Testing Membership for Timed Automata

Richard Lassaigne
University Paris, IMJ-CNRS
lassaigne@math.univ-paris-diderot.fr

Michel de Rougemont
University Paris II, IRIF-CNRS
mdr@irif.fr

Abstract. Given a timed automaton which admits thick components and a timed word w, we present a tester which decides if w is in the language of the automaton or if w is ϵ-far from the language, using finitely many samples taken from the weighted time distribution μ associated with the input w. We introduce a distance between timed words, the *timed edit distance*, which generalizes the classical edit distance. A timed word w is ϵ-far from a timed language if its relative distance to the language is greater than ϵ.

Keywords: Property testing, Approximate algorithms, Timed automata

Contents

1 Introduction

We study the membership of *timed words* for timed automata, an NP-complete problem [3], from the approximation point of view. We introduce a *timed edit distance* between timed words and study how to distinguish an accepted timed word from a timed word which is ϵ-far from the language of accepted words. We follow the property testing approach with this new distance between timed words. Samples are taken following the weighted time distribution μ where the probability to choose a letter is proportional to its relative time delay or duration which we call its *weight*. The tester takes samples which are factors of weight at least k, taken from the distribution μ. Such samples can also be taken from a stream of timed words, without storing the entire input.

We consider timed automata with linear constraints [2] and construct the associated region automata with m states. Let G be the graph whose nodes are the connected components of the region automaton, which we assume to be thick or of non-vanishing entropy [7]. Let B be the maximum constant appearing in a time constraint of an automaton, let l be the diameter of G. We require l independent samples of μ, each of weight $k \geq 24l.m.B/\epsilon$. It guarantees that if a timed word of total weight T is ϵ-far from the language of the timed automaton, it will be rejected with constant probability, i.e. independent of T.

Let Π be a path of G, i.e. a sequence of connected components. Given a path Π and samples from μ, definition 4 specifies when these samples are compatible with Π. The tester checks if there is a Π such that the l samples are compatible for this Π. It rejects if no Π is compatible with the samples.

The main result, theorem 1, presents the analysis of the tester. First, lemma 2 guarantees that a word of the language of the timed automaton is accepted by the tester. To prove that an ϵ-far timed word is rejected with constant probability, we construct a corrector for a connected component (lemmas 3 to 5) of the region automaton and extend it to a sequence Π. We then use the corrector (lemmas 6 and 7) to prove the main result.

In the second section, we fix our notations of timed automata and recall the definitions of thick and thin components. In the third section, we define the timed edit distance in the property testing context. In the fourth section, we define our membership tester and give its analysis.

2 Timed automata

Let X be a finite set of variables, called clocks. A clock valuation over X is a mapping $v : X \longrightarrow \mathbb{R}^+$ that assigns to each clock a time value. For each $t \in \mathbb{R}^+$, the valuation $v+t$ is defined by $\forall x \in X \;\; (v+t)(x) = v(x)+t$. A *clock constraint* over X is a conjunction g of atomic constraints of the form: $x \bowtie c$ where $x \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Let $\mathcal{C}(X)$ be the set of all clock constraints. We write $v \models g$ when the clock valuation v satisfies the clock constraint g and we note $[g]$ the set of clock valuations satisfying g. For a subset Y of X, we denote by $[Y \leftarrow 0]v$ the *reset* valuation such that for each $x \in Y$, $([Y \leftarrow 0]v)(x) = 0$ and for each $x \in X \backslash Y$, $([Y \leftarrow 0]v)(x) = v(x)$.

A timed automaton is a tuple $\mathcal{A} = (\Sigma, Q, X, E, I, F)$ where Σ is a finite set of events, Q is a finite set of locations, $I \subseteq Q$ is the set of initial locations, $F \subseteq Q$ is a set of final locations and $E \subseteq Q \times (\mathcal{C}(X) \times \Sigma \times 2^X) \times Q$ is a finite set of transitions. A transition is a triple (q, e, q') where $e = (g, a, Y)$, i.e. g is the clock constraint, $a \in \Sigma$ and Y is the set of clocks which are reset in the transition.

A state is a tuple (q, v), a location q and a valuation of the clocks v. Let B be the maximum value of the constant c in the atomic constraints.

2.1 Timed words

A timed word w is a sequence $(a_i, t_i)_{1 \leq i \leq n}$ where $a_i \in \Sigma$ and t_i is a strictly monotonic sequence of values in $\mathcal{R}^+$. A path π in $\mathcal{A}$ is a finite sequence of consecutive transitions: $(q_{i-1}, e_i, q_i)_{1 \leq i \leq n}$ where $(q_{i-1}, e_i, q_i) \in E$ and $e_i = (g_i, a_i, Y_i)$ where $g_i \subseteq \mathcal{C}(X)$, $a_i \in \Sigma$ and $Y_i \subseteq X$, for each $i \geq 0$. The path π is *accepting* if it starts in an initial location $q_0 \in I$ and ends in a final location $q_n \in F$. For a timed word w, let $untime(w)$ be the sequence of letters $(a_i)_{1 \leq i \leq n}$. A *run of the automaton along the path* π is a sequence:

$$(q_0, v_0) \xrightarrow[t_1]{g_1, a_1, Y_1} (q_1, v_1) \xrightarrow[t_2]{g_2, a_2, Y_2} (q_2, v_2) \dots \xrightarrow[t_n]{g_n, a_n, Y_n} (q_n, v_n)$$

where $(a_i, t_i)_{1 \leq i \leq n}$ is a timed word and $(v_i)_{1 \leq i \leq n}$ a sequence of clock valuations such that:

$$(*) \quad \forall x \in X \quad v_0(x) = 0 \quad v_{i-1} + (t_i - t_{i-1}) \models g_i$$

$$v_i = [Y_i \leftarrow 0](v_{i-1} + (t_i - t_{i-1}))$$

We read a_i for a period of time t such that $v_{i-1} + t \models g_i$ and $v_i(x) = 0$ if $x \in Y_i$, $v_i(x) = v_{i-1} + t$ if $x \notin Y_i$. The Y_i define the resets on each transition. A *local run* is a run where (q_0, v_0) can be arbitrary. The label of the run is the timed word $w = (a_i, t_i)_{1 \leq i \leq n}$, also written $w = (a_i, \tau_i)_{1 \leq i \leq n}$ to use the relative time delays $\tau_i = t_i - t_{i-1}$ for $i > 1$ and $\tau_1 = t_1$. Such a run will be denoted by $(q_0, v_0) \xrightarrow{w} (q_n, v_n)$.

A timed word w is accepted by the timed automaton if there is such an accepting path π. The set of all finite timed words accepted by $\mathcal{A}$ is denoted by $L_f(\mathcal{A})$.

