Institute of Science and Technology

Edit distance for timed automata

Krishnendu Chatterjee, Rasmus Ibsen-Jensen, Rupak Majumdar

https://doi.org/10.15479/AT:IST-2013-144-v1-1

Deposited at: 12 Dec 2018 11:53 ISSN: 2664-1690

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

AUSTRIA

Institute of Science and Technology

Edit Distance for Timed Automata

Krishnendu Chatterjee and Rasmus Ibsen-Jensen and Rupak Majumdar

Technical Report No. IST-2013-144-v1+1
Deposited at UNSPECIFIED
http://repository.ist.ac.at/144/1/main.pdf

IST Austria (Institute of Science and Technology Austria)
Am Campus 1
A-3400 Klosterneuburg, Austria

Edit Distance for Timed Automata

Krishnendu Chatterjee
IST Austria

Rasmus Ibsen-Jensen
IST Austria

Rupak Majumdar
MPI-SWS

ABSTRACT

The edit distance between two (untimed) traces is the minimum cost of a sequence of edit operations (insertion, deletion, or substitution) needed to transform one trace to the other. Edit distances have been extensively studied in the untimed setting, and form the basis for approximate matching of sequences in different domains such as coding theory, parsing, and speech recognition.

In this paper, we lift the study of edit distances from untimed languages to the timed setting. We define an edit distance between timed words which incorporates both the edit distance between the untimed words and the absolute difference in time stamps. Our edit distance between two timed words is computable in polynomial time. Further, we show that the edit distance between a timed word and a timed language generated by a timed automaton, defined as the edit distance between the word and the closest word in the language, is PSPACE-complete. While computing the edit distance between two timed automata is undecidable, we show that the approximate version, where we decide if the edit distance between two timed automata is either less than a given parameter or more than δ away from the parameter, for $\delta > 0$, can be solved in exponential space and is EXPSPACE-hard. Our definitions and techniques can be generalized to the setting of hybrid systems, and analogous decidability results hold for rectangular automata.

Keywords. *Timed automata; Edit distance; Rectangular hybrid automata.*

1. INTRODUCTION

The edit distance [14] between two strings is the minimum cost of a sequence of edit operations (insertion, deletion, or substitution of one letter by another) that transforms one string to another. The edit distance between a string w and a language L is the minimal distance between strings belonging to L and w. The notion of *edit distance* provides a quantitative measure of "how far" one string is from another, or from a given language. It forms the basis for approximately comparing sequences, a problem that arises in many different areas, such as error-correcting codes, natural language processing, and computational biology.

Algorithms for edit distance have been studied extensively for (untimed) words [14, 1, 16, 18, 13, 15]. In this paper, we generalize the definition of edit distance from untimed to timed words. We define the edit distance between two timed words tw and tw' as the lexico-graphic ordering of two components: the first is the (normal) edit distance on their untimed parts, and the second is the maximum difference in time stamps.[1] We study algorithmic aspects of the edit distance between timed words and timed languages. We show that the edit distance between two timed words can be computed in polynomial time. Moreover, we show that the edit distance between a timed word and a timed language generated by a timed automaton can be computed in polynomial space. The corresponding decision problem is PSPACE-complete. A nice by-product of our result is that the edit distance problem for an untime word and untimed non-deterministic finite-state automata (NFA) is NL-complete (complete for non-deterministic log-space).

One can generalize edit distances to capture the distance between two languages: the edit distance between L_1 and L_2 is the supremum over all strings w in L_1 of the edit distance between w and L_2. We show that the edit distance between two timed languages generated by timed automata is not computable. However, we show that the approximate version of the problem, where we ask if the edit distance is either less than α or more than $\alpha+\delta$ for an additive error $\delta > 0$, can be solved in exponential space, and is EXPSPACE-hard.

Our results use the following technical constructions. For the computation of edit distance between a timed word and a timed automaton, we construct two timed automata which are polynomial in the size of the input automaton, and show that the decision problem for edit distance reduces to checking non-emptiness of the constructed automata. The key intuition is to use non-determinism in the timed automata to model edits in the word, and use additional clocks with rectangular constraints to bound the mismatch in time stamps. For the computation of the approximation of edit distance between two timed automata, we generalize the approach for computation of edit distance between two untimed automata [4]. The algorithm uses the classical region abstraction, but requires non-trivial generalization of the untimed case [4] to capture the quantitative timing aspects.

Besides intellectual curiosity, our definition and algorithmic computation of edit distances between timed words and timed languages form the foundations of a quantitative approach to timed verification. The calculation of timed edit distance is the basis for *repairing* timed specifications, generalizing the untimed case [4], and for providing robust semantics to timed automata and timed logics [9, 8]. For example, in simulation-based verification of a real-time implementation against a timed automaton model, the simulation trace may differ slightly from the model due to inac-

[1] While we focus on this definition, we show that several related definitions have similar algorithmic properties.

HSCC'14.

curacies in the implementation and errors in measuring the timing behavior. Thus, a timed trace of the implementation may not be in the model. However, instead of rejecting the implementation, one can quantify the distance between a measured trace and the model. Quantitative semantics for timed and hybrid logics have been the basis for some recent verification tools [10, 7]. Our work can be seen as providing a quantitative semantics for timed automaton models.

Finally, while we focus on timed systems, we sketch how our definitions and algorithmic techniques extend to hybrid automata, with EXPTIME algorithms for the edit distance between a hybrid trace and a rectangular hybrid automaton, and 2EXPTIME algorithm for the approximate distance between two rectangular automata.

2. DEFINITIONS

In this section we first present the basic definition of timed automata, and then the notion of edit distance for them.

2.1 Timed Automata

Timed automata [2] suggest a finite syntax for specifying finite-state automata with real-valued clocks. We first start with the notion of clock constraints.

Clock constraints. For a set X of clock variables, the set $\Phi(X)$ of *clock constraints* ψ is defined inductively by

$$\psi \ := \ x \leq d \mid d \leq x \mid \neg\psi \mid \psi_1 \wedge \psi_2,$$

where x is a clock in X and d is a constant in natural numbers.

Timed automata. A *timed automaton* $\mathcal{A}$ over finite words is a tuple $\langle L, \Sigma, C, \rightarrow, \gamma, S_0, F\rangle$, where

- L is a finite set of locations.
- Σ is a finite set of input alphabet.
- C is a finite set of clocks.
- $\rightarrow \subseteq \ L \times L \times \Sigma \times 2^C \times \Phi(C)$ gives the set of transitions, where $\Phi(C)$ is the set of clock constraints over C. An edge $(\ell, \ell', \sigma, \lambda, \psi)$ represents a transition from location ℓ to location ℓ' on input letter σ, $\lambda \subseteq C$ represents the set of clocks to be reset with the transition and ψ is a clock constraint over C.
- $\gamma : L \mapsto \mathsf{Constr}(C)$ is a function that assigns to every location an invariant on clock valuations. All clocks increase uniformly at the same rate. When at location ℓ, a valid execution must move out of ℓ before the invariant $\gamma(\ell)$ expires. Thus, the timed automaton can stay at a location only as long as the invariant is satisfied by the clock values.
- $S_0 \subseteq L \times \mathbb{R}_+^{|C|}$ is the set of initial states.
- $F \subseteq L$ is a finite set of accepting locations.

Each clock increases at rate 1 inside a location. A *clock valuation* is a function $\kappa : C \mapsto \mathbb{R}_{\geq 0}$ that maps every clock to a non-negative real. The set of all clock valuations for C is denoted by $K(C)$. Given a clock valuation $\kappa \in K(C)$ and a time delay $\Delta \in \mathbb{R}_{\geq 0}$, we write $\kappa + \Delta$ for the clock valuation in $K(C)$ defined by $(\kappa + \Delta)(x) = \kappa(x) + \Delta$ for all clocks $x \in C$. For a subset $\lambda \subseteq C$ of the clocks, we write $\kappa[\lambda := 0]$ for the clock valuation in $K(C)$ defined by $(\kappa[\lambda := 0])(x) = 0$ if $x \in \lambda$, and $(\kappa[\lambda := 0])(x) = \kappa(x)$ if $x \notin \lambda$. A clock valuation $\kappa \in K(C)$ *satisfies* the clock constraint θ, written $\kappa \models \theta$, if the condition θ holds when all clocks in C take on the values specified by κ. A *state* $s = \langle \ell, \kappa \rangle$ of the timed automaton $\mathcal{A}$ is a location $\ell \in L$ together with a clock valuation $\kappa \in K(C)$ such that the invariant at the location is satisfied, that is, $\kappa \models \gamma(\ell)$. We let S be the set of all states of $\mathcal{A}$. The semantics of timed automata are given as timed transition systems, which is standard [2], and omitted here.

Timed and untimed words. An *untimed word* $w \in \Sigma^*$ is a finite sequence of input letters, and a *timed word* $\mathrm{tw} \in (\Sigma \times \mathbb{R})^*$ is a finite sequence of input letters and time stamps such that the time stamps are non-decreasing. Equivalently a timed word $\mathrm{tw} = (w, \overline{t})$ can be considered as a pair of sequences, where the first sequence $w = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is the sequence of letters (i.e., the untimed word corresponding to tw), and the second sequence is the corresponding time stamps $\overline{t} = (t_1, t_2, \ldots, t_n)$, and we require that for all $1 \leq i \leq n-1$ we have $t_i \leq t_{i+1}$. The *length* of a timed word tw is the number of letters in it, i.e., the length of the untimed word.

Language of timed automata. A timed word tw induces a set of runs over a timed automata (see [2] for the standard semantics of runs). A word tw is accepted by an automata $\mathcal{A}$ if there exists a run that ends in an accepting location. For a timed automaton $\mathcal{A}$ we denote by $\mathcal{L}(\mathcal{A})$ the set of timed words accepted by $\mathcal{A}$.

