Regular Language Inference for Learning Rules of Simplified Boardgames

Jakub Kowalski, Andrzej Kisielewicz

University of Wrocław, Poland

CIG

15.08.2018

Simplified Boardgames

Simplified Boardgames (Y. Björnsson; 2012)

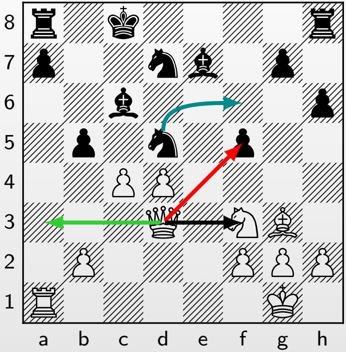
- Turn based; two players; zero-sum games;
- Rectangular board; fixed initial position; max one piece per square;
- One piece movement per turn;
- *Capturing* only at destination square;
- Winning conditions:
  - reaching a *goal* square using a certain piece,
  - captured some number of opponent's pieces;
- *Draw* occurs when the preset maximum game length is reached.

Set of piece's move rules

Regular language over an alphabet Σ containing triplets $(\Delta x, \Delta y, on)$:

- Δx, Δy are relative column/row distances;
- $on \in \{e, p, w\}$ describes the content of the destination square:
  - e – **e**mpty square,
  - p – square occupied by an o**p**ponent piece,
  - w – square occupied by an o**w**n piece.

Move examples



[1]

- ♕d3-a3: $(-1, 0, e)(-1, 0, e)(-1, 0, e)$
- ♕d3-f5: $(1, 1, e)(1, 1, p)$
- ♞d5-f6: $(2, 1, e)$
- ♕d3-f3: ~~$(1, 0, e)(1, 0, w)$~~

[1] Deep Blue vs Garry Kasparov, 1997, Game 6, Move 19 (last)

Simplified Chess example

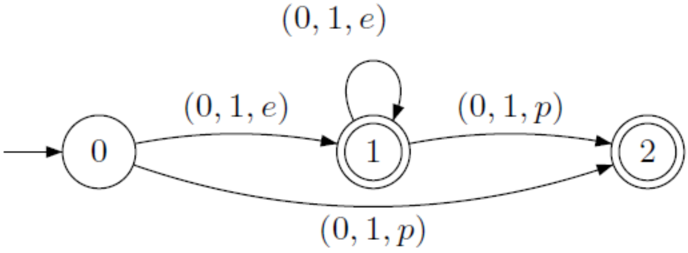
```
<--BOARD-->
8 8
rnbqkbnr
pppppppp
........
........
........
........
PPPPPPPP
RNBQKBNR
```

```
<--GOALS-->
200 &
@P 0 7, 1 7, 2 7, 3 7, 4 7, 5 7, 6 7, 7 7 &
@p 0 0, 1 0, 2 0, 3 0, 4 0, 5 0, 6 0, 7 0 &
#K 1 &
#k 1 &
```

```
<--PIECES-->
P (0,1,e) + (-1,1,p) + (1,1,p) +
  (0,1,e)(0,5,e)(0,-4,e) + ... &
N (2,1,e) + (2,-1,e) + (-2,1,e) + (-2,-1,e) + (1,2,e)... &
Q (0,1,e)^* + (0,1,e)^*(0,1,p) + (1,1,e)^* + ... &
```

Deterministic Finite Automata

- Let Σ^* be the set of all possible words over the alphabet Σ.
- For DFA $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, where Q is the set of states, q_0 is the initial state, $F \subseteq Q$ is the set of accepting states, and $\delta : Q \times \Sigma \to Q$ the transition function (which, in our study, is usually a partial function),
- by $L(\mathcal{A})$ we denote the language accepted by $\mathcal{A}$.



Learning by Observing

Learning Rules of Simplified Boardgames by Observing, Y. Björnsson; 2012

Goal

Learn game rules given records of previously played matches.
In Simplified Boardgames it turns to learn each piece's DFA.