Given a timed word w, a factor is a subsequence $(a_j, \tau_j)_{j=i,i+1,\dots i+l}$ starting in position i. Its weight is $\sum_{j=i,\dots i+l} \tau_j$ and its relative weight is:

$$\frac{\sum_{j=i,\dots i+l} \tau_j}{\sum_{j=1,\dots n} \tau_j}$$

2.2 Region automata

Let X be a set of clocks and $\mathcal{C}$ be a finite subset $\mathcal{C}(X)$. A finite partitioning $\mathcal{R}$ of the set of valuations is a *set of regions* for the constraints $\mathcal{C}$ if the following compatibility conditions are satisfied:

1. $\mathcal{R}$ is *compatible with the constraints* $\mathcal{C}$: for every constraint $g \in \mathcal{C}$, and every $R \in \mathcal{R}$, either $[g] \subseteq R$ or $[g] \cap R = \emptyset$,
2. $\mathcal{R}$ is *compatible with elapsing of time*: for all $R, R' \in \mathcal{R}$, if there exists some $v \in R$ and $t \in \mathbb{R}^+$ such that $v + t \in R'$, then for every $v' \in R$, there exists some $t' \in \mathbb{R}^+$ such that $v' + t' \in R'$,
3. $\mathcal{R}$ is *compatible with resets*: for all $R, R' \in \mathcal{R}$, for every subset $Y \subseteq X$, if $[Y \leftarrow 0]R \cap R' \neq \emptyset$, then $[Y \leftarrow 0]R \subseteq R'$.

$\mathcal{R}$ defines an equivalence relation $\equiv_{\mathcal{R}}$ over valuations: $v \equiv_{\mathcal{R}} v'$ if for each region R of $\mathcal{R}$, $v \in R \iff v' \in R$. From a set of regions $\mathcal{R}$ one can define the time-successor relation: a region R' is a time-successor of a region R if for each valuation $v \in R$, there exists a $t \in \mathbb{R}^+$ such that $v + t \in R'$.

Let $\mathcal{A}$ be a timed automaton with a set of constraints $\mathcal{C}$ and $\mathcal{R}$ be a finite set of regions for $\mathcal{C}$. The *region automaton* $\mathcal{A}_{\mathcal{R}}$ is the finite automaton defined by:

- the set of states is $Q \times \mathcal{R}$,
- the initial states are $I \times \{R_0\}$, where R_0 is the region containing the valuation assigning 0 to each clock,
- the final states are $F \times \mathcal{R}$,
- there is a transition $(q, R) \xrightarrow{g,a,Y} (q', R')$ whenever there exists a transition $q \xrightarrow{g,a,Y} q'$ in $\mathcal{A}$ and a region R'' which satisfies g and $R' = [Y \leftarrow 0]R''$.

Alur and Dill have shown [2] how to construct a set of regions, and the size of the region automaton is exponential in the number clocks. As $\mathcal{A}_{\mathcal{R}}$ is a finite automaton, for every timed automaton $\mathcal{A}$ for which we can construct a set of regions, we can decide reachability properties using the region automaton construction. For a run of the automaton $\mathcal{A}$ of the form:

$$(q_0, v_0) \xrightarrow{a_1,t_1} (q_1, v_1) \xrightarrow{a_2,t_2} (q_2, v_2) \dots \xrightarrow{a_n,t_n} (q_n, v_n)$$

let its projection be the sequence

$$(q_0, R_0) \xrightarrow{a_1} (q_1, R_1) \xrightarrow{a_2} (q_2, R_2) \dots \xrightarrow{a_n} (q_n, R_n)$$

where $(R_i)_{1 \leq i \leq n}$ is the sequence of regions such that $v_i \in R_i$ for each $1 \leq i \leq n$. From the definition of the transition relation for $\mathcal{A}_{\mathcal{R}}$, it follows that the projection is a run of $\mathcal{A}_{\mathcal{R}}$ over $(a_i)_{1 \leq i \leq n}$.

Given some connected component C of the region automaton, a timed word u is *C-compatible* if there exists a local run $(q, v) \xrightarrow{u} (q', v')$ such that its projection is in C. Let $L_f(C)$ be the set of timed words u which are C-compatible.

The membership of *timed words* for timed automata is an NP-complete problem [3,5], when we consider the simple constraints used in our definition.

2.3 Robustness for timed systems

Timed automata assume perfect clocks and perfect precision. The relaxation to a robust acceptance has been introduced in [16] where the reachability is undecidable. Imperfect clocks with a drift are considered in [20,4], and uncertainty in the guards is introduced in [12].

A survey of the robustness in timed automata is presented in [10]. We introduce a different approach, with a natural distance between timed words which extends to a distance between a timed word and a language L of timed words. We then show that the approximate membership problem becomes easy, in this setting. Precisely, we provide an $O(1)$ algorithm using an approximate decision.

2.4 Thin and thick components

For a strongly connected component C, a *limit cycle* is a local run such that its projection is in C and there is a state (q, v) and a timed word w of positive length such that $(q, v) \xrightarrow{w} (q, v)$. A *progress cycle* is a cycle where every clock is reset on some edge of the cycle. A *forgetful cycle* π_f is a subcase where for all state (q, R) on the cycle, for all $v, v' \in R$ we can find a word w such that:

$$(q, v) \xrightarrow{w} (q, v')$$

By extension, given two states (q, R) and (q', R') of the region automaton, we can use the forgetful cycle to link two states $(q, v), v \in R$ and $(q', v'), v' \in R'$. Connect first (q, R) to the forgetful cycle π_f, follow π_f and then connect to (q', R'). We can then connect $(q, v), v \in R$ and $(q', v'), v' \in R'$.

In [7], the existence of a forgetful cycle in a connected component is proved to be equivalent to the existence of a limit cycle, introduced in [20]. Such components are called *thick* and *thin* components are not thick. We assume that all the connected components are thick, i.e. admit forgetful cycles, and we use this hypothesis in a fundamental way.

3 Property Testing

For approximate decision problems, the approximation is applied to the input and suppose a distance between input structures. An ϵ-tester for a property P accepts all inputs which satisfy the property and rejects with high probability all inputs which are ϵ-far from inputs that satisfy the property. The approximation on the input was implicit in *Program Checking* [8,9,21], in *Probabilistically Checkable Proofs* (PCP) [6], and explicitly studied for graph properties under the context of property testing [15].

These restrictions allow for sublinear algorithms and even $O(1)$ time algorithms, whose complexity only depends on ϵ. Let $\mathbf{K}$ be a class of finite structures with a normalized distance dist between structures, i.e. dist lies in $[0, 1]$. For any $\epsilon > 0$, we say that $U, U' \in \mathbf{K}$ are *ϵ-close* if their distance is at most ϵ. They are *ϵ-far* if they are not ϵ-close. In the classical setting, the satisfiability of a property P is the decision problem whether U satisfies P for a structure $U \in \mathbf{K}$ and a property $P \subseteq \mathbf{K}$. A structure $U \in \mathbf{K}$ *ϵ-satisfies* P, or U is ϵ-close to $\mathbf{K}$ if U is ϵ-close to some $U' \in \mathbf{K}$ such that U' satisfies P. We say that U is ϵ-far from $\mathbf{K}$ if U is not ϵ-close to $\mathbf{K}$.

Definition 1 (Property Tester [15]) Let $\epsilon > 0$. An *ϵ-tester* for a property $P \subseteq \mathbf{K}$ is a randomized algorithm $A(\epsilon)$ such that, for any structure $U \in \mathbf{K}$ as input:
(1) If U satisfies P, then $A(\epsilon)$ accepts;
(2) If U is ϵ-far from P, then $A(\epsilon)$ rejects with probability at least $2/3$.[1]

A *query* to an input structure U depends on the model for accessing the structure. For a word w, a query asks for the value of $w[i]$, for some i. For a tree T, a query asks for the value of the label of a node i, and potentially for the label of its j-th successors, for some j. For a dense graph a query asks if there exists an edge between nodes i and j. The *query complexity* is the number of queries made to the structure. The *time complexity* is the usual definition, where we assume that the following operations are performed in constant time: arithmetic operations, a uniform random choice of an integer from any finite range not larger than the input size, and a query to the input.