Clock region equivalence. Clock region equivalence, denoted as $\cong$ is an equivalence relation on states of timed automata. The equivalence classes of the relation are called *regions*, and induce a time abstract bisimulation on the corresponding timed transition system [2]. There are finitely many clock regions; more precisely, the number of clock regions is bounded by $|L| \cdot \prod_{x \in C}(c_x + 1) \cdot |C|! \cdot 4^{|C|}$. For a real $t \geq 0$, let $\mathsf{frac}(t) = t - \lfloor t \rfloor$ denote the fractional part of t. Given a timed automaton $\mathcal{A}$, for each clock $x \in C$, let c_x denote the largest integer constant that appears in any clock constraint involving x in $\mathcal{A}$ (let $c_x = 1$ if there is no clock constraint involving x). Two states $\langle \ell_1, \kappa_1 \rangle$ and $\langle \ell_2, \kappa_2 \rangle$ are said to be *region equivalent* if all the following conditions are satisfied: (a) $\ell_1 = \ell_2$, (b) for all clocks x, we have $\kappa_1(x) \leq c_x$ iff $\kappa_2(x) \leq c_x$, (c) for all clocks x with $\kappa_1(x) \leq c_x$, we have $\lfloor \kappa_1(x) \rfloor = \lfloor \kappa_2(x) \rfloor$, (d) for all clocks x, y with $\kappa_1(x) \leq c_x$ and $\kappa_1(y) \leq c_y$, we have $\mathsf{frac}(\kappa_1(x)) \leq \mathsf{frac}(\kappa_1(y))$ iff $\mathsf{frac}(\kappa_2(x)) \leq \mathsf{frac}(\kappa_2(y))$, and (e) for all clocks x with $\kappa_1(x) \leq c_x$, we have $\mathsf{frac}(\kappa_1(x)) = 0$ iff $\mathsf{frac}(\kappa_2(x)) = 0$. Given a state $\langle \ell, \kappa \rangle$ of $\mathcal{A}$, we denote the region containing $\langle \ell, \kappa \rangle$ as $\mathsf{Reg}(\langle \ell, \kappa \rangle)$.

Region graph. The region graph $\mathsf{Reg}(\mathcal{A})$ corresponding to a timed automata $\mathcal{A}$ is the time-abstract bisimulation quotient graph induced by the region equivalence relation. The states of $\mathsf{Reg}(\mathcal{A})$ are the regions of $\mathcal{A}$. In the region graph, for regions R and R', there exists a transition $R \rightarrow R'$ iff there exists $s \in R$ and $s' \in R'$ such that there exists a transition from s to s' in the timed automata. We denote by $|\mathsf{Reg}(\mathcal{A})|$ the number of states in the region graph, which is bounded by $|C|! \cdot 4^{|C|} \cdot (c_{\max} + 1)^{|C|} \cdot |L|$, where C is the set of clocks, $c_{\max}$ the largest constant in the clock constraints, and $|L|$ is the number of locations.

2.2 Edit distance

In this section we first recall the notion of edit distance for untimed words, and then introduce the definition of edit distance for timed words. Finally we present the definition of edit distance between a timed word and a timed automaton, and between two timed automata.

Edit distance between untimed words. Consider a pair of untimed words w and w'. A *word edit* WE from w to w' is a finite sequence of some deletions, substitutions, and insertions of letters into w such that the sequence of transformations changes

w to w'. We denote by $\mathsf{WE}(w, w')$ the set of word edits from w to w', and $\mathrm{Opt}(w, w')$ be the set of optimal word edits between w and w', i.e., $\mathrm{Opt}(w, w')$ is the subset of $\mathsf{WE}(w, w')$ such that every sequence in $\mathrm{Opt}(w, w')$ has the minimal length among the sequences in $\mathsf{WE}(w, w')$. The *edit-distance* $\mathcal{D}(w, w')$ is the minimum number of edits required to transform w to w', i.e., the length of a sequence in $\mathrm{Opt}(w, w')$. A word edit WE is *optimal* if it belongs to $\mathrm{Opt}(w, w')$. Given a word edit WE, we say that the i-th index of w is *retained* if the i-th letter w_i was not deleted by the deletions of WE nor substituted by the substitutions of WE. Also, we say that the i-th index of w *corresponds* to the j-th index of w' if i was retained and there was $j - i$ insertions minus deletions in WE before the i-th index. Note that if index i is retained, there is always some j such that i corresponds to j. Also note that for any index j, there is at most one index i such that index i corresponds to index j.

Example. Informally, the edit distance between two timed words is a pair, where the first component is the edit distance between the untimed words, and the second component is the absolute maximal time mismatch. We illustrate with some examples the definition for edit distance between timed words. First consider two timed words where the untimed parts match, i.e., $\mathrm{tw} = (w, \overline{t})$ and $\mathrm{tw}' = (w, \overline{t}')$. Then the first component of the edit distance is 0 and the second component is the absolute maximal mismatch in the timing. Now, consider two timed words $\mathrm{tw} = (w, \overline{t})$, where $w = abcd$ and $\overline{t} = (1, 2, 3, 4)$, and $\mathrm{tw}' = (w', \overline{t}')$ where $w' = abbcd$ and $\overline{t}' = (1, 2, 2, 4, 4)$. We first extend the timed word tw to a timed word $\mathrm{tw}'' = (w'', \overline{t}'')$ such that $w'' = w'$ and the time sequences in $\overline{t}''$ matches the ones of $\overline{t}$ for the occurrences that match in w and w''. For example, an extension of tw is $w'' = abbcd$ and $\overline{t}'' = (1, 2, 2, 3, 4)$. Thus the first component of the edit distance is 1, and the second component is also 1.

Extension of timed words. Given a pair of timed words $\mathrm{tw} = (w, \overline{t})$ and $\mathrm{tw}' = (w', \overline{t}')$, we first consider the corresponding untimed words w and w'. Given a word edit WE between w and w', the timed word tw can be *extended* to tw' by WE if for each pair of indices i, j, such that index i of w corresponds to index j of w' under WE, we have that $t_i = t'_j$. In other words, the word edit creates a word whose untimed word matches with w' and the time stamps corresponding to the letters in w match with the time stamps in w. Given a timed word tw, a word w', and a word edit WE between w and w', let $\mathrm{Ext}(\mathrm{tw}, w', \mathsf{WE})$ be the set of timed words tw' such that tw can be extended to tw' by WE.

Edit distance between timed words. Let $(a_1, b_1) \in \mathbb{R}^2$ and $(a_2, b_2) \in \mathbb{R}^2$ be two pairs of real numbers, then the *lexico-graphic* ordering $\leq_{\mathsf{lex}}$ and $<_{\mathsf{lex}}$ is defined as follows:

$$(a_1, b_1) \leq_{\mathsf{lex}} (a_2, b_2) \quad \text{iff} \quad (a_1 < a_2) \vee (a_1 = a_2 \wedge b_1 \leq b_2);$$

$$(a_1, b_1) <_{\mathsf{lex}} (a_2, b_2) \quad \text{iff} \quad (a_1 < a_2) \vee (a_1 = a_2 \wedge b_1 < b_2);$$

and we use similar notations for $\geq_{\mathsf{lex}}$ and $>_{\mathsf{lex}}$. The edit distance for timed words has two components, the first component is the number of edits for the untimed word, and the second component is the maximal mismatch in the time stamps. We consider edit distance between timed words where we consider the lexico-graphic ordering of the two components, i.e., edits to discrete transitions are more costly. Formally, the edit distance $\mathcal{D}(\mathrm{tw}, \mathrm{tw}')$ between two timed words is defined as follows, where $\mathcal{D}_1(\mathrm{tw}, \mathrm{tw}')$ and $\mathcal{D}_2(\mathrm{tw}, \mathrm{tw}')$ are the first and second component, respectively:

1. For a pair of timed words $\mathrm{tw} = (w, \overline{t})$ and $\mathrm{tw}' = (w', \overline{t}')$ of length n, such that $w = w'$, the first component of the edit distance is 0 and the second component $\mathcal{D}_2(\mathrm{tw}, \mathrm{tw}')$ is defined as follows:

$$\mathcal{D}_2(\mathrm{tw}, \mathrm{tw}') = \max_{1 \leq i \leq n} |t_i - t'_i| \ .$$

2. For a pair of timed words $\mathrm{tw} = (w, \overline{t})$ and $\mathrm{tw}' = (w', \overline{t}')$ such that $w \neq w'$ we have $\mathcal{D}_1(\mathrm{tw}, \mathrm{tw}') = \mathcal{D}(w, w')$, i.e., the first component is the edit distance of the untimed words. For the second component we first consider the extension of tw and then compute the second component. Formally,

$$\mathcal{D}_2(\mathrm{tw}, \mathrm{tw}') = \inf_{\substack{\mathsf{WE} \in \mathrm{Opt}(w, w') \text{ and }, \\ \mathrm{tw}'' \in \mathrm{Ext}(\mathrm{tw}, w', \mathsf{WE})}} \mathcal{D}_2(\mathrm{tw}'', \mathrm{tw}') \ .$$

Note that above we have that the untimed part of tw'' and tw' coincide and hence we apply the definition of the first item above where the untimed parts coincide. Intuitively, we first pick some optimal word edit for the untimed word, and then extend the first word under this word edit, and then compute the second component. Finally, among all the choices we consider the one that minimizes the second component.

PROPOSITION 1 (COMPUTATION OF EDIT DISTANCE). *Given two timed words* tw *and* tw' *the edit distance* $\mathcal{D}(\mathrm{tw}, \mathrm{tw}')$ *can be computed in polynomial time.*

PROOF. It is straightforward to find the edit distance between two timed words $\mathrm{tw}, \mathrm{tw}'$ in polynomial time, and we describe the main ideas below. The first component is computed simply running the classical dynamic programming algorithm of [20] on the untimed words. Given a bound β on the second component the standard dynamic programming algorithm of [20] is modified to ensure that for all i, j, the i-th character of tw matches the j-th character of tw' iff they use the same letter and the difference between the time stamps is at most β. It is also clear that there are at most $|\,\mathrm{tw}\,| \cdot |\,\mathrm{tw}'\,|$ different "possible" values for β: the difference between each pair of time stamps (except in the case where no letter match, in which case the value of β is 0). By simply using a binary search algorithm over the possible choices, we get an algorithm with a running time of $O(|\,\mathrm{tw}\,| \cdot |\,\mathrm{tw}'\,| \cdot \log(|\,\mathrm{tw}\,| \cdot |\,\mathrm{tw}'\,|))$. □

Edit distance of timed words and timed automata, and between pairs of timed automata. Consider a pair of timed automata $\mathcal{A}$ and $\mathcal{A}'$, and a timed word tw. The edit distance between the pairs, and between the timed word and an automaton is defined as follows:

1. For the timed word tw and the timed automaton $\mathcal{A}$, the edit distance $\mathcal{D}(\mathrm{tw}, \mathcal{A})$ is $\inf_{\mathrm{tw}' \in \mathcal{L}(\mathcal{A})} \mathcal{D}(\mathrm{tw}, \mathrm{tw}')$, i.e., the edit distance is the minimal edit distance among all words accepted by the automaton $\mathcal{A}$. Also note that we consider the lexico-graphic ordering to compare the edit distance which consists of a pair of numbers.