Proposed scenarios

- *single move known*
  the observer sees only the performed move (real-world)
- *all legal moves*
  for every position all legal moves are listed (GGP-like)

Motivation

- Produce fast alternative to move-generation mechanism in GGP domain

Proposed algorithm

Algorithm LearnDFA(Piecetype *pt*, TrainingData *td*)

```
dfa ← constructPTA(pt, td)
if not consistent(pt, td) then
    return null
end if
dfa_min ← minimizeDFA(dfa)
n ← 0
Q.insert(dfa_min)
while not Q.empty() and n < MaxExpansions do
    dfa ← Q.pop()
    if |dfa| < |dfa_min| then
        dfa_min ← dfa
    end if
    statepairs ← generalizeCandidates(dfa, K)
    for all (s, s′) ∈ statepairs do
        dfa′ ← NFAtoDFA(collapse(dfa, s, s′))
        if consistent(pt, dfa′, td) then
            dfa′ ← minimizeDFA(dfa′)
            Q.insert(dfa′)
        end if
    end for
    n ← n + 1
end while
return dfa_min
```

Consistency checking

Consistent DFA should generate all moves known to be legal and no moves known to be illegal.

Algorithm consistent(Piecetype pt, DFA dfa, TrainingData td)

```
for all {pos ∈ td} do
    for all {sq ∈ pos.board | pieceType(sq) = pt} do
        movesDFA ← generateMoves(dfa, pos, sq)
        if pos.moves(sq) ⊈ movesDFA then
            return False
        end if
        if pos.movelisting = all then
            if movesDFA ⊃ pos.moves(sq) then return False end if
        else { pos.movelisting = some }
            for all {pmp ∈ movesDFA \ U} do
                if 𝔉(pmp) ⊈ U then
                    return False
                end if
            end for
        end if
    end for
end for
return True
```

Regular Language Inference

Regular Language Inference

Definition

The problem of finite automata identification using labeled samples: given a disjoint sets of words S_+ containing words belonging to the target language L, and S_- containing words that do not belong to L, we ask what is the size of the minimal DFA consistent with these sets.

Algorithms

- Proven to be NP-Hard (Gold, 1978)
- TB/Gold (Trakhtenbrot and Barzdin, 1973 / Gold, 1978)
- RPNI – Regular Positive and Negative Inference (Oncina and Garcia, 1992 / Lang, 1992)
- RPNI guarantees that the obtained DFA is consistent, and is equivalent to the target DFA if some special conditions are met for S_+, S_-.

RPNI

Algorithm $\mathrm{RPNI}(S_+ \subseteq \Sigma^*, S_- \subseteq \Sigma^*)$

$\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle \leftarrow PTA(S_+)$
$blue \leftarrow \{\delta(q_0, a) : a \in \Sigma\}$
$red \leftarrow \{q_0\}$
while $blue \neq \emptyset$ **do**
    choose $q \in blue$
    if $\exists p \in red.\ L(MergeAndFold(\mathcal{A}, p, q)) \cap S_- = \emptyset$ **then**
        $\mathcal{A} \leftarrow MergeAndFold(\mathcal{A}, p, q)$
    else
        $red \leftarrow red \cup \{q\}$
    end if
    $blue \leftarrow \{\delta(q, a) : q \in red, a \in \Sigma\} \setminus red$
end while
return $\mathcal{A}$

Boardgame Rules Inference

Problem Statement and Model

Moves partitioning

Given data obtained by observing movements of a piece p, there exist the partition of Σ^* (alphabet of all possible moves) into the following languages:

- the language S_+ of observed legal moves,
- the language S_- of known illegal moves,
- the language S_0 containing all words that are impossible to perform from any square on the board,
- and $S_?$ containing all the other words.

