Grammar Inference:
NFA learning with constraints

T. Jastrzab[1] F. Lardeux[2] E. Monfroy[2]

[1]Silesian Univ. of Technology, Gliwice, Poland.
[2]LERIA, Université d'Angers, France

Outline

1. Grammar inference
2. A basic SAT model
3. Improved models
4. Hybrid models
5. Experimental Results
6. Conclusion

Grammar inference

- Process of learning a grammar from observations
- Grammar: production rules or automaton
- Observations: sample of words
  - positive words: words of the language
  - negative words: not element of the language
- Applications: compiler design, bioinformatics, speech recognition, pattern recognition, machine learning, …

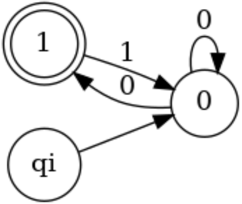
Our problem

- Learning Non-determinist Finite Automaton (NFA) with k states
- Data: $S = S^+ \cup S^-$:
  - a sample S^+ of positive words accepted by the NFA
  - a sample S^- of negative words rejected by the NFA

A simple example

+ 0 1 0
+ 0
+ 0 0
- 0 1
- 1
-

+ k=2 $\Rightarrow$



Our tools

- Constraint-based solvers: interleaving of
  - search space reduction (constraint propagation)
  - decision (enumeration)
- Various possible domains:
  - FD: finite domain integers with linear arithmetic and global constraints
  - Zero-One variables: NLP / quadratic constraints, $\sum$, max
  - SAT: Boolean variables and Boolean formula (in CNF)
- Goal: improving NFA learning with classic SAT solvers

Some notations: NFA

Let $A = (Q, \Sigma, q, F)$ be a NFA with:

- $Q = \{q_1, \dots, q_k\}$ a set of states,
- Σ a finite alphabet (a set of n symbols),
- q the initial state,
- and F the set of final states.

Moreover, λ is the empty word, and $K = \{1, \dots, k\}$

SAT model: Boolean variables

- k: size of the NFA to be learned
- $F = \{f_1, \ldots, f_k\}$: final states
  f_i is true iff State q_i is final
- $\Delta = \{\delta_{s,\overrightarrow{q_i q_j}} | s \in \Sigma,\ i, j \in K\}$: $n.k^2$ transitions
  $\delta_{s,\overrightarrow{q_i q_j}}$ is true iff there is a transition from State q_i to q_j with symbol s

Direct SAT model: Constraints

- empty word λ:

$$(\lambda \in S^+ \longrightarrow f_1) \wedge (\lambda \in S^- \longrightarrow \neg f_1) \qquad (1)$$

- For $w \in S^+$: at least a path from q_1 to a final q_j

$$\bigvee_{j \in K} \bigvee_{d \in D_{w,\overrightarrow{q_1 q_j}}} (d \wedge f_j) \qquad (2)$$

$D_{w,\overrightarrow{q_1 q_j}}$: set of paths for w from q_1 to q_j

- For $w \in S^-$, and q_j: there is no path, or q_j is not final:

$$\neg \left[\bigvee_{j \in K} \bigvee_{d \in D_{w,\overrightarrow{q_1 q_j}}} (d \wedge f_j) \right] \qquad (3)$$

Direct model: spatial complexity

- converted in CNF with Tseitin transformations
- number of variables:

$$\mathcal{O}(|S^+| \cdot k\cdot^{|\omega_+|})$$

- number of clauses

$$\mathcal{O}(|S^+| \cdot (|\omega_+| + 1) \cdot k^{|\omega_+|})$$

- with ω_+ the length of the longest word of S^+

Prefix model

- Idea:
  - shared prefixes
  - one Boolean variable per prefix

- Realization:
  - $Pref(W) = \cup_{w \in W} Pref(w)$
  - For each $w \in Pref(S)$,
    a Boolean variable $p_{w,\overrightarrow{q_1 q_i}}$ to determine the existence of a path for w from state q_1 to q_i.

