

Minimization of Weighted Automata

Fabian Klopfer

Modelling of Complex Self-Organizing Systems Group, 12.06.2020

Introduction

In the last presentation . . .

- two models for stochastic dynamical systems were considered:
Weighted Automata (WA) and Differential Equations (DE)
- an example for modelling a CRN's dynamics in both models was given:
DE Solving Chemical Master Equation
WA Monte Carlo CTMC

Goals specified

1. Implement minimization algorithm for weighted automata [1]. ✓
2. Implement model reduction algorithm for ODEs [2].
3. Develop reproducible benchmarks
4. Write report

What has been done so far

- Software Requirement Specification & Software Design Document
- Random Basis Minimal WA Construction Algorithm by Kiefer/Schützenberger [1]
- Execution of example by Matlab script and hand
- Implementation of minimization & equivalence algorithm, interfaces, TUI, CLI, tests

The Weighted Automaton Minimization Algorithm [1] I

Weighted Automaton $A = (n, \Sigma, \alpha, \mu, \eta)$, where

- n the number of states
- Σ the input alphabet
- α the initial vector with a non-zero value for all starting states
- μ the set of transition matrices, one per input character
- η the final vector with non-zero values for all ending states

The Weighted Automaton Minimization Algorithm [1] II

Reduction to Schwarz-Zippel Lemma (Polynomial Identity Testing) provide bounds on correctness $\frac{n}{K}$

Author claims complexity is in \mathcal{RNC} , this is not correct (c.f. later)

In the following slides use forward reduction, the backwards reduction is similar

Basic scheme of the algorithm:

Apply forward reduction to WA, then apply backward reduction on the output of the former

The Weighted Automaton Minimization Algorithm [1] III

- Find a basis F of the prefix space using random vectors r_i
 - Add the vectors of all prefix words up to length n together and multiply this vector by n different factors yielding $\{v_1, \dots, v_n\}$
 - Factors are derived by random vectors and structure of prefixes
 - Base is then the maximally linear independent subset of $\{\alpha, v_1, \dots, v_n\}$
- Use basis to do Schützenberger Construction [3]: $\vec{A} = (\vec{n}, \Sigma, \vec{\alpha}, \vec{\mu}, \vec{\eta})$ With
 - $\vec{\mu} = \vec{F}_\mu \vec{F}^{-1}$ or $\vec{F}_\mu = \vec{\mu} \vec{F}$
 - $\vec{\alpha} = e_1$
 - $\vec{\eta} = \vec{F}_\eta$
 - $\vec{n} = \text{rank}(\vec{\mu})$

The Weighted Automaton Minimization Algorithm: Pseudo Code I

Algorithm 1: minimize(WeightedAutomaton WA)

Input: A weighted automata WA

Parameters: K setting the maximal random number

Output: A minimal version of WA

begin

 List<Matrix> randVs;

 WeightedAutomaton minWA;

 randVs \leftarrow generate_random_vectors(WA, K);

 minWA \leftarrow forward_reduction(WA, randVs);

 randVs \leftarrow generate_random_vectors(minWA, K);

 minWA \leftarrow backward_reduction(minWA, randVs);

return minWA;

The Weighted Automaton Minimization Algorithm: Pseudo Code II

Algorithm 2: forward_reduction(WA, randVs)

Input: A weighted automata WA, random vectors randVs

Output: WA transformed by a random minimal forward space base

begin

List<Vector> rhoVectors \leftarrow calculate_rho_vectors(WA, randVs);

Matrix $\vec{F} \leftarrow \text{vstack}(\mathbf{A}.\alpha, \text{rhoVectors}[0], \dots, \text{rhoVectors}[n - 1]);$

int $\vec{n} \leftarrow \text{rank}(\vec{F});$

$\vec{F} \leftarrow \vec{F}[:, \vec{n} - 1, :];$

RowVector $\vec{\alpha} \leftarrow \text{standard_basis}(\vec{n})[0, :];$

Vector $\vec{\eta} \leftarrow \vec{F} \cdot \eta;$

foreach $\mu_i \in \mathbf{WA}.\mu$ **do**

$\vec{\mu}_i \leftarrow \text{Solver}(\vec{F}^T).\text{solve}((\vec{F} \cdot \mu_i)^T)^T;$

return $(\vec{n}, \mathbf{WA}.\Sigma, \vec{\alpha}, \vec{\mu}, \vec{\eta});$

The Weighted Automaton Minimization Algorithm: Pseudo Code III

Algorithm 3: calculate_rho_forward_vectors(WA, randVs)

