Procedia Computer Science
Volume 80, 2016, Pages 257–268
ICCS 2016. The International Conference on Computational Science

On Parallel Induction of Nondeterministic Finite Automata

Tomasz Jastrzab[1]

Silesian University of Technology, Gliwice, Poland
Tomasz.Jastrzab@polsl.pl

Abstract
The paper discusses the problem of the induction of a minimal nondeterministic finite automaton (NFA) consistent with a given set of examples and counterexamples. The main contribution of the paper is the proposal of an efficient parallel algorithm transforming the NFA induction problem into a family of constraint satisfaction problems (CSP). Two original techniques for fast CSPs evaluation are proposed and discussed. The efficacy of the parallel algorithm is evaluated experimentally on selected benchmarks. The proposed algorithm solves all analyzed benchmark examples, so called Tomita languages, in the time below half a minute, which should be considered an important achievement, as compared to our previous efforts which required minutes or hours to solve some of the aforementioned benchmarks.

Keywords: nondeterministic finite automata, parallel algorithms, constraint satisfaction

1 Introduction

Grammatical inference is the intensively studied field of research with a variety of applications including speech recognition, syntactic and structural pattern recognition, machine learning and others. The main task of grammatical inference is to provide, given some limited input data, some broader information on them. In this paper we study the problem of finding a nondeterministic finite automaton (NFA) consistent with a given finite set of examples and counterexamples. We are interested in inducing an automaton of a minimum size in terms of the number of states. Our research is motivated by the following fact. Namely, there are problems such as the induction of ordinary generating functions [19], for which the induction of a minimal automaton and/or grammar is desired, particularly if the size (length) of input data is limited. Moreover, as stated in [12], minimal automata and/or grammars help to analyze the structures of languages or mechanisms of grammars. An exemplary application in which minimal automata can be helpful in analyzing the structures of languages can be the analysis of peptide sequences. Being able to induce an automaton describing the structures of amyloid (harmful) and non-amyloid peptides, we can better understand the rules of harmful peptide sequences creation and possibly also predict some other potentially harmful sequences.

doi:10.1016/j.procs.2016.05.318

In particular, we can apply the proposed algorithm to the analysis of hexapepetides [3], i.e. peptides consisting of six amino acids, since they satisfy the assumption of limited input data length mentioned above.

A nondeterministic finite automaton A is defined as a quintuple $A = \{Q, \Sigma, \delta, q_0, Q_F\}$, where Q is the finite set of states, Σ is the alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the starting state and $Q_F \subseteq Q$ is the set of accepting states [11]. The NFA is said to accept a word w iff the word can be spelled out on at least one path starting in the initial state q_0 and ending in one of the final states $q \in Q_F$, where the path is defined as a set of successive transitions performed according to the rules specified by δ. Taking the above into account the following definition may be formulated. Let $S = \{S_+, S_-\}$ be a finite input sample, where S_+ denotes examples and S_- denotes counterexamples. Then the task of inducing an NFA consistent with S is to find an automaton A such that all examples and no counterexamples are accepted by the automaton.

The problem of minimal nondeterministic automata induction is known to be computationally hard. The problem of deterministic finite automata (DFA) induction has been proved to be NP-hard [10] while the decision problem of whether a DFA can be converted into minimal NFA is PSPACE-complete [14]. Also the induction of a minimal regular expression is PSPACE-complete. Finally, as stated in [2], the problem of the minimization of a regular expression for an arbitrary input sample is NP-hard. Thus we conjecture that induction of a minimal NFA consistent with given sample is also of that complexity.

The main contribution of the paper is the proposal of a parallel algorithm reducing the problem of minimal NFA induction into a family of constraint satisfaction problems (CSP). The CSP provides the answer whether it is possible to induce a k-state nondeterministic finite automaton consistent with given input sample, where k is a given positive integer. The advantage of the proposed transformation is that it allows to capture precisely the details of the automaton's transition function. In other words, by solving the CSP we are not only providing the answer if a consistent k-state NFA exists, but we are also constructing the automaton at the same time. In the paper we present several useful remarks and basing on these considerations we construct the parallel algorithm for building and solving the CSP. We present also a couple of original techniques for speeding up the computation. Finally, we report the experimental results obtained to evaluate the efficiency of the proposed parallel algorithm.

