GO IMPLEMENTATION OF UP-TO TECHNIQUES FOR EQUIVALENCE OF WEIGHTED LANGUAGES

Alessandro Cheli
Undergraduate Student
Department of Computer Science
Università di Pisa
Pisa, PI 56127
a.cheli6@studenti.unipi.it

October 1, 2020

ABSTRACT

Weighted automata generalize non-deterministic automata by adding a quantity expressing the weight (or probability) of the execution of each transition. In this work we propose an implementation of two algorithms for computing language equivalence in finite state weighted automata (WAs). The first, a linear partition refinement algorithm, calculates the largest linear weighted bisimulation for any given LWA (Linear Weighted Automaton) through an iterative method, the second algorithm checks the language equivalence of two vectors (states) for a given weighted automata by using an additional data structure representing a congruence relation between states. We then compare results of the two algorithms to verify their correctness and performance on randomly generated samples. We finally provide comparison of runtime statistics and suggest which of the two algorithm is a the best choice for some usage cases.

***K*eywords** First keyword · Second keyword · More

1 Introduction

In [1], up-to techniques are defined for weighted systems over arbitrary semirings, while in [2], up-to techniques are defined for Linear Weighted Automata (LWAs), under a more abstract coalgebraic perspective.

2 Preliminaries and Notation

Note. Given two vector spaces V_1, V_2 we write $V_1 + V_2$ to denote span $(V_1 \cup V_2)$

Definition 2.1. A *weighted automaton* over a field $\mathbb{K}$ and an alphabet A is a triple (X, o, t) such that X is a finite set of states, $t = \{t_a : X \to \mathbb{K}\}_{a \in A}$ is a set of transition functions indexed over the symbols of the alphabet A and $o : X \to \mathbb{K}$ is the output function. A^* is the set of all words over A. ϵ is the empty word and aw is the concatenation of a symbol a to the word $w \in A^*$. A weighted language is a function $\psi : A^* \to \mathbb{K}$. A function mapping each state vector into its accepted language, $[\![\cdot]\!] : \mathbb{K}^X \to \mathbb{K}^{A^*}$ is defined as follows for every weighted automaton:

$$\forall v \in \mathbb{K}^X, a \in A, w \in A^* \qquad [\![v]\!](\epsilon) = o(v) \qquad [\![v]\!](aw) = [\![t_a(v)]\!](w)$$

Two vectors $v_1, v_2 \in \mathbb{K}^{X \times 1}$ are weighted called language equivalent, denoted with $v_1 \sim_l v_2$ if and only if $[\![v_1]\!] = [\![v_2]\!]$. One can extend the notion of language equivalence to states rather than for vectors by assigning to each state $x \in X$ the corresponding unit vector $e_x \in \mathbb{K}^X$. When given an initial state i for a weighted automaton, the language of the automaton can be defined as $[\![i]\!]$.

Definition 2.2. A binary relation $R \subseteq X \times Y$ between two sets X, Y is a subset of the cartesian product of the sets. A relation is called *homogeneous* or an *endorelation* if it is a binary relation over X and itself: $R \subseteq X \times X$. In such case, it is simply called a binary relation over X. An *equivalence relation* is a binary relation that is reflexive, symmetric and transitive.

Definition 2.3. The congruence closure $c(R)$ of a relation R is the smallest congruence relation R' such that $R \subseteq R'$

An equivalence relation which is compatible with all the operations of the algebraic structure on which it is defined on, is called a *congruence relation*. Compatibility with the algebraic structure operations means that algebraic operations applied on equivalent elements will still yield equivalent elements.

We omit the coalgebraic definition for *linear weighted automata* seen in [2] and give a more intuitive definition. In this implementation, we focus only on weighted automata defined over the field of real numbers $\mathbb{R}$.

Definition 2.4. A *linear weighted automaton* (in short, LWA) over the field $\mathbb{K}$ and an alphabet A is a triple $L = (V, o, \{t_a\}_{a \in A})$ where V is a vector space representing the state space, $o : V \to \mathbb{K}$ is a linear map associating to each state its output weight, and $t = \{t_a = V \times V\}_{a \in A}$ is the set of transition functions, represented with liner maps that for each input $a \in A$ associate the next state, in this case a vector in V. As in [3], we have that $\dim(L) = \dim(V)$.