2. For the pair of timed automata $\mathcal{A}, \mathcal{A}'$, the edit distance $\mathcal{D}(\mathcal{A}, \mathcal{A}')$ is $\sup_{\mathrm{tw} \in \mathcal{L}(\mathcal{A})} \mathcal{D}(\mathrm{tw}, \mathcal{A}')$, i.e., it is the maximal edit distance between a word in the language of $\mathcal{A}$ to the automaton $\mathcal{A}'$.

3. EDIT DISTANCE BETWEEN A TIMED WORD AND A TIMED AUTOMATON

In this section we consider the edit distance problem between a timed word and a timed automaton. We show that the problem is PSPACE-complete. We first define the decision problem and start with the lower bound.

Decision problem for edit distance between a timed word and a timed automaton. The *edit-distance decision* problem $\mathsf{EdDec}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$ is as follows: given a non-negative integer α, a number $\beta \in \mathbb{Q} \cup \{\infty\}$, where $\mathbb{Q}$ is the set of rationals, a timed word tw, and a timed automaton $\mathcal{A}$, the decision problem asks whether the edit distance $\mathcal{D}(\mathrm{tw}, \mathcal{A}) \leq_{\mathsf{lex}} (\alpha, \beta)$? In the sequel we always consider $\alpha \in \mathbb{N}$ and $\beta \in \mathbb{Q} \cup \{\infty\}$ such that β is non-negative.

LEMMA 2 (PSPACE LOWER BOUND). *The edit-distance decision problem* $\mathsf{EdDec}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$ *is PSPACE-hard.*

PROOF. Since the reachability problem for timed automata is PSPACE-hard [2], it follows that the non-emptiness question for timed automata (i.e., given a timed automaton $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A})$ is non-empty) is also PSPACE-hard. If the language $\mathcal{L}(\mathcal{A})$ for a timed automaton $\mathcal{A}$ is non-empty, then it accepts a timed word of length say at most d^* (d^* is at most exponential in the size of $\mathcal{A}$ and linear in the size of the region graph). Then the answer to the question $\mathsf{EdDec}(d^* + 1, 0, \epsilon, \mathcal{A})$ is YES iff $\mathcal{L}(\mathcal{A})$ is non-empty, where ϵ is the empty word. The PSPACE lower bound follows. □

PSPACE upper bound. The rest of the section is devoted to presenting a PSPACE upper bound for the edit distance decision problem $\mathsf{EdDec}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$.

Bound on the components of the edit distance. We start with a bound of the first component of edit distance.

1. *(Bound on first component).* For a given timed automaton $\mathcal{A}$, if $\mathcal{L}(\mathcal{A})$ is not empty, then as mentioned above $\mathcal{A}$ accepts a word of length at most exponential in the size of the automaton (at most the size of the region graph). Hence the first component of the edit distance between a timed word tw and a timed automaton $\mathcal{A}$ is at most $\max\{|\,\mathrm{tw}\,|, d\}$, where $|\,\mathrm{tw}\,|$ is the length of the timed word and d the length of the shortest word in $\mathcal{L}(\mathcal{A})$.

2. *(Bound on second component).* If the first component is bounded by α, then the second component can be at most $\max(|\,\mathrm{tw}\,| + \alpha) \cdot c_{\max}, t_{|\,\mathrm{tw}\,|}$, where $c_{\max}$ is the greatest number appearing in a clock constraint and $t_{|\,\mathrm{tw}\,|}$ is the last time stamp in tw. This is because any run in $\mathcal{A}$ that ensures that the first component is at most α cannot be longer than $(|\,\mathrm{tw}\,|+\alpha)$ and we can bound the wait in each move by $c_{\max}$.

PSPACE algorithm. We now give an algorithm which solves the decision problem $\mathsf{EdDec}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$ in polynomial space. We refer to our algorithm as $\textsc{SolEd}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$. Given $\alpha, \beta, \mathrm{tw}$, and $\mathcal{A}$, we construct two timed automata $\mathcal{A}'$ and $\mathcal{A}''$ and return NO iff $\mathcal{L}(\mathcal{A}')$ and $\mathcal{L}(\mathcal{A}'')$ are both empty, i.e., if either of the automata has a non-empty language, then the answer to the edit-distance decision problem is YES. The construction of $\mathcal{A}''$ given $\alpha, \beta, \mathrm{tw}$, and $\mathcal{A}$, is the same as the construction of $\mathcal{A}'$ given $\alpha - 1, \infty, \mathrm{tw}$ and $\mathcal{A}$, and thus we only explicitly give the construction of $\mathcal{A}'$.

Construction of $\mathcal{A}'$ given $\alpha, \beta, \mathrm{tw}$, and $\mathcal{A}$. The construction of $\mathcal{A}'$ given $\alpha, \beta, \mathrm{tw}$, and $\mathcal{A}$, is as follows:

1. *(Locations).* The timed automaton $\mathcal{A}'$ contains $(|\,\mathrm{tw}\,| + 1) \cdot (\alpha + 1)$ copies of $\mathcal{A}$, each location in each copy is annotated with a pair of integers (j, k), where $0 \leq j \leq |\,\mathrm{tw}\,|$ and $0 \leq k \leq \alpha$, where j corresponds to how far the timed word tw has been processed, and k to the number of edits that have been made. The location corresponding to location ℓ in $\mathcal{A}$, annotated with (j, k) is location (ℓ, j, k) in $\mathcal{A}'$. Furthermore there is a location err from which no accepting location can be reached, and corresponds to the fact that more than α edits have been made (i.e., the target on edit distance has been exceeded).

2. *(Accepting locations).* The only accepting locations in the automata $\mathcal{A}'$ are the locations in the copies of $\mathcal{A}$, which are annotated with $(|\,\mathrm{tw}\,|, k)$ for some k and which corresponds to accepting locations of $\mathcal{A}$.

3. *(Clocks).* The set of clocks C' in $\mathcal{A}'$ is C, the set of clocks in $\mathcal{A}$, together with the two additional clocks $\{\mathsf{x}, \mathsf{x}'\}$. The clock x measures the total time used and the clock x' measures the time used in the current location. Hence, x is never reset and x' is reset in every transition.

4. *(Transitions).* The location (ℓ, j, k) have up to $3 \cdot d + 1$ transitions, where d is the number of transitions in location ℓ of $\mathcal{A}$. Each transition from ℓ to ℓ' in $\mathcal{A}$ is copied three times and there is also at most one more transition t. The transition t exists iff $j \neq |\,\mathrm{tw}\,|$. If transition t exists, it resets the clock x' (though this is not necessary, but makes it conceptually easier to follow), uses the letter w_{j+1}, and has a clock constraint of $\mathsf{x}' = 0$. That is, it can only be used if no time has passed since arriving in (ℓ, j, k). The transition goes to $(\ell, j + 1, k + 1)$ (the transition t models *insertions* of the next letter). For a fixed transition t' between ℓ and ℓ' in $\mathcal{A}$, the three copies of it from (ℓ, j, k) each resets the same clocks as t', but also the clock x' and otherwise are as follows:

   (a) The first copy has the same clock constraint as t' but goes to location $(\ell', j, k+1)$, if $k < \alpha$ or err otherwise and has the letter ϵ (this copy corresponds to *deletion* of the current letter).

   (b) The second copy only exists if $j < |\,\mathrm{tw}\,|$. The second copy (if it exists) also has the same clock constraint as t' but goes to location $(\ell', j + 1, k + 1)$, if $j < \alpha$ or err otherwise and has the letter w_{j+1} (this copy corresponds to *substitution* of the current letter).

   (c) The third copy also only exists if $j < |\,\mathrm{tw}\,|$ and that t' is a w_{j+1}-transition. The third copy (if it exists) has the clock constraint $\mathcal{G}(t') \wedge (\mathsf{x} \in [t_{j+1} - \beta; t_{j+1} + \beta])$, where $\mathcal{G}(t')$ is the clock constraint of t' (the clock constraint is the same as for t' if $\beta = \infty$) and goes to location $(\ell, j+1, k)$ and has letter w_{j+1} (this copy corresponds to *no edit* having been made with the current letter).

   Intuitively, the transition t' checks for insertions, the first two copies of the transition check for deletions and substitutions, and the final copy of the transition checks for a correct move (i.e., no edits).

5. *(Invariant).* The invariant at location (ℓ, j, k) is the same as in location ℓ.

Before the correctness argument and complexity analysis we first present an example for illustration.

Example. Consider the timed automaton $\mathcal{A}$ for the timed language over a, b which (i) ends in a; and (ii) in which there is a difference in time of at most 1 between each consecutive a's and between each consecutive b's; and (iii) the first move has a delay of at most 1. The automaton consists of two locations, 1 and 2, location 1 is the start

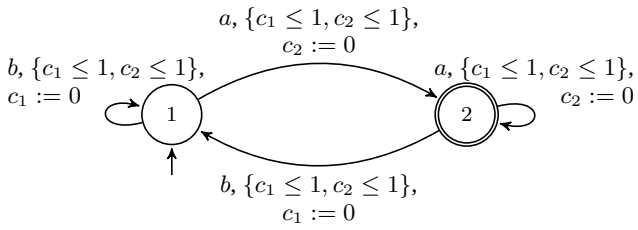

Figure 1: Example automata $\mathcal{A}$.

location and location 2 is the accepting location. There are two clocks in the automaton c_1 and c_2. The automaton contains four transitions, and each transition has clock constraint $c_1 \leq 1$ and $c_2 \leq 1$. From location j there are two outgoing transitions t_1^j and t_2^j. The transition t_1^j goes to location 1, resets clock c_1, and uses letter b. The transition t_2^j goes to location 2, resets clock c_2, and uses letter a. A pictorial illustration is given in Figure 1.