So, if $w \in S_?$, then it is theoretically possible that w belongs to L_p (the language of legal movements of p), yet there is no evidence in the data that it is legal or not.

- In the *single move known* scenario the S_- set is always empty, and $S_?$ may be non-empty.
- In the *all legal moves known* scenario S_- is not empty, yet we can still have words in $S_?$.

Example

Consider moves of the white knight: $(-1, 2, e) \in S_+$ because it is listed as one of the legal moves; $(-1, 2, e)(-1, 2, p) \in S_-$ because this move can be performed on current position (capturing black knight) but is not listed as a legal one; $(-1, 2, e)^2 \in S_?$ because in given position we cannot decide its correctness; $(-1, 2, e)^4 \in S_0$ because move length exceeds the board size.

```
r.k....r
p..nb.p.
..b....p
.p.n.p..
..PP....
...Q.NB.
.P...PPP
R.....K.
43
P 1 1 (0,1,e)
P 1 1 (0,1,e)(0,5,e)(0,-4,e)
N 5 2 (2,1,e)
N 5 2 (-2,-1,e)
N 5 2 (1,2,e)
N 5 2 (-1,2,e)
N 5 2 (-1,-2,e)
Q 3 2 (1,1,e)
Q 3 2 (1,1,e)(1,1,p)
      ...
```

Partitioning languages properties

- Languages S_+, S_- and $S_?$ are finite and closed on taking substrings, i.e. for any $w \in S_+ \cup S_- \cup S_?$ every substring of w also belongs to $S_+ \cup S_- \cup S_?$.
- Language S_0 is infinite, and such that for all $w \in S_0$ and $a \in \Sigma$, we have $aw \in S_0 \wedge wa \in S_0$.
- There exist a procedure $\mathcal{O} : \Sigma^* \to \{T, F\}$ which, for given w, decide in time $\Theta(|w|)$ if $w \in S_0$. (We can iterate through the word summing relative distances and checking if the board size was exceeded.)

Observation

Given board of size $n \times n$, and $S = S_+ \cup S_- \cup S_?$. We have that:

$$2 \sum_{k=1}^{n^2-1} 3^k \frac{(n^2-1)!}{(n^2-1-k)!} \leq |S| \leq n^2 \sum_{k=1}^{n^2-1} 3^k \frac{(n^2-1)!}{(n^2-1-k)!},$$

which estimates the total number of moves that can be made on such a board.

Consistency

For a given piece p, let $\mathcal{A}$ be a DFA approximating L_p based on given observations.

- It is required that $S_+ \subseteq L(\mathcal{A})$ and $L(\mathcal{A}) \cap S_- = \emptyset$.
- Whether $L(\mathcal{A}) \cap S_0$ will be empty or not, in practice do not influence the results generated by $\mathcal{A}$.
- $L(\mathcal{A})$ could contain some words from $S_?$
  - allows to simplify the language definition and minimize produced DFA,
  - unsafe if e.g., making illegal move means instant loss

Björnsson's Algorithm consistency analysis

Consider the *limited rider* piece, i.e., a piece that moves in one direction for a given limited distance (e.g., Short Rook, Cloud Eagle).

Theorem

Let $\mathcal{D}$ be a training data in the all legal moves scenario, such that for a limited rider $\mathcal{Q}$ all its legal moves were observed, but any extension of its movements (i.e. one-step longer rides) are in $S_?$.
If $\mathcal{Q}$ is a limited rider in a game $\mathcal{G}$, and a training data $\mathcal{D}$, then the DFA returned by the Björnsson's Algorithm generates illegal moves.

Theorem

Consider single move known scenario and a game $\mathcal{G}$ containing a limited rider $\mathcal{Q}$ and another piece $\mathcal{R}$ with unlimited ride, i.e. satisfying $\{a_i^j, a_i^{j-1}b_i\} \subseteq L_{\mathcal{R}} \cup S_0$ for all $j > 0$. Let training data $\mathcal{D}$ be such that all the moves of the form a_i^j and $a_i^{j-1}b_i$ are observed for the piece $\mathcal{R}$. Then, the DFA returned by the Björnsson's Algorithm generates illegal moves.