Input: A weighted automata WA, random vectors randVs

Output: Candidate vectors as base of the Forward space

begin

 List<Word, Vector> words \leftarrow generate_words_forward(WA, WA.n);

for $i = 0$ **to** randVs.length - 1 **do**

for $j = 0$ **to** words.length - 1 **do**

$v_i \mathrel{+}= \text{words}[j].\text{vector} * \text{get_word_factor}(\text{words}[j].\text{word}, \text{randVs}[i]);$

return $\{v_0, \dots, v_{\text{randVs.length}-1}\};$

Algorithm 4: get_word_factor(WA, randVs)

Input: Word w, randVs random vector

Output: WA transformed by a random minimal forward space base

begin

 result $\leftarrow 1;$

for $i = 0$ **to** do

 result $\mathrel{*}= \text{randVs}(\text{word}[i], i);$

Array **randVs** has size A.n and matrices $r_i \in \{1, \dots, K \cdot n\}^{\Sigma \times n}$

The Weighted Automaton Minimization Algorithm: Pseudo Code IV

Algorithm 5: generate_words_forward(WA, randVs)

Input: A weighted automata WA, length of words k

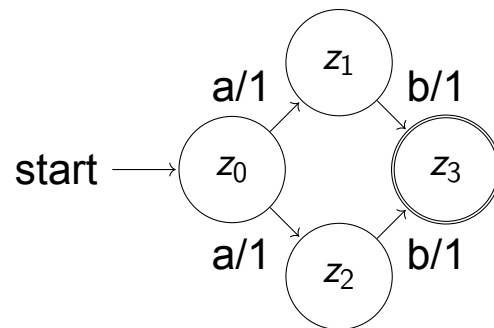
Output: List of Tuples of word and corresponding vector

```
begin
  List<Word, Vector> result;
  if  $k == 1$  then
    result = ;
    foreach  $\mu_i \in WA.\mu$  do
      vect  $\leftarrow$   $WA.\alpha \cdot \mu_i$ ;
      if !vect.isZero() then
        result.add(i, vect);
  else
    result = generate_words_forward(WA,  $k - 1$ );
    for (word, wVector)  $\in$  result do
      if word.length ==  $k - 1$  then
        foreach  $\mu_i \in WA.\mu$  do
          vect  $\leftarrow$  wVector  $\cdot \mu_i$ ;
          if !vect.isZero() then
            newWord  $\leftarrow$  word;
            newWord.append(i);
            result.add(newWord, vect);
  return result;
```

The Weighted Automaton Minimization Algorithm: Example I

Weighted automaton $\mathcal{A} = (n, \Sigma, \mu, \alpha, \eta)$ with

- $n = 3,$
 - $\Sigma = \{a, b\},$
 - $\alpha = (1, 0, 0, 0),$
 - $\eta = (0, 0, 0, 1),$
- $\mu(a) = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$
 - $\mu(b) = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}.$



The Weighted Automaton Minimization Algorithm: Example II

$$r^{(1)} = \begin{pmatrix} 9 & 5 & 5 & 7 \\ 6 & 11 & 2 & 1 \end{pmatrix}; \quad r^{(2)} = \begin{pmatrix} 2 & 3 & 1 & 2 \\ 12 & 3 & 9 & 4 \end{pmatrix} \quad r^{(3)} = \begin{pmatrix} 2 & 7 & 9 & 10 \\ 1 & 11 & 2 & 6 \end{pmatrix} \quad r^{(4)} = \begin{pmatrix} 4 & 5 & 2 & 10 \\ 5 & 9 & 5 & 5 \end{pmatrix}$$

$$\alpha\mu(a)r_i + \alpha\mu(a)\mu(b)r_i = (0, 1, 1, 0)r_i + (0, 0, 0, 2)r_i$$

$$v_1 = 9 \cdot (0, 1, 1, 0) + 9 \cdot 11 \cdot (0, 0, 0, 2) = (0, 9, 9, 198)$$

$$v_2 = 2 \cdot (0, 1, 1, 0) + 2 \cdot 3 \cdot (0, 0, 0, 2) = (0, 2, 2, 12)$$

$$v_3 = 2 \cdot (0, 1, 1, 0) + 2 \cdot 11 \cdot (0, 0, 0, 2) = (0, 2, 2, 44)$$

$$v_4 = 4 \cdot (0, 1, 1, 0) + 4 \cdot 9 \cdot (0, 0, 0, 2) = (0, 4, 4, 72)$$

$$\vec{F} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 9 & 9 & 198 \\ 0 & 2 & 2 & 12 \end{pmatrix}$$