[1] The constant $2/3$ can be replaced by any other constant $0 < \gamma < 1$ by iterating $O(\log(1/\gamma))$ the ϵ-tester and accepting iff all the executions accept.

Definition 2 A property $P \subseteq \mathbf{K}$ is *testable*, if there exists a randomized algorithm A such that, for every real $\epsilon > 0$ as input, $A(\epsilon)$ is an ϵ-tester of P whose query and time complexities depend only on ϵ (and not on the input size).

Property testing of regular languages was first considered in [1] for the *Hamming distance*, where the Hamming distance between two words is the minimal number of character substitutions required to transform one word into the other. The (normalized) edit distance between two words (resp. trees) of size n is the minimal number of insertions, deletions and substitutions of a letter (resp. node) required to transform one word (resp. tree) into the other, divided by n.

The testability of regular languages on words and trees was studied in [17] for the edit distance with *moves*, that considers one additional operation: moving one arbitrary substring (resp. subtree) to another position in one step. This distance seems to be more adapted in the context of property testing, since their tester is more efficient and simpler than the one of [1], and can be generalized to tree regular languages. A statistical embedding of words which has similarities with the Parikh mapping [19] was developped in [14]. This embedding associates to every word a sketch of constant size (for fixed ϵ) which allows to decide any property given by some regular grammar or even some context-free grammar.

In this paper we introduce a new distance on timed words and apply the property testing framework with this distance to the membership problem of timed automata.

3.1 Timed edit distance

The classical *edit distance* on words is a standard measure between two words w and w'. An *edit* operation is a *deletion*, an *insertion* or a *modification of a letter*. The *absolute edit distance* is the minimum number of edit operations to transform w into w' and the *relative edit distance* is the absolute edit distance divided, by $Max(|w|, |w'|)$. We mainly use the relative distance, a value between 0 and 1.

Consider the *timed edit* operations:

- Deletion of (a, τ) has cost τ,
- Insertion of (a, τ) has cost τ,
- Modification of (a, τ) into (a, τ') has cost $|\tau - \tau'|$.

A transformation is a sequence of operations which transform w into w', and the total cost D is the sum of the elementary costs. The *absolute timed edit distance* $\mathsf{Dist}(w, w')$ between two timed words w and w' is the minimal total cost over all possible transformations from w into w'.

Definition 3 The *relative timed edit distance* between two timed words $w = (a_i, \tau_i)_{1 \leq i \leq n}$ and $w' = (a'_i, \tau'_i)_{1 \leq i \leq n'}$, is :

$$\mathsf{dist}(w, w') = \frac{1}{2} \cdot \frac{\mathsf{Dist}(w, w')}{Max(\sum_{i=1,\dots n} \tau_i, \sum_{i=1,\dots n'} \tau'_i)}$$

If T is the maximum time of w of w', $\mathsf{dist}(w, w')$ is also $\frac{D}{2 \cdot T}$. Two words w, w' are ϵ-close if $\mathsf{dist}(w, w') \leq \epsilon$. The distance between a word w and a language L of timed words is defined as $\mathsf{dist}(w, L) = Min_{w' \in L} \mathsf{dist}(w, w')$.

Examples. The absolute distance between $(a, 10)$ and $(a, 13)$ is 3. The absolute distance between $(a, 10)$ and $(b, 10)$ is 20. The absolute distance between $(a, 1), (a, 100)$ and $(a, 100), (a, 1)$ is 2.

To the best of our knowledge, the *relative timed edit distance* is a new distance, although other distances have been considered in the context of words with weights.

3.2 Other distances

The edit distance has been generalized to a weighted edit distance where a fixed weight is associated to each letter and to each pair of letters. The cost of an insertion or deletion of a letter is the weight of the letter and the cost of a modification of a by b is the cost of the pair (a, b). For the timed edit distance, the costs are not fixed and depend on the time.

In the context of timed words, [5] introduces a metric on timed words: for two words w, w' of length n such that $untime(w) = untime(w')$, let $\mathsf{dist}(w, w') = max\{|t_i - t'_i|, 0 \leq i \leq n\}$ where t_i is the absolute time. In [11], this distance is generalized to a vector whose first component captures the classical edit distance and the second component measures the maximum difference of the time intervals. It emphasizes the classical edit distance between the words. As an example, the distance of [11] between the timed words $w = (a, 1), (a, 100)$ and $w' = (a, 100), (a, 1)$ is the vector $(0, 99)$ as the edit distance is 0 and the maximum time difference is 99. In our framework, the absolute timed edit distance is 2: we remove $(a, 1)$ of the first timed word at the cost 1 and add it after $(a, 100)$ for the same cost.

Another distance based on time intervals has been introduced in [13]. A generalization of the classical edit distance to weighted automata has been introduced in [18].

3.3 Algorithm for the timed edit distance

The *absolute timed edit distance* between two words w_1, w_2 is computable in polynomial time by just generalizing the classical algorithm [22] for the edit distance. Let $A(i, j)$ be the array where w_1 appears on the top row ($i = 1$) starting with the empty character ϵ, w_2 appears on the first column starting with the empty character ϵ as in Figure 1. For each letter w, let $w(i)$ be the relative time τ_i. The value $A(i, j)$ for $i, j > 1$ is the absolute timed edit distance between the prefix of w_1 of length $j - 2$ and the prefix of w_2 of length $i - 2$. Let $\Delta(i, j) =\mid \tau(i) - \tau(j) \mid$ be the time difference between $w_1(i)$ and $w_2(j)$ if the letter symbols are identical, ∞ otherwise. It is the timed edit distance between two letters.

For $i, j > 1$, there is a simple recurrence relation between $A(i, j), A(i - 1, j), A(i, j - 1)$ and $A(i - 1, j - 1)$, which reflects 3 possible transformations: deletion of $w_1(i - 2)$, deletion of $w_2(j - 2)$ or edition of the last letters. Hence:

$$A(i, j) = Min\{A(i, j - 1) + w_1(i - 2), A(i - 1, j) + w_2(j - 2), A(i - 1, j - 1) + \Delta(i, j)\}$$

In the example of Figure 1, the absolute timed edit distance is 10, and we can trace the correct transformations by tracing the minimum for each $A(i, j)$: in this case, we erase $(a, 5)$ and reinsert it at the right place.

4 Testing membership of timed words

Given a timed word $w = (a_i, t_i)_{1 \leq i \leq n}$, we want to *approximately* decide if w is in language L, i.e. decide if w is accepted or if w is ϵ-far from a language L, for the timed edit distance, i.e. if $\mathsf{dist}(w, L) \geq \epsilon.T$ for $T = t_n$. A query is specified by a weight k and returns a factor of the word w of weight at least k taken according to the distribution μ, introduced in section 4.1. This is the classical approximation taken in Property Testing.

w_1

w_2	ε	(a,2)	(b,4)	(a,5)
ε	0	2	6	11
(a,5)	5	3	7	6
(a,2)	7	5	9	8
(b,4)	11	9	5	10

Fig. 1 Classical array $A(i,j)$ for timed edit distance between $w_1 = (a,2),(b,4),(a,5)$ and $w_2 = (a,5),(a,2),(b,4)$

Assume the region automaton $\mathcal{A}_R$ has some strongly connected components C_i for $i = 1,...,p$ and some transient states s_j. Let $\Pi = s_0.s_1.s_2.C_{i_1}.s_3, s_4.C_{i_2}....C_{i_l}$ be a path in the graph of the connected components of the region automaton from the initial transient state s_0. Two components C_i and C_j are connected if there is a transition in the region automaton between a state $(q,R) \in C_i$ and $(q',R') \in C_j$.