Existing algorithms complexity analysis

Observation

Let S_+ be the language accepted by the piece's prefix tree acceptor pt, and $\mathcal{C}(k, td)$ the complexity of consistency checking the given training data td and a DFA with k states. Then, assuming that the MaxExpansions and K parameters are constant, the complexity of $LearnDFA$(pt,td) may be bounded by

$$O(||S_+||(2^{||S_+||} + \mathcal{C}(||S_+||, td))).$$

Observation

Let S_+ be the language accepted by the piece's prefix tree acceptor pt, and $\mathcal{C}(k, td)$ the complexity of consistency check given training data td and DFA with k states. Then, the complexity of $RPNI$ algorithm is

$$O((||S_+|| + \mathcal{C}(||S_+||, td))||S_+||^2).$$

Efficient Consistency Checking

Fast consistency check

Assumption

$$S_+ \subseteq L \;\text{ and }\; L \cap (S_- \cup S_?) = \emptyset, \tag{1}$$

Algorithm FastCheck($\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F\rangle$, $x \in Q$, $T = \langle Q', \Sigma, \delta', q'_0, F'\rangle$, $x' \in Q'$, $w \in \Sigma^*$)

for all $a \in \Sigma$ **do**
    if $\exists y, y'.\ \delta(x,a) = y \wedge \delta'(x',a) = y'$ **then**
        if $F(y) \neq F'(y')$ **then return** *False* **end if**
        if $\neg$FastCheck($\mathcal{A}$, y, T, y', wa) **then return** *False* **end if**
    end if
    if $\mathcal{O}(wa)$ **then continue end if**
    if $F(\delta(x,a))$ **then return** *False* **end if**
    if $\neg$FastCheck($\mathcal{A}$, $\delta(x,a)$, T, *null*, wa) **then return** *False* **end if**
end for
return *True*

Complexity

Linear in $||L(\mathcal{A}) \cap (S_+ \cup S_?)||$.

Fractional Acceptance Consistency Check

Assumption

If a DFA representing a language of piece movements is small and does not contradict given data, then with a high probability it is correct.

FractionalCheck$_\alpha$

If it produces language L, then at most $(1-\alpha)|L \setminus S_0|$ generated words belong to $S_?$.

Description

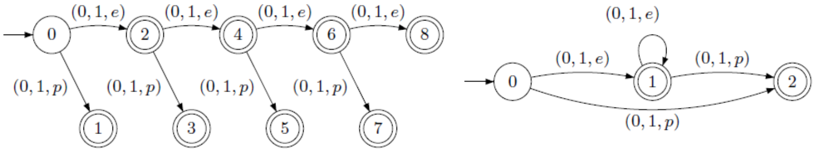
- Instead of returning *False* in *FastCheck* in case of finding an accepted move from $S_?$, the algorithm tracks the number of such moves.
- Function returns *False* if their number exceeds the threshold.
- In the *all legal moves* scenario, browses through all the observed positions and checks subsets between observed and DFA-generated moves.

Spine Compaction Algorithm

Idea

- RPNI and Björnsson's Algorithm are general purpose methods.
- Let's do something more domain-specific.

Typical case



Goal

- Fast learning of probable piece movements
- performing only specialized state merges
- minimize the number of required consistency checks
- Successfully deal with not-so-typical cases

Spines selection

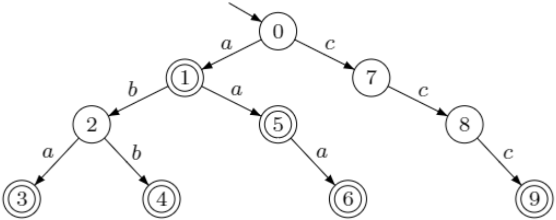
- PTA's usually have a form of multiple *spines* with similar subtrees attached.
- We will search for a *vertebra* words, i.e., the words that can be used as a period of some spines in PTA.
- Vertebras are proper if the corresponding states it points out have the same acceptance.
- States along the spine (distant of vertebra length) will be the promising candidates for a merge.