We then consider the decision problem $\mathsf{EdDec}(1, 1, \mathrm{tw}, \mathcal{A})$, where $\mathrm{tw} = ((a, 2), (b, 3))$. There is an illustration of the timed automaton $\mathcal{A}'$ corresponding to $\mathsf{EdDec}(1, 1, \mathrm{tw}, \mathcal{A})$ in Figure 2. For the sake of readability, we have removed the unreachable locations (which are location $(2, 0, 0)$, location $(1, 1, 0)$ and location $(2, 2, 0)$) in the figure, and instead of annotating the transitions with the letter, clock constraints and resets, we have annotated them only with letters in $\{N, D, I, S\}$, corresponding to a no-edit-transition, a deletion-transition, an insertion-transition, or a substitution-transition, respectively. Note that if there are multiple letters on an edge, then there is a copy of each transition in $\mathcal{A}'$, between the designated locations for each letter.

We see that there are only three paths in the graph of Figure 2 that reaches an accepting location from the start location. The paths corresponds to the timed words described below:

1. The sequence N, I which gives the run $(1, 0, 0) \to (2, 1, 0) \to (2, 2, 1)$. This sequence corresponds to the timed word $(a, 1)$ in $\mathcal{A}$, which has an edit distance of $(1, 1)$ from $(a, 2), (b, 3)$ (by inserting $(b, 3)$). This timed word is in $\mathcal{A}'$.

2. The sequence N, S which also gives the run $(1, 0, 0) \to (2, 1, 0) \to (2, 2, 1)$. This sequence corresponds to the timed word $(a, 1), (a, z)$ in $\mathcal{A}$ for some $z \geq 1$, which has an edit distance of $(1, 1)$ from $(a, 2), (b, 3)$ (by substituting (a, z) with $(b, 3)$). This timed word is in $\mathcal{A}'$.

3. The sequence N, N, D which gives the run $(1, 0, 0) \to (2, 1, 0) \to (1, 2, 0) \to (2, 2, 1)$. This sequence does not correspond to any run in $\mathcal{A}'$: the requirements on the first no-edit-transition is that $c_1 \leq 1, c_2 \leq 1, \mathsf{x} \in [2 - 1; 2 + 1]$ (which can only be satisfied by waiting one time unit in the start location), followed by a reset of c_2; and the requirement on the second no-edit-transition is that $c_1 \leq 1, c_2 \leq 1, \mathsf{x} \in [3-1; 3+1]$, but this cannot be satisfied, because $c_1 = \mathsf{x} = 1$ at the start location and any positive amount of waiting will ensure that we violate $c_1 \leq 1$, but we must wait at least one time unit before $\mathsf{x} \in [3 - 1; 3 + 1]$. Note that if we considered the decision problem $\mathsf{EdDec}(1, 2, \mathrm{tw}, \mathcal{A})$ instead, then there is such a run, e.g. $(a, 1), (b, 1), (a, 2)$ in $\mathcal{A}'$ and the word $(a, 1), (b, 1), (a, 2)$ has an edit distance of $(1, 2)$ from $(a, 2), (b, 3)$.

We now establish the correctness of the reduction and then analyse the complexity.

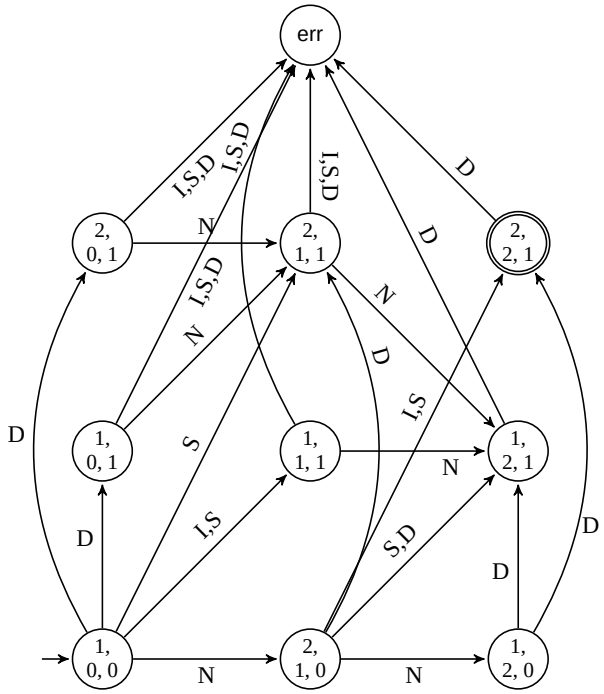

Figure 2: The automaton $\mathcal{A}'$ constructed from the timed automaton $\mathcal{A}$ in Figure 1.

LEMMA 3 (CORRECTNESS ARGUMENT). *The algorithm* $\mathrm{SOLED}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$ *correctly solves the decision problem* $\mathsf{EdDec}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$.

PROOF. For a given decision problem $\mathsf{EdDec}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$, we show that $\mathcal{A}'$ or $\mathcal{A}''$ is not empty iff there is a timed word in $\mathcal{A}$, with edit distance at most (α, β) to tw.

1. *(Non-emptiness implies $\mathcal{D}(\mathrm{tw}, \mathcal{A}) \leq_{\mathsf{lex}} (\alpha, \beta)$).* Consider an accepting word tw' of $\mathcal{A}'$ or $\mathcal{A}''$, ending in location $(\ell, |\,\mathrm{tw}\,|, k)$ for some k. Let tw'' be the word in $\mathcal{A}$ we get by following the transitions in tw', which are not insertions. Note that such a word exists, since the clock constraints on transitions in $\mathcal{A}'$ and $\mathcal{A}''$ which are not insertions are stronger than in $\mathcal{A}$ and the insertions does not matter (since they go between (ℓ, j, k) and $(\ell, j+1, k+1)$ and no time has passed). Note that tw' and tw spells the same (untimed) word (ignoring ϵ). Therefore, by making the modifications to the untimed word of tw'' as indicated by tw', we obtain tw. Note that there are at most k modifications, which is at most α. We now consider two cases: either $k = \alpha$ or $k < \alpha$.

   - If $k = \alpha$ (indicating that $\mathrm{tw}' \in \mathcal{A}'$), then whenever we used a no-edit-transition (or correct-move-transition) from (ℓ, j, k) to $(\ell', j + 1, k)$, then the corresponding move in $\mathcal{A}$ was such that the total time T was in $[t_{j+1} - \beta; t_{j+1} + \beta]$ and the letter used was the $(j+1)$-st letter of tw, indicating that no edit has been made and $|t_{j+1} - T| \leq \beta$. Hence the edit distance is at most $(k, \beta) = (\alpha, \beta)$.
   - If $k < \alpha$ (indicating that $\mathrm{tw}' \in \mathcal{A}''$, because in such cases the requirements in $\mathcal{A}'$ are stronger than in $\mathcal{A}''$), then using an argument like the preceding and the construction of $\mathcal{A}''$, we get that the edit distance is at most $(k, \infty) <_{\mathsf{lex}} (\alpha, \beta)$.

2. *($\mathcal{D}(\mathrm{tw}, \mathcal{A}) \leq_{\mathsf{lex}} (\alpha, \beta)$ implies non-emptiness).* Consider a timed word $\mathrm{tw}'' \in \mathcal{A}$, such that the edit distance $\mathcal{D}(\mathrm{tw}, \mathrm{tw}'') = \mathcal{D}(\mathrm{tw}'', \mathrm{tw})$ is at most (α, β). We consider the case that $(\alpha - 1, \infty) <_{\mathsf{lex}} \mathcal{D}(\mathrm{tw}'', \mathrm{tw})$ and show that $\mathcal{A}'$ is non-empty (the case $\mathcal{D}(\mathrm{tw}'', \mathrm{tw}) \leq_{\mathsf{lex}} (\alpha - 1, \infty)$ is similar, but in this case we show $\mathcal{A}''$ is non-empty instead of $\mathcal{A}'$). Let WE be a word edit which is used to show that $\mathcal{D}(\mathrm{tw}'', \mathrm{tw})$ is at most (α, β). We now show an accepting run of $\mathcal{A}'$ from tw'' and WE. Define $\ell_1 \in \mathcal{A}$ to be the start location of tw'' and let the corresponding location $(\ell_1, 0, 0)$ be the start location of the run. We can view the sequence of operators that WE makes on tw'' as the following sequence of letter operators: for all $i \geq 1$, the word edit WE firsts inserts some letters before the i-th letter of tw'', then it either substitutes, deletes, or keeps the i-th letter and then repeat for the $(i+1)$-st letter. Whenever WE inserts an letter into tw'', follow the insertion transition from the current location. In the other cases, there is a corresponding transition t in the word $\mathrm{tw}'' \in \mathcal{A}$. In that case follow the (substitution, deletion, no-edit) transition depending on the choice of WE in the obvious way. Note that if it follows the no-edit case, the time T spent on the sub-word up to transition t must be within β of the time used for the corresponding letter of tw by definition of WE and hence, in each case, we can use the indicated transition. At the end we end up in $(\ell', |\,\mathrm{tw}\,|, \alpha)$, where ℓ' is an accepting location of $\mathcal{A}$ the run tw'' ends in. Hence it follows that $\mathcal{A}'$ is non-empty.