For example $\Pi = s_0.s_1.s_2.C_1.s_3, s_4.C_2.C_3$ is a sequence of transient states and connected components. Let us specify the state used in each component C_i when a run leaves the component C_i. It is a distinguished state $\bar{s^i} \in C_i$ and we write $C_i[\bar{s^i}]$:

$$\Pi = s_0.s_1.s_2.C_1[\bar{s^1}].s_3, s_4.C_2[\bar{s^2}].C_3$$

Let us define an *extended component* $\bar{C_i}$ as a component C_i with a prefix of transient states and a distinguished state as a suffix. In our example, $\bar{C_1} = s_0.s_1.s_2.C_1[\bar{s^1}]$ and $\bar{C_2} = \bar{s^1}.s_3, s_4.C_2[\bar{s^2}]$ and $\bar{C_3} = \bar{s^2}.C_3$. The concatenation $\bar{C_1}.\bar{C_2} = s_0.s_1.s_2.C_1[\bar{s^1}].\bar{s^1}.s_3, s_4.C_2[\bar{s^2}] = s_0.s_1.s_2.C_1[\bar{s^1}].s_3, s_4.C_2[\bar{s^2}]$, i.e. $C_1[\bar{s^1}].\bar{s^1} = C_1[\bar{s^1}]$. Let $\bar{\Pi} = \bar{C_1}.\bar{C_2}....\bar{C_l}$ be the general sequence after the introduction of the extended components.

4.1 Samples and Compatibility

The *weighted time distribution* μ selects a position $1 \leq j \leq n$ in a word $w = (a_i,t_i)_{1\leq i\leq n}$, i.e. a letter (a_j,t_j), proportionally to its weight τ_j:

$$Prob_\mu[j] = \tau_j/t_n$$

To efficiently choose such a j, we choose a uniform real value $i \in_r [0,t_n]$ and find j such that $t_{j-1} \leq i < t_j$ by dichotomy. We first compare i and $t_{n/2}$ and find the exact j after at most $\log n$ steps. A k-sample of

$w = (a_i, t_i)_{1 \leq i \leq n} = (a_i, \tau_i)_{1 \leq i \leq n}$ is a factor u of weight $|u| = \sum_{i=j}^{i=j+l} \tau_i \geq k$ for the smallest l if it exists. If we reach the end of w, we select a k-sample of weight less than k. We use the absolute values to efficiently select a sample, and the relative values for its weight.

Given a word w, we select l independent samples of weight at least k according to μ, which we order as $(u_1, ...u_l)_\mu$. Notice that two samples u_i and u_j of weight k, where $i < j$, may overlap or may be identical. Given a timed automaton $\mathcal{A}$, and its region automaton, let Π be a sequence of transient states and connected components of the region automaton.

We now introduce the central notion of *compatibility* for an arbitrary sequence Π and a sequence of possibly overlapping ordered factors $(u_1, ...u_l)$ of a word w. As an example, we can write Π as:

$$\Pi = s_0.s_1.s_2.C_1[\bar{s^1}].s_3, s_4.C_2[\bar{s^2}].C_3$$

The associated sequence of extended components is:

$$\bar{\Pi} = \bar{C_1}.\bar{C_2}.\bar{C_3}$$

For each factor u_i we extend the definition of compatibility introduced in section 2.2 for a connected component C to a sequence Π. We examine if it could start from some transient state s_j or from some connected component C_j of Π and end on a transient state or on a connected component.

- A sample u_i is *compatible* for Π if:
  - A sample u_i is *compatible*, for example from transient state s_2 to transient state s_4 for

    $$\Pi = s_0.s_1.s_2.C_1[\bar{s_1}].s_3, s_4.C_2[\bar{s_2}].C_3$$

    if $\exists\ (q, v), (q', v') \quad (q, v) \xrightarrow{u_i} (q', v')$, $s_2 = (q, R)$, $v \in R$ and $s_4 = (q', R')$, $v' \in R'$. The first letter of u_i reaches a state of C_1, the last letter reaches s_4 from s_3, the letter before the last one reaches s_3 from the state $\bar{s_1} \in C_1$.
  - The sample u_i is *compatible* from transient state s_2 to a connected component C_2 if there is a state $(q', R') \in C_2$ such that the previous definition is satisfied.
  - This definition can be generalized when the starting state is either a transient state or a component C_j.
- If two factors u_i, u_{i+1} overlap, consider their union, i.e. the largest factor which contains both. It must be compatible for Π.
- The sequence of ordered factors $(u_1, ...u_l)$, where $l \leq m$, with possible overlap of a word w *is Π-compatible* if each u_i is compatible for Π from some state or component to some other state or component. The order of these states follows the order of Π.

Let $L_f(\Pi)$ be the set of timed words u which are Π-compatible, the *language of* Π, and similarly for $L_f(\bar{\Pi})$.

Definition 4 The sequence of ordered factors $(u_1, ...u_l)$ with possible overlap of a word w is compatible with a timed automaton $\mathcal{A}$ if there exists a sequence Π such that $(u_1, ...u_l)$ is Π-compatible.

The tester will take l samples $(u_1, ...u_l)$, each u_i is of weight at least k, which we order according to their position in w.

4.2 Compatibility properties

Let w be a timed word accepted by a timed automaton $\mathcal{A}$. What can be said about the compatibility of samples $(u_1, ...u_l)$?

Lemma 1 *If $w \in L_f(\mathcal{A})$, then for all l and for all ordered l-samples $(u_1, ...u_l)$ there is a sequence Π such that $(u_1, ...u_l)$ is Π-compatible.*

Proof If $w \in L_f(\mathcal{A})$, there is a run for w, i.e. a sequence Π defined by the run from the origin state to some final state q. If two samples overlap, consider their union. The independent samples are Π-compatible.

Consider the following decision procedure to decide if $(u_1, ...u_l)$ is Π-compatible. We first enumerate all pairs (s_i, s_i') such that u_i is compatible for Π from s_i to s_i' and then use algorithm A_1. Let u_i be a factor and Π a sequence:

Algorithm $A_1(u_i, \Pi)$. *Enumerate all pairs (s_i, s_i') where s_i and s_i' are either a transient state of Π or a state of a connected component and accept if u_i is compatible for Π from s_i to s_i'.*

A_1 solves a system of linear constraints for each (s_i, s_i') where the variables are the valuations on the states from s_i to s_i'. We accept if there is a solution to the system and reject otherwise. We extend A_1 to A_2 which takes $(u_1, ...u_l)$ the ordered samples as input, instead of a single u_i.