Prefix model: $P_k = (1) \wedge (4) \wedge (5) \wedge (6) \wedge (7)$

- For each prefix $w = a \in \Sigma$, there is a path of size 1 for w:

$$\bigvee_{i \in K} \delta_{a,\overrightarrow{q_1 q_i}} \leftrightarrow p_{a,\overrightarrow{q_1 q_i}} \tag{4}$$

- For each prefix $w = va$, $w, v \in Pref(S)$, and $a \in \Sigma$:

$$\bigwedge_{i \in K} (p_{w,\overrightarrow{q_1 q_i}} \leftrightarrow (\bigvee_{j \in K} p_{v,\overrightarrow{q_1 q_j}} \wedge \delta_{a,\overrightarrow{q_j q_i}})) \tag{5}$$

- For each word $w \in S^+ \setminus \{\lambda\}$:

$$\bigvee_{i \in K} p_{w,\overrightarrow{q_1 q_i}} \wedge f_i \tag{6}$$

- For each word $w \in S^- \setminus \{\lambda\}$:

$$\bigwedge_{i \in K} (\neg p_{w,\overrightarrow{q_1 q_i}} \vee \neg f_i) \tag{7}$$

Suffix model

- Idea:
  - shared suffixes
  - one Boolean variable per suffix

- Realization:
  - $Suf(W) = \cup_{w \in W} Suf(w)$
  - For each $w \in Suf(S)$,
    a Boolean variable $p_{w,\overrightarrow{q_i q_j}}$ to determine the existence of a path for w from state q_i to q_j.

université angers

Suffix model: $S_k = (1) \wedge (8) \wedge (9) \wedge (6) \wedge (7)$

- For each suffix $w = a \in \Sigma$, there is a path of size 1 for w:

$$\bigvee_{(i,j)\in K^2} \delta_{a,\overrightarrow{q_iq_j}} \leftrightarrow p_{a,\overrightarrow{q_iq_j}} \tag{8}$$

- For each suffix $w = av$, $v \in Suf(S)$ and $a \in \Sigma$:

$$\bigwedge_{(i,j)\in K^2} (p_{w,\overrightarrow{q_iq_j}} \leftrightarrow (\bigvee_{k\in K} \delta_{a,\overrightarrow{q_iq_k}} \wedge p_{v,\overrightarrow{q_kq_j}})) \tag{9}$$

Complexity

- Prefix model (in CNF):
  - number of variables: $\mathcal{O}(\sigma \cdot k^2)$
  - number of clauses $\mathcal{O}(\sigma \cdot k^2)$
    with $\sigma = \Sigma_{w \in S}|w|$
- Suffix model (in CNF):
  - number of variables: $\mathcal{O}(\sigma \cdot k^3)$
  - number of clauses $\mathcal{O}(\sigma \cdot k^3)$

Hybrid models

- Idea:
  - splitting each word into 2 words: $w_i = p_i.s_i$
  - S_p set of p_i
  - S_s set of s_i

- Challenge: where to split each word?

Hybrid models: $H_k = (1) \wedge (4) \wedge (5) \wedge (8) \wedge (9) \wedge (10) \wedge (11)$

Constraints:

- for each prefix of $Pref(S_p)$: generate Constraints (4, 5)
- for each suffix of $Suf(S_s)$: generate Constraints (8,9)
- for each $w = p.s$, link p to s:
  - if $w = p.s \in S^-$:

$$\bigwedge_{(j,k) \in K^2} (\neg p_{p,\overrightarrow{q_1 q_j}} \vee \neg p_{s,\overrightarrow{q_j q_k}} \vee \neg f_k) \tag{10}$$

  - if $w = p.s \in S^+$:

$$\bigvee_{(j,k) \in K^2} p_{p,\overrightarrow{q_1 q_j}} \wedge p_{s,\overrightarrow{q_j q_k}} \wedge f_k \tag{11}$$