The Weighted Automaton Minimization Algorithm: Example III

$$\vec{F}_{\mu(\sigma)} = \vec{\mu}(\sigma) \vec{F} \equiv \vec{\mu}(\sigma) = \vec{F}_{\mu(\sigma)} \vec{F}_R^{-1}$$

$$\vec{\mu}(a) = \begin{pmatrix} 0 & \frac{-1}{24} & \frac{11}{16} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\vec{\mu}(b) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{8} & \frac{-9}{16} \\ 0 & \frac{1}{36} & \frac{-1}{8} \end{pmatrix}$$

$$\vec{\eta} = \vec{F}_{\eta} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 9 & 9 & 198 \\ 0 & 2 & 2 & 12 \end{pmatrix} \cdot (0, 0, 0, 1)^T = (0, 198, 12)^T$$

$$\Rightarrow \vec{\mathcal{A}} = (3, \{a, b\}, \vec{\mu}, (1, 0, 0), (0, 198, 12)^T)$$

Weighted Automata Equivalence Algorithm [1]

Zeroneess Problem: $\forall w \in \Sigma^* : A(w) = 0$

Automaton A with n states

If $A \not\equiv 0$

$\Rightarrow \exists w \in \Sigma^{n-1} : A(w) \neq 0$

[4]

Multivariate polynomial $P(x)$ of deg n
Plugging in random $r_i \in \{1, \dots, Kn\}$ for x_i :
 $Pr[P(r) = 0 \mid P(x) \neq 0] = \frac{n}{Kn} = \frac{1}{K}$

[5]–[8]

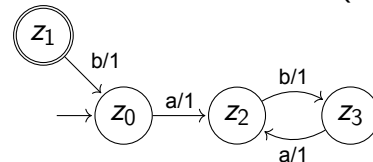
Monomial with words as vars:

$$A(w)x_w$$

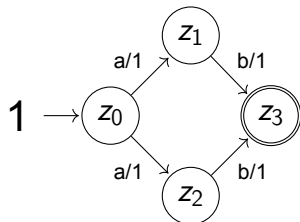
Substitute random r_w for x_w

$$P(r) = \sum_{k=0}^{n-1} \sum_{w \in \Sigma^k} A(w) \cdot r_w$$

$$\forall w \in \Sigma^* : A(w) = 0$$



Equivalence Problem: $\forall w \in \Sigma^* : A(w) = B(w)$



How to use zeroness to check equivalence?