Compatibility Algorithm $A_2((u_1, ...u_l), \Pi)$. *Choose for each u_i some pair (s_i, s_i') using Algorithm $A_1(u_i, \Pi)$ and accept if the choices of the s_i, s_i' witness the compatibility of $(u_1, ...u_l)$ for Π, i.e. follow the order of Π with the overlaping condition.*

4.3 Tester

Given a timed automaton $\mathcal{A}$ let $L_f(\mathcal{A})$ be the language accepted. Let $\mathcal{A}_\mathcal{R}$ be the region automaton with m states, and let B be the maximal value used in the time constraints. We first generate all the maximal Π, for the inclusion, which include connected components and transient states and write $\bar{\Pi} = \bar{C_{i_1}}.\bar{C_{i_2}}....\bar{C_{i_l}}$ the corresponding sequences of extended components. We first define a Tester along a $\bar{\Pi}$ and the final Tester considers all possible maximal $\bar{\Pi}$.

Tester along a path $\bar{\Pi} = \bar{C_{i_1}}.\bar{C_{i_2}}....\bar{C_{i_l}}$
Input: timed word w, ϵ,
Output: Accept or Reject

1. Sample l independent factors $(u_1 < u_2.... < u_l)$ of weight $k \geq 24l.m.B/\epsilon$ of w for the weighted time distribution μ. If two samples overlap, we merge them.
2. Accept if $A_2((u_1, ...u_l), \Pi)$ accepts else Reject.

We then obtain the general tester for a regular timed language $L_f(\mathcal{A})$.

Tester for $L_f(\mathcal{A})$
Input: word w, ϵ
Output: Accept or Reject

1. Construct all the $\bar{\Pi}$ corresponding to maximal Π, of the region automaton $\mathcal{A}_{\mathcal{R}}$, starting in the initial state,
2. For each $\bar{\Pi}$, apply the Word Tester along $\bar{\Pi}$,
3. If there is a $\bar{\Pi}$ such that Word Tester along $\bar{\Pi}$ accepts, then Accept else Reject.

4.4 Analysis

We have to verify the two properties of a Tester, given in definition 1.

Lemma 2 *If $w \in L_f(\mathcal{A})$ then the Word Tester for $L_f(\mathcal{A})$ always accepts.*

Proof Consider a run of the automaton $\mathcal{A}$, labeled by w. There exists a $\bar{\Pi}$ with at most l extended components such that all the factors u of weigth k of w are compatible for Π. Any sequence of ordered factors $(u_1, ...u_l)$ is also Π-compatible. Hence the Tester accepts.

The more difficult task is to show that if w is ϵ-far from $L(\mathcal{A})$ then the Tester will reject with constant probability. Equivalently, we could show the contrapositive, i.e. if the Tester accepts with constant probability, then w is ϵ-close to $L(\mathcal{A})$.

We construct a *corrector* for w in order to prove this property. A corrector is a non deterministic process which transforms an incorrect w into a correct w' with timed edit operations. We first describe a corrector for a given component C in section 4.4.1, for an extended component $\bar{C}$ in section 4.4.2 and for a path $\bar{\Pi} = \bar{C_1}, ...\bar{C_l}$ of extended components in section 4.4.3. In the last case, we decompose a word w into l factors which we will be correct for each $\bar{C_i}$.

4.4.1 Corrector for a component C

C is a thick component, i.e. admits a forgetful cycle [7]. In this case, from a state (q, R) and $v \in R$, we can reach any (q', R') and $v' \in R'$ with a small timed word, called a *link* in lemma 3. We then introduce a decomposition of a word w into compatible fragments separated by *weighted cuts*. In lemma 4, we show that if the total relative weight of the cuts is small, then the word is ϵ-close to $L_f(C)$. In lemma 5 we show that if w is ϵ-far, samples of weight at least k, a function of ϵ, are incompatible for $L_f(C)$ with constant probability.

Lemma 3 *For all pairs of states $(q, R), (q', R')$ of a thick component C and for all valuations $v \in R, v' \in R'$, there exists a timed word σ such that $(q, v) \xrightarrow{\sigma} (q', v')$.*

Proof As C is a thick connected component, there is a forgetful cycle π and a path from (q, R) to a state (q_0, R_0) on the cycle π and a path from (q_0, R_0) to (q', R'). For all valuations $v \in R$ there is a $v_0 \in R_0$ and a timed word σ_0 such that $(q, v) \xrightarrow{\sigma_0} (q_0, v_0)$ and similarly for all valuations $v' \in R'$ there is a $v'_0 \in R_0$ and a timed word σ_2 such that $(q_0, v'_0) \xrightarrow{\sigma_2} (q', v')$. As π is a forgetful cycle, there is a word σ_1 such that $(q_0, v_0) \xrightarrow{\sigma_1} (q_0, v'_0)$. Hence $\sigma = \sigma_0.\sigma_1.\sigma_2$ is the desired path, as $(q, v) \xrightarrow{\sigma} (q', v')$.

We decompose any word into compatible factors for the component C and introduce the notion of a weighted cut. The sum V of the weights of the different cuts is the key parameter of the decomposition.

Definition 5 A *weighted cut for C* in a timed word $w = (a_i, \tau_i)_{1 \leq i \leq n}$ is a decomposition of w into the longest possible compatible prefix w_1 and a distinct suffix w'_1, i.e. $w = w_1.(a_i, \tau_i).w'_1$, such that w_1 is compatible for C but $w_1.(a_i, \tau_i)$ is not compatible for C.

Notice that we start from an arbitrary state (q, v) of C where $v \in R$. We can't extend the run because of a_i, or τ_i or both. The position i which satisfies the conditions of the decomposition is also called a cut.

In order to define the *weight of a cut*, we distinguish between a *weak cut* and a *strong cut*. A cut is *strong* if the single letter (a_i, τ_i) is not compatible with C, i.e. there is no run for any state of C. A cut is *weak* if it is not strong. We repeat the analysis for w_1' and it is therefore important to find a local correction between w_1 in state (q, R) with some $v \in R$ and w_1' starting in state (q', R') with any $v' \in R'$. We correct the weak cuts using lemma 3.

- In a weak cut, let $(a_i, \tau_i).w_2$ be the longest compatible timed word from some state (q, R). Lemma 3 provides a link σ before the letter (a_i, τ_i). The edit cost is then $c =\mid \sigma \mid \leq 3mB$, because a path which includes a forgetful cycle is made of three segments of length at most m and therefore of cost less than $3mB$.
- In a strong cut, (a_i, τ_i) is not compatible but there may exist $\tau_i' \neq \tau_i$, such that (a_i, τ_i') is compatible. The correction cost is $c =\mid \tau_i - \tau_i' \mid$. If it is not the case, we erase the letter and the cost is $c = \tau_i$.

In a strong cut, the cost of the correction can be high, of the order of τ_i. The distinction is important, as a sample u which contains a strong cut is incompatible for C whereas a sample which contains a weak cut may be compatible. A starting state (q, R) for the sample u may define a possible run, although the corresponding factor in w contained a cut.

A sample which contains two consecutive weak cuts is incompatible. At the first cut we consider all possible states of C and choose the state for which the longest possible factor is compatible. All the runs block before the second cut or precisely at the second cut. Hence the sample is incompatible.

We then write: $w = w_1 \mid_c w_2$ where w_2 is w_1' of definition 5 and c is the cost of the correction. We iterate this process on w_2 starting in an arbitrary state (q, R) and $v \in R$. We write $w_2 = w_{2,1} \mid w_{2,2}$ and $w = w_1 \mid_{c_1} w_2 \mid_{c_2} w_3$ instead of $w = w_1 \mid_{c_1} w_{2,1} \mid_{c_2} w_{2,2}$. We obtain an algorithmic decomposition, written as:

$$w = w_1 \mid_{c_1} w_2 \mid_{c_2} w_3 \mid_{c_3} \mid \ldots \mid_{c_h} w_{h+1}$$

if there are h cuts of total value $V = \sum_{j=1\ldots h} c_j$, for the C decomposition of w. Each w_i is the largest compatible factor of w starting at the $(i-1)$-th cut to the ith cut.