Given a weighted automaton, one can build an isomorphic linear weighted automaton by considering the free vector space generated by the set of states X in the WA, and by linearizing o and t. If X is finite, we can use the same matrices for t and o in both the WA and the corresponding LWA. We are only considering a finite number of states and therefore finite dimensional vector spaces. Let n be the (finite) number of states in an WA. We have that in the corresponding LWA, the transition functions t_a are still represented as $\mathbb{K}^{n \times n}$ matrices. $o \in \mathbb{K}^{1 \times n}$ is represented as a row vector. $t_a(v)$ denotes the vector obtained by multiplying the matrix t_a by the column vector $v \in \mathbb{K}^{n \times 1}$. $o(v)$ denotes the scalar $s \in \mathbb{K}$ obtained by dot product of the row vector o with $v \in \mathbb{K}^{n \times 1}$.

Definition 2.5. The language recognized by a vector $v \in V$ of an LWA (V, o, t) is defined for all words $w \in A*$ as $[\![v]\!]^{\mathcal{L}}_V(w) = o(v_n)$ where v_n is the vector reached from v through the composition of the transition functions corresponding to the words in w.

$$[\![v]\!]^{\mathcal{L}}_V(w) = \begin{cases} o(v) & \text{if } w = \epsilon \\ [\![t_a(v)]\!]^{\mathcal{L}}_V(w') & \text{if } w = aw' \end{cases}$$

We define $\approx_{\mathcal{L}}$ as the behavioral equivalence for a given LWA (V, o, t) as

$$\forall v_1, v_2 \in V,\ v_1 \approx_{\mathcal{L}} v_2 \iff [\![v_1]\!]^{\mathcal{L}}_V = [\![v_2]\!]^{\mathcal{L}}_V \tag{1}$$

Lemma 2.1. *$\approx_{\mathcal{L}}$ coincides with $\sim_l$:*

Let (X, o, t) be a WA and $(\mathbb{K}^X, o^\sharp, t^\sharp)$ the corresponding linear weighted automaton. Then $\forall x \in X,\ [\![x]\!] = [\![x]\!]^{\mathcal{L}}_{\mathbb{K}^X}$

Proof. Proved in section 3.3 of [2] □

Language equivalence can be expressed in terms of linear weighted bisimulations (LWBs for short). Differently from weighted bisimulations, LWBs can be seen both as relations and as subspaces. The subspace representation of LWBs is used in the backwards partition refinement algorithm implemented in [2] and in this work.

3 The Problem

We want to check if two vectors are language equivalent $v_1 \sim_l v_2$.

4 Algorithms

We now recall the backwards algorithm for computing $\approx_{\mathcal{L}}$ defined in [2].

4.1 Backwards Partition Refinement Algorithm for the Largest Weighted Bisimulation

The algorithm is defined by the iterative method:

$$R_0 = \ker(o)^0, \qquad R_{i+1} = R_i + \sum_{a \in A} t^t_a(R_i) \tag{2}$$

Where $\ker(o)^0$ is an annihilator. The algorithm stops when $R_{j+1} = R_j$. An index $j \leq \dim(V)$ is guaranteed to exist, such that the algorithms terminates at step j. It follows that $\approx_{\mathcal{L}} \equiv R_j^0$. Proof is available in section 4.2 of [2]

5 Implementation

The algorithms and data structures for this paper are implemented in the Go programming language. This implementation makes use of the Gonum library for numerical computations. We only import the Gonum libraries for matrices and linear algebra and visual plotting of samples and functions. Real numbers are implemented with double precision floating point numbers, known as the `float64` type in the Go programming language.