$$\forall w \in \Sigma^* : A(w) - B(w) = 0$$

$$\exists w \in \Sigma^{n-1} :$$

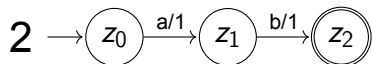
$$C(w) \neq 0$$

$$\Rightarrow A \neq B$$

$$\forall w \in \Sigma^{n-1} :$$

$$C(w) = 0$$

$$\Rightarrow A = B$$



Subtraction Automaton C

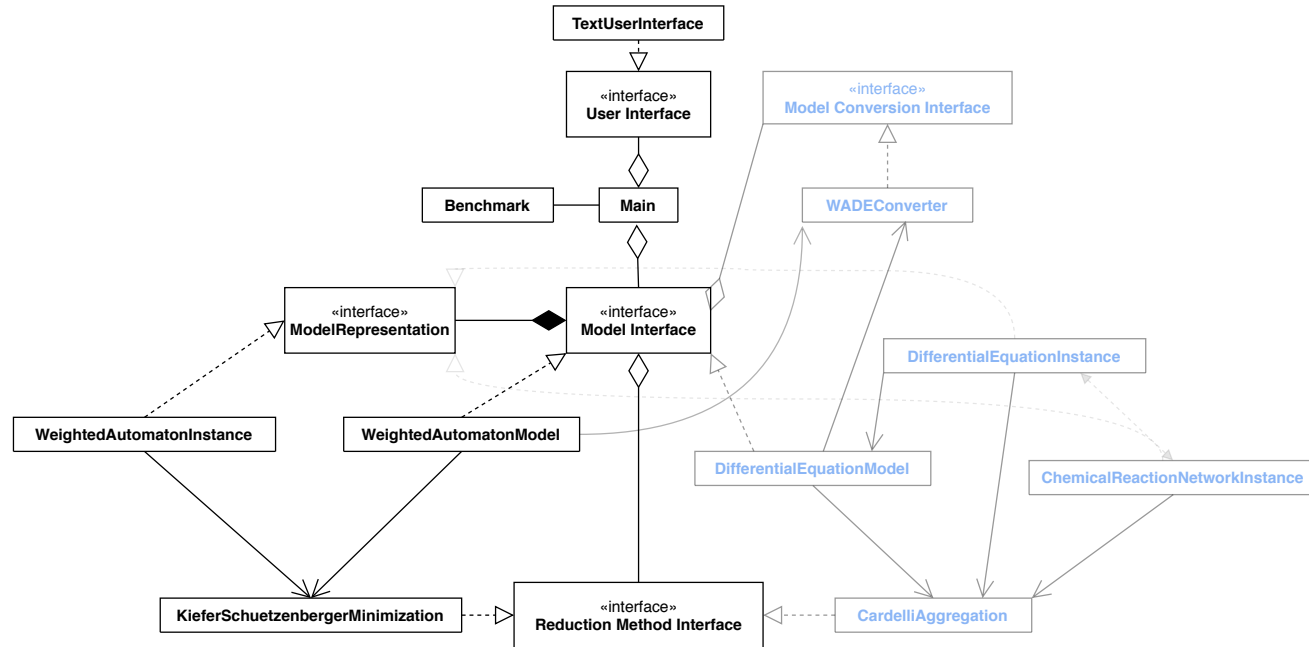
$$n_C = n_A + n_B, \Sigma_C = \Sigma_A \cup \Sigma_B$$

$$\alpha_C = (\alpha_A m - \alpha_B)$$

$$\eta_C = (\eta_A, \eta_B)^T$$

$$\mu_C = \begin{pmatrix} \mu_A & 0 \\ 0 & \mu_B \end{pmatrix}$$

Implementation Details I



Implementation Details II

Sparse vs. Dense Matrices

Templates, Concepts and Eigen Wrapper

OpenMP multi-threading pragmas

further optimization possibilities

Code & Demo









Up Next

Previously greyed out part: Ordinary Differential Equations & Cardelli et al. [2]

What's the connection now? **extremely sloppy notation ahead:**

LTS \leftrightarrow WA \leftrightarrow PA \leftrightarrow MC \leftrightarrow Rule-based Modeling \leftrightarrow ODE \leftrightarrow DE

Bibliography

-  S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell, “On the complexity of equivalence and minimisation for q-weighted automata”, *Logical Methods in Computer Science*, vol. 9, 2013.
-  L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin, “Maximal aggregation of polynomial dynamical systems”, *Proceedings of the National Academy of Sciences of the United States of America*, vol. 114 38, pp. 10 029–10 034, 2017.
-  M. P. Schützenberger, “On the definition of a family of automata”, *Information and control*, vol. 4, no. 2-3, pp. 245–270, 1961.
-  W.-G. Tzeng, “A polynomial-time algorithm for the equivalence of probabilistic automata”, *SIAM Journal on Computing*, vol. 21, no. 2, pp. 216–227, 1992.
-  R. Zippel, “Probabilistic algorithms for sparse polynomials”, in *International symposium on symbolic and algebraic manipulation*, Springer, 1979, pp. 216–226.
-  J. T. Schwartz, “Fast probabilistic algorithms for verification of polynomial identities”, *Journal of the ACM (JACM)*, vol. 27, no. 4, pp. 701–717, 1980.
-  R. A. Demillo and R. J. Lipton, “A probabilistic remark on algebraic program testing”, *Information Processing Letters*, vol. 7, no. 4, pp. 193–195, 1978, ISSN: 0020-0190. DOI: [https://doi.org/10.1016/0020-0190\(78\)90067-4](https://doi.org/10.1016/0020-0190(78)90067-4). [Online]. Available: <http://www.sciencedirect.com/science/article/pii/0020019078900674>.
-  Ø. Ore, “Über höhere kongruenzen”, *Norsk Mat. Forenings Skrifter*, vol. 1, no. 7, p. 15, 1922.