Let $c_s = \sum_{\text{strong cut } j} c_j$ the total weight of the strong cuts and $c_w = \sum_{\text{weak cut } j} c_j$ the total weight of the weak cuts.

Lemma 4 *If w has cuts of total weight V for C, then w is $\frac{V}{T}$-close to the regular language $L_f(C)$.*

Proof Lemma 3 indicates that we can always correct a cut and start from an arbitrary new state. For each cut i of weight c_i, we have a correction of weight at most c_i. Hence a total relative distance of $\frac{V}{T}$ to the language of C.

By the contraposition of lemma 4, if w of weight T is ϵ-far from C, then $V \geq \epsilon.T$, i.e. large.

Suppose w is ϵ-far from C and we take some sample u of weight at least $2k$ with the weighted time distribution μ. We want to show that the probability that u is incompatible is a constant independent of T, which only depends on ϵ and on the automaton. We will prove that for factors u of w of weight greater than $2k$, a large proportion will be incompatible. Let α_i be the number of w_j in the decomposition of w along the cuts, whose weight is larger than 2^{i-1} and less than 2^i, i.e. :

$$\alpha_i = |\{w_j : \ 2^{i-1} \leq |w_j| < 2^i\}|$$

where $|w_j|$ is the weight of w_j. By definition $h = \sum_i \alpha_i$ is the total number of cuts. We need to find a bound i_l such that the sum of the costs of the associated cuts satisfies:

$$\sum_{0 \leq |w_i| \leq 2^{i_l}} c_i \geq \epsilon.T$$

It will guarantee that samples of weight $2k$ where $k = 2^{i_l}$ will contain incompatible factors with constant probability when we take a sample according to the weighted time distribution.

Lemma 5 *If w is ϵ-far from the language of C, there exists $k = 24mB/\epsilon$ such that a sample of weight $2k$ from the weighted time distribution μ is such that: either the probability to find an incompatible unit factor (strong cut) is greater than $\epsilon/2$ or the probability to find an incompatible factor (weak cut) is greater than $\delta = 3.\epsilon^2/5$.*

Proof By the contraposition of lemma 4, if w is ϵ-far from C, then $V \geq \epsilon.T$, i.e. large. Consider the two cases determining whether c_s is large or whether c_w is large.

Case 1. The strong cuts are dominant, i.e. $c_s \geq V/2 \geq \epsilon.T/2$. In this case if we take a sample of one element according to the time distribution, it is incompatible with probability larger than $\epsilon/2$.

Case 2. The weak cuts are dominant, i.e. $c_w \geq V/2 \geq \epsilon.T/2$. In this case each cut has a weight less than $3mB$. Hence the number h of cuts is larger than $\epsilon.T/6mB$.

We need to bound the weight of the samples u which guarantees that a sample contains a w_i, hence two cuts, with constant probability. In this case u is incompatible. Recall that $h = \sum_i \alpha_i \geq \epsilon.T/6mB$.

We need to estimate $\sum_{0 \leq i_\leq i_l} \alpha_i$ when we choose $k = 2^{i_l}$, as we would bound the number of w_i of weight smaller than k. Let

$$\beta = \sum_{i \geq i_l} \alpha_i \qquad \gamma = \sum_{i < i_l} \alpha_i$$

First let us estimate β, i.e. for large i. There are at most $T/2^{i_l}$ feasible w_j of weight larger than 2^{i_l}, i.e. $\beta \leq T/2^{i_l}$. Hence $\sum_i \alpha_i = \gamma + \beta \geq \epsilon.T/6mB$ and

$$\gamma \geq \epsilon.T/6mB - \beta \geq \epsilon.T/6mB - T/2^{i_l}$$

Let $k = 24mB/\epsilon = 2^{i_l}$, or $i_l = \log(24mB/\epsilon)$. Then

$$\beta = \sum_{i \geq i_l} \alpha_i \leq T/2^{i_l} \leq \epsilon.T/24mB$$

$$\gamma \geq T/6mB - \beta \geq \epsilon.T/6mB - \epsilon.T/24mB = \epsilon.T/8mB$$

$$\gamma \geq 3.\beta$$

Le us show that if we take w_j with the weighted time distribution, then:

$$Prob[|w_j| \leq k] \geq 4\epsilon/5 \qquad (1)$$

$$Prob[|w_{j+1}| \leq k \ \mid \ |w_j| \leq k] \geq 3\epsilon/4 \qquad (2)$$

We argue with different possible corrections. For the first inequality (1), consider the correction where we erase all the small blocks, at a cost of $Prob[|w_j| \leq k].T$ and then correct at most β large blocks, at a cost of $\beta.3mB$. As the word is ϵ-far, then:

$$Prob[|w_j| \leq k].T + \beta.3mB \geq \epsilon.T$$

$$Prob[|w_j| \leq k].T \geq \epsilon.T - \beta.3mB \geq \epsilon.T - \epsilon.T.3mB/24mB = 7\epsilon.T/8 \geq 4\epsilon.T/5$$

$$Prob[|w_j| \leq k] \geq 4\epsilon/5$$

For the second inequality (2), consider the situation where we erase all the small blocks which have a small predecessor. There remains small blocks followed by large blocks. As $\gamma \geq 3\beta$, in the worst case there are β remaining small blocks. We must therefore correct at most 2β cuts, for the large blocks and the remaining small blocks. The cost of the erasure is $Prob[|w_{j+1}| \leq k \mid |w_j| \leq k].T$ and the correction cost is at most $2\beta.3mB$. As the word is ϵ-far, then:

$$Prob[|w_{j+1}| \leq k \mid |w_j| \leq k].T + 2\beta.3mB \geq \epsilon.T$$

$$Prob[|w_{j+1}| \leq k \mid |w_j| \leq k].T \geq \epsilon.T - 2\beta.3mB$$

$$Prob[|w_{j+1}| \leq k \mid |w_j| \leq k].T \geq \epsilon.T - 2\epsilon.T.3mB/24mB$$

$$Prob[|w_{j+1}| \leq k \mid |w_j| \leq k].T \geq 3\epsilon.T/4$$

We can then bound the probability that a sample of weight $2k$ is incompatible: it is greater than the probability that a sample u contains 2 successive small blocks.

$$Prob[\text{sample } u \text{ of weight } 2k \text{ is incompatible}] \geq Prob[|w_j| \leq k].Prob[|w_{j+1}| \leq k \mid |w_j| \leq k]$$

$$Prob[\text{sample } u \text{ of weight } 2k \text{ is incompatible}] \geq (4\epsilon/5).(3\epsilon/4) \geq \delta = 3\epsilon^2/5$$

We need to generalize this argument to a sequence $\bar{\Pi}$ of extended components.

4.4.2 Correction for extended components

Let (q, R) be a transient state which appears in Π. Let $w_1.(a_i, \tau_i).w'$ be the word w where we read (a_i, τ_i) in state (q, R) with the value $v \in R$. Assume w_1 is compatible for a prefix π of Π but $w_1.(a_i, \tau_i)$ is not compatible for the prefix π followed by the transient state (q, R).