Main procedure

1. Find all spines in given PTA (iterating for possible periods k)
2. Analyze spines in proper order and candidates for merge (pairs if states)

Spines selection

Example



For the given PTA, the spine selection procedure returns spines as $(vertebra, initial\ state, length)$ triples. Result for $k = 1$: $(a, 1, 2)$, $(c, 0, 2)$. Result for $k = 2$: $(ab, 0, 3)$, $(bb, 1, 2)$, $(aa, 1, 2)$, $(cc, 0, 2)$.

Time complexity

Assuming that we traverse only trees, the time complexity of spine selection is $\Theta(|Q|)$.

Spine compaction

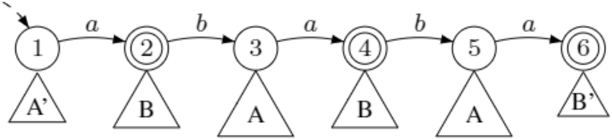
- If we have two corresponding states in a spine, we can safely merge them iff. subtrees rooted in them are the same (except vertebra outgoing edge).
- This is too costly to check, so instead we only require first level equality.
- Moreover, we allow subtrees further from the root to be smaller (assuming sparse data).

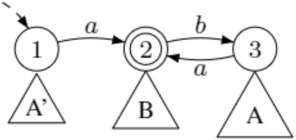
Sketch

- *Pair selection* procedure starts with a spine root r as a candidate for merging with $\delta(r, w)$.
- We traverse through the spine comparing corresponding subtrees and replace candidate for merging if we meet a larger corresponding subtree.
- The complexity of the procedure is $\Theta(|v| \cdot d)$, assuming d is the maximal degree of a node in a spine.

Spine compaction

Example





Example of the spine compaction process. On the top ab-spine rooted in 1, with subtrees such that $A' \subset A$ and $B' \subset B$. On the bottom the situation after compaction: pair $(1, 3)$ is not a safe candidate for merging, so the states 2 and 4 were selected and merged instead.

Main algorithm

- We use constant indicating maximal allowed length of cycle
- We search for all spines in the initial PTA (excluding the root)
- Pairs selection procedure is filtering spines to find the proper candidates to merge
- We try to merge every pair and check consistency – starting with shorter vertebra first and the longest spines
- Pairs of states are merged using the same deterministic merge procedure as in RPNI
- Sets of pairs resulting in inconsistent automata are stored to skip repetitions.
- As a experience-based heuristic we prevent creating multiloops.

Main algorithm

Observation

Let S_+ be the language accepted by the piece's PTA pt, and $\mathcal{C}(k, td)$ the complexity of consistency check given training data td and DFA with k states. We can estimate upper bound on the number of spines selected in Spine Compaction algorithm by $O(||S_+||^2)$. Then, the complexity of the Spine Compaction procedure is

$$O((||S_+||^2 + \mathcal{C}(||S_+||, td))||S_+||^2).$$

Experiments and Empirical Evaluation

Experiments

Games

We used 8 fairy chess games, containing altogether 41 pieces

- chess, Los Alamos, Tamerlane, Breakthrough Checkers
- 3 PCG games (including Legacy of Ibis)
- Special game containing *limited rider* and with a setup forcing the requirements for the theorems.

