition. In Nelma Moreira and Rogério Reis, editors, *Implementation and Application of Automata - 17th International Conference, CIAA 2012, Porto, Portugal, July 17-20, 2012. Proceedings*, volume 7381 of *Lecture Notes in Computer Science*, pages 278–288. Springer, 2012.

[62] Pavel V. Martyugin. Computational complexity of certain problems related to carefully synchronizing words for partial automata and directing words for nondeterministic automata. *Theory of Computing Systems*, 54(2):293–304, 2014.

[63] Bastien Maubert. *Synchronizing automata and their applications to games with imperfect information.* Research Master's Thesis, Computer Science, Université Rennes, France, 2009.

[64] Bastien Maubert and Sophie Pinchinat. Games with opacity condition. In Olivier Bournez and Igor Potapov, editors, *Reachability Problems, 3rd International Workshop, RP*, volume 5797 of *LNCS*, pages 166–175. Springer, 2009.

[65] Jacques Mazoyer. A six-state minimal time solution to the firing squad synchronization problem. *Theoretical Computer Science*, 50(2):183–238, 1987.

[66] Alexander Nadel. Generating diverse solutions in SAT. In Karem A. Sakallah and Laurent Simon, editors, *Theory and Applications of Satisfiability Testing, 14th SAT*, volume 6695 of *LNCS*, pages 287–301. Springer, 2011.

[67] B. K. Natarajan. An algorithmic approach to the automated design of parts orienters. In *Annual Symposium on Foundations of Computer Science, 27th FOCS*, pages 132–142. IEEE Computer Society, 1986.

[68] Hector Palacios and Hector Geffner. Compiling uncertainty away in conformant planning problems with bounded width. *Journal of Artificial Intelligence Research*, 35:623–675, 2009.

[69] Thierry Petit and Andrew C. Trapp. Finding diverse solutions of high quality to constraint optimization problems. In Qiang Yang and Michael J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI*, pages 260–267. AAAI Press, 2015.

[70] Thierry Petit and Andrew C. Trapp. Enriching solutions to combinatorial problems via solution engineering. *INFORMS Journal on Computing*, 31(3):429–444, 2019.

[71] Amirreza Rahmani, Meng Ji, Mehran Mesbahi, and Magnus Egerstedt. Controllability of multi-agent systems from a graph-theoretic perspective. *SIAM Journal on Control and Optimization*, 48(1):162–186, 2009.

[72] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences (extended abstract). In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, STOC*, pages 411–420. ACM, 1989.

[73] Igor K. Rystsov. On minimizing the length of synchronizing words for finite automata. In *Theory of Designing of Computing Systems*, pages 75–82. Institute of Cybernetics of the Ukrainian Academy of Science, 1980. (in Russian).

[74] Igor K. Rystsov. Polynomial Complete Problems in Automata Theory. *Information Processing Letters*, 16(3):147–151, 1983.

[75] Sven Sandberg. Homing and synchronizing sequences. In Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors, *Model-Based Testing of Reactive Systems, Advanced Lectures*, volume 3472 of *LNCS*, pages 5–33. Springer, 2005.

[76] Patrick Schittekat and Kenneth Sörensen. OR practice — supporting 3PL decisions in the automotive industry by generating diverse solutions to a large-scale location-routing problem. *Operations Research*, 57(5):1058–1067, 2009.

[77] Ilka Shinahr. Two- and three-dimensional firing-squad synchronization problems. *Information and Control*, 24(2):163–180, 1974.

[78] David E. Smith and Daniel S. Weld. Conformant graphplan. In Jack Mostow and Chuck Rich, editors, *National Conference on Artificial Intelligence and Innovative Applications of Artificial Intelligence Conference, 15th AAAI & 10th IAAI*, pages 889–896. AAAI Press / The MIT Press, 1998.

[79] Peter H. Starke. Eine bemerkung über homogene Experimente. *Elektronische Informationsverarbeitung und Kybernetik*, 2(4):257–259, 1966.

[80] Peter H. Starke. A remark about homogeneous experiments. *Journal of Automata, Languages and Combinatorics*, 24(2-4):133–137, 2019.

[81] Yann Strozecki. Enumeration complexity. *Bulletin of the EATCS*, 129, 2019.

[82] Andrew C. Trapp and Renata A. Konrad. Finding diverse optima and near-optima to binary integer programs. *IIE Transactions*, 47(11):1300–1312, 2015.

[83] Uraz Cengiz Türker and Hüsnü Yenigün. Complexities of some problems related to synchronizing, non-synchronizing and monotonic automata. *International Journal on Foundations of Computer Science*, 26(1):99–122, 2015.

[84] Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, 2nd International Conference, LATA*, volume 5196 of *LNCS*, pages 11–27. Springer, 2008.

[85] Robert A. Wagner. On the complexity of the extended string-to-string correction problem. In William C. Rounds, Nancy Martin, Jack W. Carlyle, and Michael A. Harrison, editors, *Annual ACM Symposium on Theory of Computing, 7th STOC*, pages 218–223. ACM, 1975.

[86] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.

[87] Mark Wineberg and Franz Oppacher. The underlying similarity of diversity measures used in evolutionary computation. In Erick Cantú-Paz, James A. Foster, Kalyanmoy Deb, Lawrence Davis, Rajkumar Roy, Una-May O'Reilly, Hans-Georg Beyer, Russell K. Standish, Graham Kendall, Stewart W. Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitchell A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasa Jonoska, and Julian F. Miller, editors, *Genetic and Evolutionary Computation Conference, GECCO '03, Part II*, volume 2724 of *LNCS*, pages 1493–1504. Springer, 2003.

[88] Petra Wolf. Synchronization under dynamic constraints. In Nitin Saxena and Sunil Simon, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, 40th FSTTCS*, volume 182 of *LIPIcs*, pages 58:1–58:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.