We can always insert (a'_i, τ'_i) such that $\tau'_i \leq B$ before (a_i, τ_i). We may also modify τ_i into τ'_i and not changing a_i if it is possible.

Lemma 6 *The correction cost for a transient state is less than B.*

Proof If we can modify τ_i into τ'_i without changing a_i at a cost less than B, we do it. Otherwise we insert (a'_i, τ'_i) such that $\tau'_i \leq B$ before (a_i, τ_i). In both cases, the correction cost is less than B.

We now generalize the correction for an extended component. An extended component $\bar{C}$, introduced at the beginning of section 4, has a prefix of transient states, a connected component C and a specific output state in C. We correct a word w for $\bar{C}$ by first correcting for the transient states, then for the connected component until we reach the output state of C.

4.4.3 Decomposition strategy for a sequence $\bar{\Pi}$ of 2 extended components

We now define a correction strategy for a sequence $\bar{\Pi}$ and consider the case $\bar{\Pi} = \bar{C_1}, \bar{C_2}$, as in Figure 2, which we can later generalize to an arbitrary $\bar{\Pi}$, for a word which is ϵ-far from $L_f(\bar{\Pi})$. The decomposition splits the word into 2 parts and we apply the correction strategy for $\bar{C_1}$ on the first part and the correction strategy for $\bar{C_2}$ on the second part. The *border* is the precise position of the partition (I_1, I_2) of the word w.

- Start in the initial state of $\bar{C_1}$ and take the longest compatible prefix w_1. It determines a cut of weight c which we correct with the previous corrector for $\bar{C_1}$. We continue in a similar way and accumulate the costs of the correction. If we reach a cut whose total weight V_c is at least $\epsilon.T/4$ and at most $3\epsilon.T/4$ in $\bar{C_1}$, the *border* is the position of this cut.
- If there are no cuts in the interval $[\epsilon.T/4, 3\epsilon.T/4]$, we necessarily read a large letter (a_i, τ_i) of weight larger than $\epsilon.T/2$, i.e. a strong cut. This letter is incompatible for $\bar{C_1}$ and for $\bar{C_2}$, otherwise the word would not be ϵ-far. The *border* is the position at the end of this large letter.

After we reach the border, we correct the cuts for $\bar{C_2}$. If we reach $\epsilon.T$ errors, we say that *the decomposition saturates* w for $\bar{\Pi} = \bar{C_1}, \bar{C_2}$. The goal is to guarantee that at least $\epsilon.T/4$ error occurs for $\bar{C_1}$ in I_1 and for $\bar{C_2}$ in I_2 if the word w is ϵ-far.

Lemma 7 *If a timed word w is ϵ-far from the language of $\bar{\Pi} = \bar{C_1}, \bar{C_2}$ then there is a border and the decomposition saturates w.*

Proof By contraposition, we consider two cases. If there is no border, then w is ϵ close to $\bar{C_1}$ hence to $\bar{C_1}, \bar{C_2}$. If the decomposition does not saturate then w is close to $\bar{C_1}, \bar{C_2}$, as we can find a correction of total cost less than $\epsilon.T$.

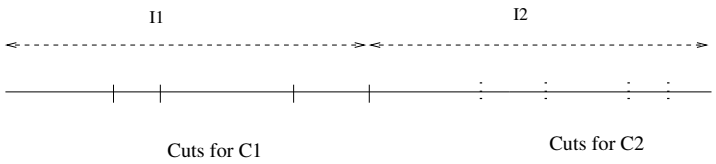

Fig. 2 A possible decomposition of w into feasible components for $\bar{\Pi} = \bar{C_1}, \bar{C_2}$.

If w is ϵ-far from $\bar{\Pi} = \bar{C_1}, \bar{C_2}$, then there are many samples u's of weight $2k$ incompatible for C_1 in the interval I_1 and many samples u's of weight $2k$ incompatible for C_2 in the interval I_2. We conclude, in lemma 8 below, that the Tester will reject with constant probability. It is important that both intervals I_1, I_2 are not empty. If a letter of the word is of weight larger than $\epsilon.T/2$, for example if we read one large incompatible letter, I_2 is empty and we need to treat this case separetely.

Lemma 8 *If w is ϵ-far from the language of $\bar{\Pi} = \bar{C_1}, \bar{C_2}$ and $k = 48mB/\epsilon$, then the Tester along $\bar{\Pi}$ rejects with constant probability.*

Proof Assume w is ϵ-far from $\bar{\Pi} = \bar{C_1}, \bar{C_2}$ and each letter is of weight at most $\epsilon.T/2$: we have a decomposition (I_1, I_2). Consider two samples $u_1 < u_2$ taken independently of weight at least $2k$. By construction they do not overlap. If u_1 is incompatible for $\bar{C_1}$ and u_2 is incompatible for $\bar{C_2}$, then the Tester rejects. Hence:

$$Prob[\text{Tester rejects}] \geq Prob[u_1 \text{ incompatible for } \bar{C_1} \wedge u_2 \text{ incompatible for } \bar{C_2}] \geq$$

$$Prob[(u_1 \in I_1 \wedge u_1 \text{ incompatible for } \bar{C_1}) \wedge (u_2 \in I_2 \wedge u_2 \text{ incompatible for } \bar{C_2}]$$

These two events are independent, hence we can rewrite the expression as:

$$Prob[(u_1 \in I_1 \wedge u_1 \text{ incompatible for } \bar{C_1})].Prob[(u_2 \in I_2 \wedge u_2 \text{ incompatible for } \bar{C_2}]$$

Let $\epsilon' = \epsilon/2$. From the lemma 5, if $k = 24mB/\epsilon' = 48mB/\epsilon$, then

$$Prob[u_1 \text{ incompatible for } \bar{C_1})|u_1 \in I_1] \geq 3\epsilon'^{\,2}/5 = 3\epsilon^2/20$$

and

$$Prob[u_1 \in I_1] \geq (\epsilon/4)$$

Hence:

$$Prob[(u_1 \in I_1 \wedge u_1 \text{ incompatible for } \bar{C_1})] \geq (\epsilon/4).3\epsilon^2/20$$

and similarly for u_2. Hence:

$$Prob[\text{Tester rejects}] \geq (3\epsilon^3/80)^2$$

If some letter is of weight greater than $\epsilon.T/2$, the decomposition (I_1, I_2) may be impossible. This large letter is incompatible for $\bar{C_1}$ and for $\bar{C_2}$. The tester will detect it with probability $\epsilon/2$.

4.4.4 Decomposition strategy for an arbitrary sequence $\bar{\Pi}$ of length l

The decomposition generalizes to $\bar{\Pi} = \bar{C_1}....\bar{C_l}$ by taking cuts for $\bar{C_1}$ of weight approximately $\epsilon.T/l$, cuts for $\bar{C_2}$ of weight at least $\epsilon.T/l$ until possible cuts for $\bar{C_l}$ of weight at least $3\epsilon.T/4l$. We consider small intervals of length $\epsilon.T/2l$: the first interval is $[3\epsilon.T/4l, 5\epsilon.T/4l]$ around $\epsilon.T/l$ and similarly for the other intervals. The notion of a *border* introduced in the previous section, generalizes to l intervals. It provides a decomposition $(I_1, I_2, ...I_l)$ into consecutive intervals $I_1, I_2, ...I_l$ of length at least $\epsilon.T/2.l$. We say that the *decomposition saturates* w if we reach $\epsilon.T$ errors.