The desired result follows. □

LEMMA 4 (SPACE COMPLEXITY ANALYSIS). *The algorithm* $\textsc{SolEd}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$ *can be implemented so that it uses polynomial space.*

PROOF. It is clear from the algorithm that we just need to solve the non-emptiness problem for $\mathcal{A}''$ and $\mathcal{A}'$ in PSPACE. Both automata have at most $n = |L| \cdot (|\,\mathrm{tw}\,| + 1) \cdot (\alpha + 1) + 1$ locations and the least common multiple (LCM) of the numbers in the clock constraints is $g \cdot d$, where g is the LCM of the numbers in the clock constraints of $\mathcal{A}$ and d the LCM of the denominators of the time stamps in the timed word tw and β, and the number of clocks is 2 more than the number of clocks $|\mathrm{C}|$ of $\mathcal{A}$. This indicates that we get a region abstraction with $(2+|\mathrm{C}|)! \cdot 4^{2+|\mathrm{C}|} \cdot (g \cdot d + 1)^{2+|\mathrm{C}|} \cdot n$ regions [2], each region of which can be written in polynomial space and the successors can also be computed in polynomial space. This indicates, similarly to how the non-emptiness problem for $\mathcal{A}$ is solved by Alur and Dill [2], that we can solve the non-emptiness problem for $\mathcal{A}'$ and $\mathcal{A}''$ in polynomial space. The desired upper bound follows. □

THEOREM 5 (COMPLEXITY). *The edit-distance decision problem* $\mathsf{EdDec}(\alpha, \beta, \mathrm{tw}, \mathcal{A})$ *is PSPACE-complete.*

PROOF. The theorem follows from Lemma 2, Lemma 3 and Lemma 4. □

REMARK 6. *We now argue that our construction above for timed automata specialized to untimed automata shows NL-completeness (non-deterministic log-space completeness) for untimed non-deterministic finite automata (NFA). In case of NFA, the second component does not exist. Also given an input untimed word w, the edit distance to an NFA $\mathcal{A}$ is at most $\max\{|w|, |L|\}$, where L is the set of locations of $\mathcal{A}$. Our construction above applied to NFA reduces the edit distance computation to non-emptiness of NFA. Moreover, since our reduction is local (i.e., it only modifies transitions of every location locally) it can be implemented in log-space. Since emptiness of NFA is NL-complete [12], we obtain the edit distance computation for an untimed word and an NFA is in NL. The same proof as in Lemma 2 shows that non-emptiness of NFA reduces to the edit distance computation problem. This gives us the following result.*

COROLLARY 7. *The edit-distance computation problem for an untimed word w and an untimed non-deterministic finite automata (NFA) is NL-complete.*

4. EDIT DISTANCE BETWEEN TIMED AUTOMATA

In this section we consider the computation of edit distance between two timed automata. We first show that the *exact* decision problem is undecidable, and then consider the *approximation* problem. We first formally define the approximation problem as a *promise* problem.

Promise problem. We will consider the following promise problem $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$: Given a rational number $\delta > 0$, a pair of numbers $(\alpha, \beta) \in \mathbb{N} \times (\mathbb{Q} \cup \{\infty\})$, and a pair of timed automata $\mathcal{A}, \mathcal{A}'$, the promise problem asks whether $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$, under the promise that either $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$ or $\mathcal{D}(\mathcal{A}, \mathcal{A}') >_{\mathsf{lex}} (\alpha, \beta + \delta)$. Intuitively the promise problem defines the approximation problem with an *additive* error in the second component.

Significance of the promise problem. We now explain why the promise problem is the appropriate formulation for approximation with additive error. First, given an algorithm for the promise problem with a space (resp. time) bound, we run a modified algorithm which runs as the given algorithm till the space (resp. time) bound has been exceeded; and if the bound has been exceeded, then it terminates and answers UNSURE. Thus even if the promise is not met, the algorithm always terminates in the required resource bound. For our concrete algorithm for the promise problem, the algorithm will always use at most exponential space, and terminate even if the promise is not satisfied, but if the promise is not satisfied, the algorithm may answer incorrectly. An alternative (perhaps more intuitive) approximation formulation is given numbers (α, β), timed automata $\mathcal{A}$ and $\mathcal{A}'$, and $\beta \geq \delta > 0$, if $(\alpha, \beta - \delta) <_{\mathsf{lex}} \mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta + \delta)$, the algorithm can answer UNSURE. If it does not, it must (correctly) answer YES if $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$, and NO if $\mathcal{D}(\mathcal{A}, \mathcal{A}') >_{\mathsf{lex}} (\alpha, \beta)$. We argue that solving the promise problem imply a solution to the above formulation, using a similar amount of resources. Given an instance of the problem, first we solve $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta - \delta)$ and $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$. Note that at least one of the answers is correct. If the results match and is YES, we have $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$ and return YES; if the results match and is NO, we have $\mathcal{D}(\mathcal{A}, \mathcal{A}') >_{\mathsf{lex}} (\alpha, \beta)$ and return NO. If the results do not match, then we have $(\alpha, \beta - \delta) <_{\mathsf{lex}} \mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta + \delta)$, and we return UNSURE. Hence we focus on the promise problem and present a solution to it.

4.1 Lower bounds

LEMMA 8 (HARDNESS OF EXACT DECISION PROBLEM). *Given two timed automata $\mathcal{A}$ and $\mathcal{A}'$ and two numbers (α, β), the decision problem whether $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$ is undecidable.*

PROOF. Let $\mathcal{A}$ be a timed automaton accepting all timed words. We will now argue that for a closed timed automata $\mathcal{A}'$, i.e. where all clock constraints are closed (except towards ∞), we have

$\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (0,0)$ iff $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ (i.e., the language universality problem for closed timed automata). It is clear that if $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$ then $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (0,0)$ and we will therefore argue that if $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (0,0)$ then $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$. The argument is as follows. Pick any timed word $\mathrm{tw} = (w, \overline{t})$. We will argue that $\mathrm{tw} \in \mathcal{L}(\mathcal{A}')$. We have that $\mathcal{D}(\mathrm{tw}, \mathcal{A}') \leq_{\mathsf{lex}} (0,0)$, which by definition, indicates that there is a sequence of timed words $(\mathrm{tw}_i)_{i \in \mathbb{N}}$, such that $\mathrm{tw}_i = (w_i, \overline{t}_i) \in \mathcal{L}(\mathcal{A}')$ and $\mathcal{D}(\mathrm{tw}, \mathrm{tw}_i) \leq_{\mathsf{lex}} (0, \frac{1}{i})$ (that is, for all i we have that $w = w_i$ and the j-th component of $\overline{t}$ and $\overline{t}_i$ differs by at most $\frac{1}{i}$ for all j). Because the clock constraints are closed, we also have that the limit of $(\mathrm{tw}_i)_{i \in \mathbb{N}}$ is in $\mathcal{A}'$. But the limit of $(\mathrm{tw}_i)_{i \in \mathbb{N}}$ is tw. Hence any arbitrary timed word is in $\mathcal{A}'$ and therefore also all timed words. Since the language universality problem for closed timed automata is undecidable, as shown by Ouaknine and Worrell [17], the desired result follows. □

Since Lemma 8 establishes the undecidability of the exact decision problem, we consider the problem of finding the first component exactly, but approximating the second component by an additive error term δ (as defined in the promise problem). Also note that multiplicative approximation is undecidable, since it would still require deciding if the edit distance is precisely $(0,0)$ or not. We now establish a complexity lower bound for the promise problem.

LEMMA 9 (HARDNESS OF APPROXIMATION). *Given two timed automata $\mathcal{A}$ and $\mathcal{A}'$, two numbers (α, β), and a rational number $\delta > 0$, the promise problem* $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$ *is EXPSPACE-hard.*

PROOF. As shown by Brenguier and Sankur [5], the decision problem for the universality of the untimed language of a timed automata is EXPSPACE-complete (i.e., given a timed automata $\mathcal{A}'$, deciding whether for every word w in Σ^* there exists $\mathrm{tw}' = (w', \overline{t}') \in \mathcal{L}(\mathcal{A}')$ such that $w' = w$ is EXPSPACE-complete). We can solve the universality of the untimed language problem, using our promise problem, let $\mathcal{A}$ be a timed automaton accepting all words and then deciding if the first component of the edit distance $\mathcal{D}(\mathcal{A}, \mathcal{A}')$ is 0 coincides with the untimed universality of $\mathcal{A}'$, i.e., $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', 0, \infty)$, for any $\delta > 0$, iff the untimed language of $\mathcal{A}'$ is universal. The desired results follows. Since [5] has not yet been published, we present an alternative proof: one can deduce that the promise problem is EXPSPACE-hard, by modifying the construction of Baier *et. al.* [3] (giving rise to the timed automaton $\mathcal{A}''$), for showing EXPSPACE-hardness of universality for their subset of timed automata. The modification is as follows: instead of requiring that each move has delay precisely 1 in a run which is rejected, we require that the floor of the total time used increases by 1 in every move. This indicates that if there exists a timed word tw which is not in their construction, then the timed word tw′, which has the same moves, but there the first delay is $\frac{1}{2}$ and the remaining are 1 is not in $\mathcal{A}''$. For that timed word tw′ we have that all timed words tw″, such that $\mathcal{D}(\mathrm{tw}', \mathrm{tw}'') \leq_{\mathsf{lex}} (0, \frac{1}{3})$ is also not in $\mathcal{A}''$, indicating that $\mathcal{D}(\mathrm{tw}', \mathcal{A}'') >_{\mathsf{lex}} (0, \frac{1}{3})$. Therefore, we can solve universality of their construction using the promise problem $\mathsf{PromEd}(\frac{1}{3}, \mathcal{A}, \mathcal{A}'', 0, 0)$, indicating that the promise problem is EXPSPACE-hard. □

4.2 Upper bound

Simplification. To simplify the remainder, we will assume that $\delta \geq 2$ and all numbers used, i.e. the ones in clock constraints of $\mathcal{A}$ and $\mathcal{A}'$ and the numbers α, β, and δ are integers. If one has an instance of the problem where this is not the case, one can simply scale all clock constraints, β and δ so that they are all integers and $\delta \geq 2$ (by multiplying with two times the LCM of the denominators), and consider $\lfloor \alpha \rfloor$ for the first component of the edit distance. We will furthermore assume that there is a bound of $c_{\max}$ on the time we can wait before moving. This assumption can be removed by including two additional columns corresponding to arbitrary high and arbitrary low difference between time stamps and suitable book-keeping. This will not be done explicitly in the present paper for sake of simplicity in presentation.

Overview of our algorithm. We will now present our algorithm in three stages.