Iterated Local Search Hybrid Models

- ILS: optimizes where to split each word
- Search space: all possible splits for each word
- Fitness: $f(S_p, S_s) = |Pref(S_p)| + k \cdot |Suf(S_s)|$
- Init: random split for each word
- at each iteration:
  - select a w randomly (roulette wheel selection)
  - find the best split for w (w.r.t. fitness)
- Note:
  - no need for random walk, restart: diversification selecting w
  - possible improvements changing Init

Genetic Algorithm Hybrid Models

- GA: optimizes where to split each word
- Search space: all possible splits for each word
- Fitness: $f(S_p, S_s) = |Pref(S_p)| + k \cdot |Suf(S_s)|$
- Population (constant size): an individual = a split for each word
- uniform crossover: children inherit randomly prefix and suffix of their parents
- mutation: new split for words of an individual (probability of mutation)
- stop: a number of populations or no improvement
- Init: random split for each word
- Possible improvements changing Init population

Hybrid models: best suffix model

Idea:

- bad complexity of S model
  $\Rightarrow$ optimize construction of suffixes

- order on suffixes: $\succcurlyeq$
  based on length of suffix $\times$ number of words with this suffix

$$s_1 \succcurlyeq s_2 \quad \Leftrightarrow \quad |s_1| * |\Omega(s_1)| \geq |s_2| * |\Omega(s_2)|$$

  with $\Omega(s)$: words of S admitting s as a suffix

Hybrid models: best suffix model

Realization:

- $\mathcal{S}$ the set of best suffixes
  - cover: contains a suffix for each word of S
  - maximize: contains the most important suffixes w.r.t. $\succcurlyeq$
  - minimal: cover of S is lost by removing a suffix
- $\mathcal{S}$ = S_s of the $S_k^\star$ hybrid model
- S_p = set of prefixes completing the best suffix of each word

Hybrid models: best prefix model

- $P_k^\star$ is built similarly as the best suffix model

Hybrid models: mixing hybrid models

Remark:

- $S_k^\star$ and $P_k^\star$ can be used as:
  - Init for the ILS model (named $ILS_k(S_k^\star, f)$)
  - Initial population for the GA model

Experimentation

- Python + PySAT
- cluster with Intel-E5-2695 CPUs, 10 GB of memory
- timeout: 10 min
- SAT solver: Glucose
- Benchmark: from FI'21
    - based on StaMinA competition
    - alphabets of size 2, 5, and 10
    - $|S^+| = |S^-|$ and varies from 10 to 100 words

Results

- $S_k^\star$ and $ILS_k(S_k^\star, f)$ generate smaller instances
  Reason: optimization of suffixes
- $ILS_k(rand, f)$ solves slightly more instances
- $S_k^\star$ model generates the fastest SAT instances to solve
- generation + solving time:
  1. $S_k^\star$
  2. $ILS_k(S_k^\star, f)$
     (penalized by slower generation)
- symmetry breaking:
  few extra clauses, good solving gain

Property: from $k+1$ NFA to k NFA

- property: NFA of size $k \Longrightarrow$ NFA of size $k+1$
  - with 1 final state (the new state)
  - with no outgoing transition from the final state
  - with redundancy of some transitions (the ones leading to previous final states)
- $\Longrightarrow$ suffix model for $k+1$ in $\mathcal{O}(\sigma \cdot (k+1)^2)$ instead of $\mathcal{O}(\sigma \cdot (k)^3)$
- and same reduction for the hybrids

Using the property

- with a reduction algorithm
  - add a final state to the model
  - a reduction algorithm to transform the $k+1$ NFA into a k NFA

  $\Longrightarrow$ the algorithm may fails

- integrate all the "building" constraints into the model
  - complete the model
  - remove $k+1$ state and its incoming transitions
  - make final state

  $\Longrightarrow$ always works

  $\Longrightarrow$ better results than with k states!!!

Conclusion

- New models $\Rightarrow$ Improved NFA learning
  - faster
  - larger instances (k, $|\Sigma|$, $|S|$)
- Good results compared to previous works
- Future works:
  - distributed computation / parallelization
  - over-constrained models