Test data

- based on plays between random agents
- learning performed for the first player only
- turnlimit set to 80
- *all legal moves* scenario:
  20 datasets per game, each containing 50 games
- *single move* scenario:
  20 datasets per game, each containing 1000 games

Results for the *all legal moves* scenario

Consistency Check	Correct size (%)			Errors (%)		
	Bjö.	RPNI	SC.	Bjö.	RPNI	SC.
Björnsson	87.6	**91.5**	87.3	**4.9**	9.1	7.6
$Fractional_{0.5}$	87.6	**89.9**	84.9	**4.9**	6.3	7.6
$Fractional_{0.6}$	89.0	**89.8**	86.6	**2.4**	4.0	5.0
$Fractional_{0.7}$	**87.1**	**87.1**	**87.1**	**2.4**	3.5	3.2
$Fractional_{0.8}$	**85.6**	85.5	**85.6**	**2.4**	3.7	3.2
$Fractional_{0.9}$	**73.7**	73.5	**73.7**	**2.4**	2.7	3.9
Fast	**69.3**	**69.3**	**69.3**	**0**	**0**	**0**

Correct size definition

Generated automata has a *correct size* if it has the same size as the optimal piece's recognizing DFA.

Results for the *single move known* scenario

Consistency Check	Correct size (%)			Errors (%)		
	Bjö.	RPNI	SC.	Bjö.	RPNI	SC.
Björnsson	**73.7**	70.6	**73.7**	**5.4**	19.8	9.6
$Fractional_{0.6}$	**88.7**	61.5	87.3	6.8	40.4	**6.1**
$Fractional_{0.7}$	**89.9**	65.7	89.8	4.9	37.6	**2.8**
$Fractional_{0.8}$	**88.0**	64.6	**88.0**	4.5	34.0	**4.4**
$Fractional_{0.9}$	**73.3**	71.5	**73.3**	3.7	14.4	**3.4**
Fast	**68.5**	**68.5**	**68.5**	**0**	**0**	**0**

Error definition

By an *error*, we mean a non-empty intersection with S_{-} of the piece's true language.

Learning times (in seconds)

Consistency Check	*all legal moves* Bjö.	RPNI	SC.	*single move known* Bjö.	RPNI	SC.
Björnsson	68.2	16.4	**4.1**	1598.6	454.1	**82.4**
Fractional$_{0.5}$	51.8	9.4	**2.2**			
Fractional$_{0.6}$	50.8	9.3	**2.1**	5.4	4.4	**0.2**
Fractional$_{0.7}$	50.8	9.2	**2.0**	5.6	3.3	**0.2**
Fractional$_{0.8}$	49.4	9.5	**1.8**	5.7	3.0	**0.2**
Fractional$_{0.9}$	42.8	10.4	**1.5**	5.7	2.6	**0.2**
Fast	4.0	4.6	**0.4**	4.6	5.3	**0.4**

Conclusions

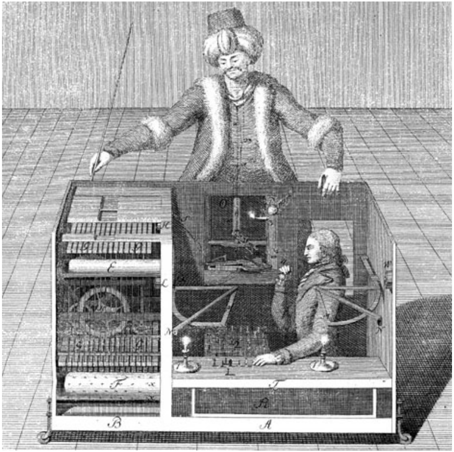
SpineCompaction learning algorithm

- provides most accurate results in the *single move known* scenario,
- comparable although more erroneous results in the GGP-like *all legal moves known* scenario,
- requires significantly much less time for learning.

Experiments show that in practice gathered observations data are incomplete, which justifies the need for a learning function based on some approximation method.

FractionalCheck$_\alpha$ and *FastCheck* consistency checking methods

- we can select an α value giving better results then the reference algorithm.
- they are also much faster.



Thank you