- (Step 1) First we will give an algorithm that finds the first component of the edit distance.
- (Step 2) For a pair of timed automata $\mathcal{A}, \mathcal{A}'$ of edit distance at most (α, β), we bound the worst case time mismatch, between indices close together, for a pair of timed words $\mathrm{tw}, \mathrm{tw}'$ such that $(\alpha - 1, \infty) <_{\mathsf{lex}} \mathcal{D}(\mathrm{tw}, \mathrm{tw}') \leq_{\mathsf{lex}} (\alpha, \beta)$.
- (Step 3) Then finally, we will give an algorithm that tests if $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$, under the promise that either $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$ or $\mathcal{D}(\mathcal{A}, \mathcal{A}') >_{\mathsf{lex}} (\alpha, \beta + \delta)$.

The first two steps of the algorithm are relatively straight-forward and we present them below. Finally we present Step 3 in details.

4.2.1 Step 1 and Step 2 of the algorithm

Step 1 of the algorithm. Given two timed automata $\mathcal{A}$ and $\mathcal{A}'$, we want to compute the first component α of the edit distance. First we construct the corresponding region graphs $\mathsf{Reg}(\mathcal{A})$ and $\mathsf{Reg}(\mathcal{A}')$ and annotate on each transition the corresponding letter. By running an algorithm by Benedikt, Puppis and Riveros [4] to compute edit distance between two finite-state (untimed) automata, on the region graphs, we obtain α. The results of [4] also imply that the first component of the edit distance is at most $(|\mathsf{Reg}(\mathcal{A})| + 1) \cdot |\mathsf{Reg}(A')|$, if it is finite.

Step 2 of the algorithm: Bounding the time difference. We now present the following lemma for Step 2 of the algorithm.

LEMMA 10. *Let a pair (α, β) of numbers, and a pair of timed automata $\mathcal{A}, \mathcal{A}'$ be given, such that $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$. If there exists a timed word $\mathrm{tw} = (w, \overline{t}) \in \mathcal{L}(\mathcal{A})$ such that $\mathcal{D}(\mathrm{tw}, \mathcal{A}') >_{\mathsf{lex}} (\alpha - 1, \infty)$, then for all timed words $\mathrm{tw}' = (w', \overline{t}') \in \mathcal{L}(\mathcal{A}')$, where $\mathcal{D}(\mathrm{tw}, \mathrm{tw}') \leq_{\mathsf{lex}} (\alpha, \beta)$, and for all integers $1 \leq i \leq |\,\mathrm{tw}\,|$ and all integers j such that $i - 2 \cdot \alpha \leq j \leq i + 2 \cdot \alpha$ and $1 \leq j \leq |\,\mathrm{tw}'\,|$, we have that*

$$|t_i - t'_j| \leq 4 \cdot \alpha \cdot c_{\max} + \beta \ .$$

PROOF. Let a pair of numbers (α, β) and a pair of timed automata $\mathcal{A}, \mathcal{A}'$ be given, such that $\mathcal{D}(\mathcal{A}, \mathcal{A}') \leq_{\mathsf{lex}} (\alpha, \beta)$. Consider a timed word $\mathrm{tw} = (w, \overline{t}) \in \mathcal{L}(\mathcal{A})$, such that $\mathcal{D}(\mathrm{tw}, \mathcal{A}') >_{\mathsf{lex}} (\alpha - 1, \infty)$, and a timed word $\mathrm{tw}' = (w', \overline{t}') \in \mathcal{L}(\mathcal{A}')$, where $\mathcal{D}(\mathrm{tw}, \mathrm{tw}') \leq_{\mathsf{lex}} (\alpha, \beta)$. Let WE be some word edit witnessing $\mathcal{D}(\mathrm{tw}, \mathrm{tw}') \leq_{\mathsf{lex}} (\alpha, \beta)$. Fix some index i in tw. If $i \leq 2 \cdot \alpha$, then note that $0 \leq t_i \leq i \cdot c_{\max} \leq 2 \cdot \alpha \cdot c_{\max}$ and $0 \leq t'_{i+\alpha} \leq (i + 2 \cdot \alpha) \cdot c_{\max} \leq 4 \cdot \alpha \cdot c_{\max}$, because the time can at most increase with $c_{\max}$ in every move, from which the statement follows, for such i. Hence, we only need to consider $i > 2 \cdot \alpha$. Consider some index j in w which corresponds to index i of w', then $|i - j| \leq \alpha$, since it is the number of insertions minus deletions before index i. Also note that in any set S of indices in tw′ of size $\alpha + 1$, at least one index i' corresponds to some index j' in tw, because otherwise there would be at least $\alpha + 1$ edits. This is especially true for the set of indices $S' = \{j - \alpha, j - \alpha - 1, \ldots, j - 2 \cdot \alpha\}$ of size $\alpha + 1$ (note that they are all indices of tw′, because $i \geq 2 \cdot \alpha$ and the length of the words cannot differ by more than α). Let

i', j' be some indices such that i' in S' corresponds to j'. We then get that $j' \leq j \leq j' + 3 \cdot \alpha$, by the preceding definition of j'. Because of the correspondence between i' and j' we also get that $|t'_{i'} - t_{j'}| \leq \beta$. Since we can increase the time used by at most $c_{\max}$ in every move and that $\overline{t}$ is monotonically non-decreasing, we also get that $t'_{i'} - \beta \leq t_j \leq t'_{i'} + \beta + 3 \cdot \alpha$. By the same argument we also get that

$$t'_{j-2\cdot\alpha} - \beta \leq t'_{i'} - \beta \leq t_j \leq t'_{i'} + \beta + 3 \cdot \alpha \leq t'_{j-2\cdot\alpha} + \beta + 4 \cdot \alpha$$

and also

$$t'_k - \beta - 4 \cdot \alpha \leq t_j \leq t_k + \beta + 3 \cdot \alpha \ ,$$

where $k = \min(|\,\mathrm{tw}'\,|, j + 2 \cdot \alpha)$. Therefore, by monotonicity of $\overline{t}'$ we get that for all $j - 2 \cdot \alpha \leq i \leq k$, that $|t_j - t'_i| \leq 4 \cdot \alpha + \beta$ and the desired result follows. □

4.2.2 Step 3 of the algorithm

We will now give an algorithm that solves the decision problem $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$.

Deducing time passage. Given a timed automata $\mathcal{A}$, we will consider $\mathsf{Eps}(\mathcal{A})$, which is identical to $\mathcal{A}$, except that (1) it has one more clock x; (2) modifies the clock constraints on the transitions in $\mathcal{A}$; and (3) also adds $|L|$ new transitions, one from each location. For each transition t in $\mathcal{A}$, the corresponding transition in $\mathsf{Eps}(\mathcal{A})$ also includes $\mathsf{x} < 1$ as a part of the clock constraint. For each location ℓ in $\mathcal{A}$, the new transition in $\mathsf{Eps}(\mathcal{A})$ from the corresponding location in $\mathsf{Eps}(\mathcal{A})$ is an ϵ-transition and a self-loop with clock constraint $\{\mathsf{x} = 1 \wedge \bigwedge_{c \in C} c \leq c_{\max}\}$. Note that this ensures that $\epsilon^{-1}(\mathcal{L}(\mathsf{Eps}(\mathcal{A}))) = \mathcal{L}(\mathcal{A})$, where ϵ^{-1} is the function on timed languages that removes all occurrences of the letter ϵ and the corresponding time stamps. The construction ensures that the floor of the total timed used in a prefix of a run is precisely the number of ϵ-transitions used in the prefix. A similar construction was used by Chatterjee and Prabhu [6] for computation of quantitative simulation.

The triple $\mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A, V, M)$**.** Given a timed word $\mathrm{tw} = (w, \overline{t})$, a pair of timed automata $\mathcal{A}, \mathcal{A}'$, and some target pair of numbers (α, β), we define the triple $\mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A, V, M)$. We now describe the components of the triple.

- The first component A is a subset of regions in $\mathsf{Reg}(\mathsf{Eps}(\mathcal{A}))$.
- The second component V is a vector of length α, and each entry in the vector is a subset of regions of $\mathsf{Reg}(\mathcal{A}')$.
- The third component M is a matrix of dimension $(8 \cdot \alpha \cdot c_{\max} + 2 \cdot \beta + 4, \alpha + 1)$. Each entry (a_1, a_2) of M is a subset of regions of $\mathsf{Reg}(\mathsf{Eps}(\mathcal{A}'))$.

To simplify the definition of $\mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta)$, we will now first assign labels to M and V. The rows are labeled $0, \ldots, \alpha$ and the columns are labeled $\{-4 \cdot \alpha \cdot c_{\max} - \beta - 2, \ldots, 4 \cdot \alpha \cdot c_{\max} + \beta + 2\}$. Similarly we assign labels $0, \ldots, \alpha - 1$ to the entries of V. The subset A is the set of regions in $\mathsf{Reg}(\mathsf{Eps}(\mathcal{A}))$ one can get to such that the i-th non-ϵ-transition used is w_i and there are $\lfloor t_i \rfloor$ many ϵ-transitions before that transition. The α-vector V is such that entry a_1 of V contains the regions, which can be reached after a timed word $\mathrm{tw}' = (w', \overline{t}')$, such that $\mathcal{D}(w, w') \leq a_1$ (this is easy to compute on the region graph, using the algorithm by Benedikt, Puppis and Riveros [4], since it only considers the untimed part). Also, a given region $r \in \mathsf{Reg}(\mathsf{Eps}(\mathcal{A}'))$ is in entry (a_1, a_2) of M (where a_1 and a_2 are resp. row and column labels of M) iff there exists a timed word $\mathrm{tw}' = (w', \overline{t}')$, such that (1) one can get to r after having processed tw'; and (2) there exists a word edit WE with at most a_1 edits between tw and tw' such that at every pair of corresponding timed letters i, j, we have that $||\lfloor t_i \rfloor - \lfloor t'_j \rfloor|| \leq \beta + 1$; and (3) $\lfloor t_{|\,\mathrm{tw}\,|} \rfloor - \lfloor t'_{|\,\mathrm{tw}'\,|} \rfloor = a_2$.