The decomposition is not possible if one of the segment I_j is empty. It may occur when w is ϵ-far from the language of $\bar{\Pi}$, if there are less than l large letters with at least one of them of weight at least $\epsilon.T/l$ which is incompatible for all $\bar{C_1}....\bar{C_l}$.

Lemma 9 *If a timed word w is ϵ-far from the language of $\bar{\Pi} = \bar{C_1}....\bar{C_l}$ then there are $l-1$ borders and the decomposition saturates w.*

Proof By contraposition, we consider two cases. If there are less than $l-1$ borders, then w is ϵ-close to a prefix of $\bar{\Pi}$ hence to $\bar{\Pi}$. If the decomposition does not saturate then w is close to $\bar{\Pi}$, as we can find a correction of total cost less than $\epsilon.T$.

Theorem 1 *If w is ϵ-far from the language of $\bar{\Pi} = \bar{C_1}...\bar{C_l}$ and $k = 24l.mB/\epsilon$, then the Tester along the path $\bar{\Pi}$ rejects with constant probability.*

Proof Assume w is ϵ-far from $\Pi = \bar{C_1}....\bar{C_l}$ and there is a decomposition $(I_1, I_2, ...I_l)$ with non-empty segment. By lemma 9, the decomposition saturates w. Consider l samples $u_1 < u_2 < .... < u_l$ taken independently of weight at least $2k$ which do not overlap. If u_1 is incompatible for $\bar{C_1}$ and u_2 is incompatible for $\bar{C_2}$.... and u_l is incompatible for $\bar{C_l}$, then the Tester rejects. Hence:

$$Prob[\text{Tester rejects}] \geq Prob[u_1 \text{ incompatible for } \bar{C_1} \wedge u_2 \text{ incompatible for } \bar{C_2}....$$

$$\wedge u_l \text{ incompatible for } \bar{C_l}]$$

$$Prob[(u_i \text{ incompatible for } \bar{C_i}) \geq Prob[(u_i \in I_i \wedge u_i \text{ incompatible for } \bar{C_i})]$$

All these events are independent, hence we can rewrite the expression as:

$$\prod_i Prob[(u_i \in I_i \wedge u_i \text{ incompatible for } \bar{C_i})]$$

Let $\epsilon' = \epsilon/l$. From the lemma 5, if $k = 24mB/\epsilon' = 24l.mB/\epsilon$, then

$$Prob[u_i \text{ incompatible for } \bar{C_i} | u_i \in I_i] \geq 3\epsilon'^2/5 = 3\epsilon^2/5l^2$$

and

$$Prob[u_i \in I_i] \geq (\epsilon/2.l)$$

Hence:

$$Prob[(u_i \in I_i \wedge u_i \text{ incompatible for } \bar{C_i})] \geq (\epsilon/2.l).3\epsilon^2/5l^2 = 3\epsilon^3/10l^3$$

$$Prob[\text{Tester rejects}] \geq \prod_i Prob[(u_i \in I_i \wedge u_i \text{ incompatible for } \bar{C_i})] \geq (3\epsilon^3/10l^3)^l$$

If the decomposition $(I_1, I_2, ...I_l)$ is impossible, there are large incompatible letters. If w is ϵ-far from $\bar{C_1}....\bar{C_l}$, then each large letter is incompatible for $\bar{C_1}$, $\bar{C_2}$ $....\bar{C_l}$. Otherwise the word would be $(2l-1).\epsilon/2.l$ close. The Tester rejects with probability larger than $\epsilon/2.l$.

5 Conclusion

We introduced the *timed edit distance* between timed words and use it to test the membership property of timed automata which admit thick components. We select factors of a timed word proportional to their weight according to the weighted time distribution μ. We followed the property testing framework and constructed a tester which selects finitely many samples of bounded weight to detect if a timed word is accepted or ϵ-far from the language of the timed automaton.

The tester can be used in a streaming context, where the timed letter appear one after the other. A *weighted reservoir sampling* can take the samples online, and we do not need to store the entire word.

This work can be extended in several directions. A first direction would ask if the tester could be generalized to automata with thin components? In this case, there would be a dependency between the time delays of samples in the same thin component. A second direction is the comparison of the languages of two timed automata. Let us say that two timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are ϵ-close if any timed word of $L(\mathcal{A}_1)$ is ϵ-close to $L(\mathcal{A}_2)$ and symmetrically. A natural question is to decide if two timed automata are close or far for finite words. A third direction would be to generalize to infinite words. The distance can be extended to infinite words by taking the limits of the distance on their prefixes. We can then ask for efficient probabilistic algorithms which approximate the equivalence of timed automata for finite and infinite timed words.

Acknowledgment. The authors thank E. Asarin for pointing out the concepts of thin and thick components [7] and his constructive remarks.

References

1. N. Alon, M. Krivelich, I. Newman, and M. Szegedy. Regular languages are testable with a constant number of queries. *SIAM Journal on Computing*, 30(6), 2000.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
3. Rajeev Alur, Robert P. Kurshan, and Mahesh Viswanathan. Membership questions for timed and hybrid automata. In *Proceedings of the 19th IEEE Real-Time Systems Symposium, Madrid, Spain, December 2-4, 1998*, pages 254–263, 1998.
4. Rajeev Alur, Salvatore La Torre, and P. Madhusudan. Perturbed timed automata. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*. Springer Berlin Heidelberg, 2005.
5. Rajeev Alur and P. Madhusudan. *Decision Problems for Timed Automata: A Survey*, pages 1–24. Springer Berlin Heidelberg, 2004.
6. Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of np. *J. ACM*, 45(1):70–122, 1998.
7. Eugene Asarin, Nicolas Basset, and Aldric Degorre. Entropy of regular timed languages. *Inf. Comput.*, 241(C):142–176, April 2015.
8. M. Blum and S. Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, 1995.
9. M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.
10. Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robustness in timed automata. In Parosh Aziz Abdulla and Igor Potapov, editors, *Reachability Problems*. Springer Berlin Heidelberg, 2013.
11. Krishnendu Chatterjee, Rasmus Ibsen-Jensen, and Rupak Majumdar. Edit distance for timed automata. *17th International Conference on Hybrid Systems: Computation and Control*, pages 303–312, 2014.
12. Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1):45–84, Dec 2008.
13. Simon Dobrisek, Janez Zibert, Nikola Pavesic, and France Mihelic. An edit-distance model for the approximate matching of timed strings. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(4):736–741, 2009.
14. Eldar Fischer, Frédéric Magniez, and Michel de Rougemont. Approximate satisfiability and equivalence. *SIAM J. Comput.*, 39(6):2251–2281, 2010.
15. O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.
16. Thomas A. Henzinger and Jean-Francois Raskin. Robust undecidability of timed and hybrid systems. Technical report, University of California at Berkeley, Berkeley, CA, USA, 1999.
17. F. Magniez and M. de Rougemont. Property testing of regular tree languages. *Algorithmica*, 49(2):127–146, 2007.
18. Mehryar Mohri. Edit-distance of weighted automata. In Jean-Marc Champarnaud and Denis Maurel, editors, *Implementation and Application of Automata*. Springer Berlin Heidelberg, 2003.
19. Rohit J. Parikh. On context-free languages. *Journal of the ACM*, 13(4):570–581, 1966.
20. Anuj Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1):87–113, Jan 2000.
21. R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):23–32, 1996.
22. Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, January 1974.