Definition 5.1. Applications of SVD

Let's consider the singular value decomposition of a matrix $A \in \mathbb{R}^{m \times n}$:

$$A = U\Sigma V^T \qquad \Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_r) \qquad U \in \mathbb{R}^{m \times m} \qquad V \in \mathbb{R}^{n \times n}$$

Where V and U are orthogonal and the singular values are ordered: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 0$. It follows that $\text{rank}(A)$ is equal to the number of nonzero singular values, and as explained in [4]:

1. $\text{rank}(A) = \text{rank}(\Sigma) = r$
2. The column space of A is spanned by the first r columns of U.
3. The null space of A is spanned by the last nr columns of V.
4. The row space of A is spanned by the first r columns of V.
5. The null space of A^T is spanned by the last mr columns of U.

Of our interest, are only the computation of the null space and columns space. The implementation, applying SVD, can be found in files `lin/colspace.go` and `lin/nullspace.go`.

5.1 Implementing the backwards partition refinement algorithm

To compute $\approx_{\mathcal{L}}$ at the last step of the algorithm, we need to compute R_j^0. If V is a vector space and W is a subspace of W, the annihilator of W, respectively W^0 is a subspace of the space V^* of linear functionals on V. W^0 are the functionals that annihilate on W. Since we are working on subspaces of $\mathbb{R}^n$, we can directly compute the orthogonal complement in our implementation instead of the annihilator.

Proposition 5.1. *If V is a finite dimensional vector space defined with an inner product $\langle \cdot, \cdot \rangle$ and W is a subspace of V then the image of the annhilitaor W^0 through the linear isomorphism $\varphi : V^* \to V$ induced by the inner product, is the orthogonal of W with respect to the said inner product.*

Proof. Let V be an inner product space over the field $\mathbb{K}$ with an inner product defined as $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{K}$. Every linear functional can be represented with a vector. Let $\xi : V \to \mathbb{K}$ be a functional, $\xi \in W^0$. Because $\xi(w) = 0 \quad \forall w \in W$, if v represents ξ we have that $(v, w) = \xi(w) = 0$ for all $w \in W$. We obtain that $\varphi(W^0) \subseteq W^\perp$. If $v \in W^\perp$ then the functional $x \mapsto (v, x)$ cancels over W (by the definition of orthogonality). □

To compute the orthogonal complement of a vector subspace W, we compute $W^\perp = \ker\left(A^T\right)$, where A is the matrix with column vectors in the spanning set of W as its columns. Precisely, W is represented as the column space of A. Proof is available in [5].

References

[1] Filippo Bonchi, Barbara König, and Sebastian Küpper. Up-to techniques for weighted systems (extended version). *CoRR*, abs/1701.05001, 2017.

[2] Filippo Bonchi, Marcello Bonsangue, Michele Boreale, Jan Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Information and Computation*, 211:77 – 105, 2012.

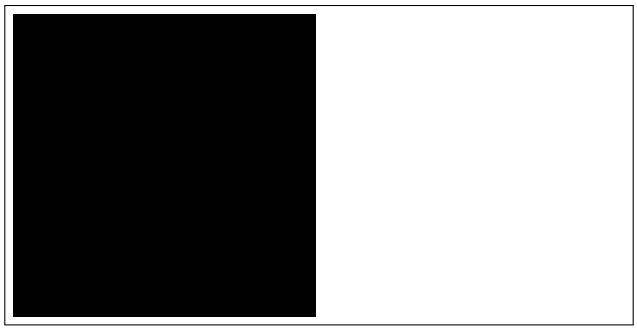

Figure 1: Sample figure caption.

Table 1: Sample table title

	Part	
Name	Description	Size (μm)
Dendrite	Input terminal	$\sim$100
Axon	Output terminal	$\sim$10
Soma	Cell body	up to 10^6

[3] Michele Boreale. Weighted bisimulation in linear algebraic form. In *International Conference on Concurrency Theory*, pages 163–177. Springer, 2009.

[4] Yan-Bin Jia. Singular value decomposition. Available online at `http://web.cs.iastate.edu/~cs577/handouts/svd.pdf`.

[5] Dan Margalit and Joseph Rabinoff. *Interactive Linear Algebra*.