Feasible and successful impact **triples.** We will call a triple (A, V, M) *feasible* if $\mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A, V, M)$ for some tw and *successful* if it is feasible and A contains a region with a location of $\mathcal{A}$ which is accepting, but no entry in neither V nor M contains a region with a location of $\mathcal{A}'$ which is accepting. We will now argue that there exists a successful triple iff the answer is NO to the promise problem $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$.

LEMMA 11. *There exists a successful triple* (A, V, M) *iff the answer is NO to the promise problem* $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$.

PROOF. We will first argue that a successful triple implies that the answer to $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$ is NO. Consider a triple $(A, V, M) = \mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta)$ for some $\mathrm{tw} = (w, \overline{t})$, which is successful. Consider some accepting region r in A and let $\mathrm{tw}' = (w', \overline{t}')$ be some timed word that goes to r from some start location, such that for all i we have that $w_i = w'_i$ and $\lfloor t_i \rfloor = \lfloor t'_i \rfloor$. Such a run exists by definition of impact. We have that $\mathrm{tw}' \in \mathcal{L}(\mathcal{A})$. Assume towards contradiction that there is a $\mathrm{tw}'' \in \mathcal{L}(\mathcal{A}')$ such that $\mathcal{D}(\mathrm{tw}', \mathrm{tw}'') \leq_{\mathsf{lex}} (\alpha, \beta)$. Let r' be the accepting region one reaches in $\mathsf{Reg}(\mathsf{Eps}(\mathcal{A}'))$ after the run $\mathrm{tw}'' = (w'', \overline{t}'')$. First consider the case that $\mathcal{D}(\mathrm{tw}', \mathrm{tw}'') \leq_{\mathsf{lex}} (\alpha - 1, \infty)$. This implies that r' is in entry $(\alpha - 1)$ of V and hence contradicts that (A, V, M) is successful. If, on the other hand $(\alpha - 1, \infty) <_{\mathsf{lex}} \mathcal{D}(\mathrm{tw}', \mathrm{tw}'') \leq_{\mathsf{lex}} (\alpha, \beta)$, then r' is in entry $(\alpha, \lfloor t'_{|\,\mathrm{tw}'\,|} \rfloor - \lfloor t''_{|\,\mathrm{tw}''\,|} \rfloor)$ of M (by Lemma 10, this is an entry of the matrix) and again contradicts that (A, V, M) is successful.

We will now argue that if the answer to $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$ is NO, then there is a successful triple. By definition of $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$ we know that there is a timed word $\mathrm{tw} = (w, \overline{t}) \in \mathcal{L}(\mathcal{A})$ such that for all timed words $\mathrm{tw}' \in \mathcal{L}(\mathcal{A}')$, we have that $\mathcal{D}(\mathrm{tw}, \mathrm{tw}') \geq_{\mathsf{lex}} (\alpha, \beta + 2)$. Fix such a timed word tw. There are two cases. Either for some tw' we have that $\mathcal{D}(\mathrm{tw}, \mathrm{tw}') >_{\mathsf{lex}} (\alpha, \infty)$ or not.

- If we have that $\mathcal{D}(\mathrm{tw}, \mathrm{tw}') >_{\mathsf{lex}} (\alpha, \infty)$, then all entries of V do not contain a region with accepting location. But the requirements to be in entry a_1 of V are satisfied by every region in (a_1, a_2) of M for all a_2. But this implies that the matrix also does not contain a region with accepting location.
- In the other case, there must be a timed word $\mathrm{tw}' = (w', \overline{t'})$ such that $(\alpha, \beta + 2) \leq_{\mathsf{lex}} \mathcal{D}(\mathrm{tw}, \mathrm{tw}') \leq_{\mathsf{lex}} (\alpha, \infty)$. First note that for all $a_1 < \alpha$, no region with an accepting location can be in entry (a_1, a_2) of M nor in entry a_1 for V, because $\mathcal{D}(\mathrm{tw}, \mathcal{A}) >_{\mathsf{lex}} (\alpha - 1, \infty)$. We therefore only need to consider the entries in row α. But then for any word edit with α edits, there must be some index i in w corresponding to index j of w', such that $|t_i - t'_j| \geq \beta + 2$, by definition of edit distance and since we consider that $\delta \geq 2$. But then also $||\lfloor t_i \rfloor - \lfloor t'_j \rfloor|| > \beta + 1$, implying that no region with accepting location can be in entry (α, a_2) of M for any a_2.

The desired result follows. □

Computing impact **- the start case.** It is easy to compute $\mathrm{impact}(\epsilon, \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A, V, M)$, because (1) A are simply the regions corresponding to time 0 on all clocks in the start locations; and (2) for each a_1, entry a_1 of V is the set of regions in $\mathsf{Reg}(\mathcal{A})$, reachable in at most a_1 moves; and (3) for each a_1 and

$a_2 \geq 0$, entry $(a_1, -a_2)$ of M (the entries (a_1', a_2'), where $a_2' > 0$ are empty, because the time stamps are always non-negative numbers, and the time for the timed word ϵ is 0) is the set of regions in $\mathsf{Reg}(\mathsf{Eps}(\mathcal{A}))$, reachable in precisely a_1 many non-ϵ-moves and a_2 many ϵ-moves (note that every letter used must be deleted to match ϵ and thus we do not need to consider the requirement on times).

Computing impact **- the move case.** Given $\mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A, V, M)$, for some A, V, M and for some timed word $\mathrm{tw} = (w, \overline{t})$, we can compute each triple $\mathrm{impact}(\mathrm{tw} \circ (\sigma, t), \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A', V', M')$ for some (σ, t). Let $t' = \lfloor t \rfloor - \lfloor t_{|\,\mathrm{tw}\,|} \rfloor$. Then A' is the set of regions one can get to from some region in A, using first t' many ϵ-transitions and then one σ-transition. Each entry a_1 of V' can be computed directly from V, similar to the algorithm by Benedikt, Puppis and Riveros [4] for untimed automata. Also, entry (a_1, a_2) of M' consists of the regions one can get to from some region in (a_1', a_2') of M for $a_2' \in [-4 \cdot \alpha \cdot c_{\max} - \beta - 1, 4 \cdot \alpha \cdot c_{\max} + \beta + 1]$ using (1) no transitions, if $a_1 = a_1' + 1$ and $a_2 = a_2' + t'$ (this corresponds to insertion); or (2) first $(a_2 - a_2' + t')$ many ϵ-transitions and then any non-ϵ-transition if $a_1 = a_1' + 1$ (this corresponds to substitution); or (3) first $\hat{t}_0$ many ϵ-transitions and then some σ-transition, followed by $\hat{t}_1$ many ϵ-transitions and then some non-ϵ-transition, followed by $\hat{t}_2$ many ϵ-transitions and then some non-ϵ-transition and so on until $\hat{t}_n$ many ϵ-transitions and then some non-ϵ-transition, where $n = a_1 - a_1'$ and $a_2 - a_2' + t' = \sum_{i=0}^{n} \hat{t}_i$ (corresponding to one correct move followed by n deletions). It is easy to see that we can always assume that all deletions comes directly after a correct move (or appears at the beginning).

Computing impact **- correctness.** We will now argue that our computation of impact satisfy the properties required.

LEMMA 12. *The computation of* impact *is correct.*

PROOF. In both the start case and the move case, it should be clear that the first two components (that is A and V) of the triple are correctly computed.

We now recall the requirements on being in entry (a_1, a_2) of M: A given region $r \in \mathsf{Reg}(\mathsf{Eps}(\mathcal{A}'))$ is in entry (a_1, a_2) of M iff there exists a timed word $\mathrm{tw}' = (w', \overline{t}')$, such that (1) one can get to r after having processed tw'; and (2) there exists a word edit WE with at most a_1 edits between tw and tw' such that at every pair of corresponding indices i, j, we have that $|\lfloor t_i \rfloor - \lfloor t_j' \rfloor| \leq \beta + 1$; and (3) $\lfloor t_{|\,\mathrm{tw}\,|} \rfloor - \lfloor t'_{|\,\mathrm{tw}'\,|} \rfloor = a_2$.

Start case of M**.** In the start case for M, it is clear that there exists a timed word $\mathrm{tw}' = (w', \overline{t}')$ to each of the regions of entry $(a_1, -a_2)$ of M, because of our use of the region abstraction [2]. Also, it contains a_1 many non-ϵ-moves and a_2 many ϵ-moves, indicating that $\mathcal{D}(\epsilon, \mathrm{tw}') \leq_{\mathrm{lex}} (a_1, \infty)$ (indicating that we satisfy (1)) and that $\lfloor t'_{|\,\mathrm{tw}'\,|} \rfloor = a_2$ (indicating that we satisfy (3)). Also, since no word edit can have any corresponding indices between ϵ and tw' we satisfy (2).

Move case of M**.** In the move case, we have $\mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A, V, M)$ and we must compute each $\mathrm{impact}(\mathrm{tw} \circ (\sigma, t), \mathcal{A}, \mathcal{A}', \alpha, \beta) = (A', V', M')$ for any (σ, t). By Alur and Dill [2] we see that we satisfy (1) (since it indicates that there is a timed word ending in each reachable region). Also, by letting the word edit WE we consider in (2), be any that have the pairs of corresponding indices defined by our correct moves, we see that we satisfy (2). It is clear that we can do the rest of the word edit afterwards in a_1 edits, since we increase the number of edits we need whenever we do not use a correct move. In regards to (3), we have the value of $\lfloor t_{|\,\mathrm{tw}\,|} \rfloor - \lfloor t_{|\,\mathrm{tw}\,|-1} \rfloor$, from our computation of A'. Wagner and Fischer [20] shows that we can split up a word edit between a word $w \circ \sigma$ and a word w', so that w is edited to w'' and σ is edited to w''' for some $w'' \circ w''' = w$. It is easy to see that this generalises to timed words. Therefore, we must have that tw' can be split up into tw'' and tw''' such that $\mathrm{tw}' = \mathrm{tw}'' \circ \mathrm{tw}'''$ and such that tw'' is in M (because of Lemma 10). From M we get $\lfloor t_{|\,\mathrm{tw}\,|-1} \rfloor - \lfloor t'_{|\,\mathrm{tw}''\,|} \rfloor$ for all possible tw''. Thus to compute $\lfloor t_{|\,\mathrm{tw}\,|} \rfloor - \lfloor t'_{|\,\mathrm{tw}'\,|} \rfloor$, we just need $\lfloor t'_{|\,\mathrm{tw}'''\,|} \rfloor = \lfloor t'_{|\,\mathrm{tw}'\,|} \rfloor - \lfloor t'_{|\,\mathrm{tw}'''\,|} \rfloor$, which is easy to find, by counting the number of ϵ-transitions used and is done correctly by the description.

We therefore conclude that also M is computed correctly. □

Space complexity of impact**-triple computation.** Observe that a triple (A, V, M) consists of some exponential number of subsets of exponential sized sets of regions and therefore each triple have at most exponential size. Given the preceding it is also clear that we can compute each successor of the (possibly) exponentially many successors in exponential space. Note that given a feasible triple (A, V, M) it is easy to check in exponential space if the triple is also successful.

The algorithm SolPromEd**.** Our algorithm SolPromEd for solving $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$, is as follows: (1) first compute $\mathrm{impact}(\epsilon, \mathcal{A}, \mathcal{A}', \alpha, \beta)$; and (2) then guess a timed word (with one letter and the floor of the corresponding time stamp at a time) to a successful triple and compute the impact triples using the move case, and check that the triple is successful and then return NO. If there is no successful triple, return YES. The correctness follows from Lemma 11 and Lemma 12. The above algorithm is non-deterministic (since it involves a guess of a timed word) and the space complexity is exponential (since the impact-triple computation and check is exponential space). Since NEXPSPACE=EXPSPACE by Savitch's Theorem [19], we obtain that the algorithm for the promise problem can be implemented in exponential space. Along with Lemma 9 we obtain the following result.

THEOREM 13 (COMPLEXITY OF APPROXIMATION). *The promise problem* $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$ *can be solved in exponential space; and the problem is EXPSPACE-hard.*

Relating our algorithm to the algorithm by Benedikt, Puppis and Riveros [4]. Our algorithm SolPromEd for deciding the decision problem $\mathsf{PromEd}(\delta, \mathcal{A}, \mathcal{A}', \alpha, \beta)$ is similar to the algorithm by Benedikt, Puppis and Riveros [4] for solving the problem of edit distance between untimed languages. There they construct $\mathrm{impact}(w, u\mathcal{A}, u\mathcal{A}', \alpha')$, for some word w, some finite-state (untimed) automata $u\mathcal{A}, u\mathcal{A}'$ and some target α', whereas we construct $\mathrm{impact}(\mathrm{tw}, \mathcal{A}, \mathcal{A}', \alpha, \beta)$, for some timed word tw, some timed automata $\mathcal{A}$ and $\mathcal{A}'$ and some target (α, β). But their construction only have parallels for the first two components of our triple (they do not have the matrix component in their construction). Also, in their construction a given location could only occur once in their vector, in contrast, we can have a given location in each column of M and in V (because, while it is always better to make less errors, it is not clear what the best time mismatch is before the next move).

5. DISCUSSION ON EXTENSIONS AND CONCLUSION

In this work we have considered the edit distance computation for timed automata under the lexico-graphic ordering. We now discuss several extensions that can be obtained from our results.

1. *Point-wise comparison and Pareto curve.* Instead of the lexico-graphic comparison we could also consider point-wise comparison between the components of the edit distance, and then compute the Pareto points where one component cannot be improved without sacrificing the other. The Pareto curve consists of all Pareto points. Consider bounds B_1 and B_2 for the bounds for Pareto curve. Given a solution to the decision problem with point-wise comparison which asks whether the first component is at most α and the second component at most β, the δ-approximation of the Pareto curve bounded by B_1 and B_2, for $\delta > 0$, can be computed as follows: enumerate α from 0 to B_1 as integers, and for a fixed α, choose β iteratively by a binary search in the interval $[0, B_2]$ until the imprecision is smaller than δ, and consider the decision problem for the point-wise comparison with α and β. Our solution for lexico-graphic ordering can also be modified to solve the point-wise comparison. The modifications are as follows: (A) For the solution of Section 3, we remove automaton $\mathcal{A}''$, and in automaton $\mathcal{A}'$ consider a location (ℓ, j, k) to be accepting if ℓ is accepting, and the automaton $\mathcal{A}'$ is non-empty iff $\mathcal{D}_1(\mathrm{tw}, \mathcal{A}) \leq \alpha$ and $\mathcal{D}_2(\mathrm{tw}, \mathcal{A}) \leq \beta$ (i.e., pointwise comparison). Also note that if the language of the input automaton is non-empty, then we have B_1 bounded by $\alpha = \max\{|\,\mathrm{tw}\,|, d\}$ and B_2 bounded by $\max\{(|\,\mathrm{tw}\,| + \alpha) \cdot c_{\max}, t_{|\,\mathrm{tw}\,|}\}$ (refer to the paragraph of Bound on the components of the edit distance in Section 3). (B) For the solution of Section 4, we simply need to remove the vector V from the triple for the solution. Also in this case B_1 is bounded by the product of the size of the region graphs (refer to Step 1 of our algorithm in Section 4).

2. *Delay instead of time mismatch.* In our definition of the second component of the edit distance we considered the more challenging notion of the absolute timing mismatch. Another alternative notion is to consider the delays, where the *delay* Δ_i in index i is the time difference $t_i - t_{i-1}$ between the $(i-1)$-th and i-th move. Then instead of the timing mismatch of t_i and t'_i we could consider the delay mismatch Δ_i and Δ'_i. The problem with the mismatch of delay is technically slightly easier (though has the same computational complexity) and we discuss the details for the solution of Section 4. To find the delay difference between $\mathcal{A}$ and $\mathcal{A}'$, we compute $\mathsf{Reg}(\mathcal{A})$ and $\mathsf{Reg}(\mathcal{A}')$, then label each transition in $\mathsf{Reg}(\mathcal{A})$ with the corresponding letter and some symbol indicating the floor of the delay used. Then each transition t in $\mathsf{Reg}(\mathcal{A})'$ is copied $2 \cdot \beta + 1$ times, one copy for each integer $y \in [-\beta, \beta]$. Let d be the floor of the delay of transition t. We mark the y-th copy of transition t with the corresponding letter of t and a symbol indicating $y + d$. We then run the algorithm of [4] on the resulting graphs. Note that whenever we match a letter, then the difference in delay must be in $[-\beta - 1, \beta + 1]$ as required.

3. *Rectangular hybrid automata.* While we have presented the solution for timed automata, our results also extends to rectangular hybrid automata [11]. First note that in our solution of Section 3, we either copy transitions, or include additional rectangular constraints, and thus our transformation ensures that if we start with a rectangular hybrid automata we obtain another rectangular hybrid automata. Since language emptiness is decidable in EXPTIME for rectangular hybrid automata [11], our solution also extends to rectangular hybrid automata giving decidability in EXPTIME. Finally the solution of Section 4 relied on the region abstraction for timed automata, and since a similar finite-quotient based abstraction exists for rectangular hybrid automata [11], the impact-triple based computation can also be done for rectangular hybrid automata. Intuitively, the computation for timed automata was a PSPACE computation over exponential size structures leading to exponential space bound, and for rectangular automata we have an EXPTIME computation over exponential size structures that gives 2EXPTIME complexity.

Concluding remarks. In this work we extended the notion of edit distance from untimed languages to timed languages defined by timed automata. Our results characterized precisely the decidability and complexity of the computation between timed words and timed automata, and between timed automata. While we established the complexity is PSPACE-complete for timed words and timed automata, the problem is undecidable for a pair of timed automata. For the approximation problem between a pair of timed automata, we establish exponential space lower and upper bound. We also discussed how our results can be extended to variants with point-wise comparison, delay instead of time mismatch, and the more general model of rectangular automata. We believe our results will provide a theoretical basis for approximate matching between timed words and timed languages.

6. REFERENCES

[1] A. Aho and T. Peterson. A minimum distance error-correcting parser for context-free languages. *SIAM J. of Computing*, 1:305–312, 1972.

[2] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] C. Baier, N. Bertrand, P. Bouyer, and T. Brihaye. When are timed automata determinizable? In *ICALP (2)*, pages 43–54, 2009.

[4] M. Benedikt, G. Puppis, and C. Riveros. Regular repair of specifications. In *LICS*, pages 335–344. IEEE Computer Society, 2011.

[5] R. Brenguier and O. Sankur. Hardness of untimed language universality. Presentation at YR-CONCUR, 2011.

[6] K. Chatterjee and V. S. Prabhu. Quantitative timed simulation functions and refinement metrics for real-time systems. In *HSCC*, 2013.

[7] A. Donzé, T. Ferrère, and O. Maler. Efficient robust monitoring of signal temporal logic. In *CAV 2013*, LNCS. Springer, 2013.

[8] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, LNCS, pages 92–106. Springer, 2010.

[9] G. Fainekos and G. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42), 2009.

[10] G. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel. Verification of automotive control applications using S-TaLiRo. In *Proc. American Control Conference*, 2012.

[11] T. A. Henzinger and P. W. Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221(1-2):369–392, June 1999.

[12] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Adison-Wesley Publishing Company, Reading, Massachusets, USA, 1979.

[13] R. Karp. Mapping the genome: some combinatorial problems arising in molecular biology. In *STOC 93*, pages 278–285. ACM, 1993.

[14] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-Doklady*, 10:707–710, 1966.

[15] M. Mohri. Edit-distance of weighted automata: general definitions and algorithms. *Intl. J. of Foundations of Comp. Sci.*, 14:957–982, 2003.

[16] T. Okuda, E. Tanaka, and T. Kasai. A method for the correction of garbled words based on the levenshtein metric. *IEEE Trans. Comput.*, 25:172–178, 1976.

[17] J. Ouaknine and J. Worrell. Universality and language inclusion for open and closed timed automata. In O. Maler and A. Pnueli, editors, *HSCC*, volume 2623 of *Lecture Notes in Computer Science*, pages 375–388. Springer, 2003.

[18] G. Pighizzini. How hard is computing the edit distance? *Information and Computation*, 165:1–13, 2001.

[19] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, Apr. 1970.

[20] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, Jan. 1974.