The paper is organized into 5 sections. Section 2 presents a brief review of the literature related to automata induction indicating the works that are most relevant to our research. Section 3 discusses the parallel algorithm we propose. Section 4 presents experimental results obtained for selected benchmarks. Section 5 contains summary and conclusions.

2 Related Works

The induction of deterministic and nondeterministic finite automata is a well-studied problem presented in the literature. DFAs are typically induced using state clustering algorithms like Regular Positive Negative Inference (*RPNI*) [7, 16] or Evidence Driven State Merging (*EDSM*) [15]. These algorithms begin with constructing a prefix tree acceptor for the input sample. The prefix tree acceptor is then successively collapsed by merging redundant states in a way that keeps the merged automaton consistent with the sample.

The solutions based on state merging that induce NFAs tend to actually produce on the output certain subclasses of NFAs. These include Residual Finite State Automata (RFSA), provided for instance by *DeLeTe2* [6] and Non-Deterministic Regular Positive Negative Inference (NRPNI) [1] algorithms, or Unambiguous Finite Automata (UFA) [4]. There are also methods proposed by García et al. [8, 9] which allow to induce valid NFAs independently of the order in which state merging is performed.

Apart from the state merging algorithms mentioned above, NFAs can also be induced by generating minimal nondeterministic subautomata corresponding to respective examples and being consistent with the set of counterexamples [5]. The work that is most related to ours is the work by Wieczorek [18]. The paper [18] proposes a transformation of the minimal NFA induction problem into an instance of Integer Nonlinear Programming (INLP) problem. An extension to this proposal, which allows to significantly reduce the size of the solution space of the INLP problem has recently been presented [13]. Our current approach differs from the previous works in the following respects:

1. While [18, 13] use three types of variables, we build a family of CSPs which are based on only one variable type. A fact worth noticing is that thanks to the original optimization techniques proposed in the paper we can recompute the CSPs on-the-fly without significant increase in the computation time.
2. Our CSPs capture *directly* the details of the automaton's transition function, while [18, 13] use only *indirect* information. The indirect information is represented by one of the variable types used in [18, 13] which we sucessfully remove from the model.

The fact that the transition function can be directly captured is an advantage of the proposed model, as mentioned in Sect. 1, since it provides a two-level information. First, it states that certain word is accepted (or rejected) by the automaton. Second, for an accepted word, it straightforwardly provides the actual path, which accepts the word. Contrarily, the proposal in [18] considers the acceptance of the word and the acceptance path separately. Thus we conjecture that our model should allow for faster NFA induction.

3 The Parallel Algorithm

3.1 Preliminaries

To make the paper self-contained let us begin with presenting the reduced INLP model discussed in [13], which was the starting point for the developments shown in the current paper. Let $S = \{S_+, S_-\}$ denote the input sample, with S_+ being the set of examples and S_- being the set of counterexamples. Let P denote the set of prefixes of all words $w \in S - \{\lambda\}$. Let $A = \{Q, \Sigma, \delta, q_0, Q_F\}$ denote the induced nondeterministic finite automaton. Then the following variable types can be defined:

- x_{wq}, which denotes whether word $w \in P$ can be spelled out on the path between state q_0 and state $q \in Q$,
- y_{apq}, which denotes whether there exists a transition between states $p, q \in Q$ using symbol $a \in \Sigma$,
- z_q, which denotes whether state $q \in Q$ is final, i.e. whether $q \in Q_F$ holds.

Let us observe that the domains of all the above variables contain only binary values, i.e. $D_x = \{0, 1\}$, $D_y = \{0, 1\}$ and $D_z = \{0, 1\}$, for all x, y and z variables. Assuming the variable definitions given above let us formulate the following constraints:

- if $\lambda \in S$ then

$$z_0 = 1 \text{ if } \lambda \in S_+, \text{ or } z_0 = 0 \text{ if } \lambda \in S_- \tag{1}$$

- for all words $w \in S_+ - \{\lambda\}$ (examples)

$$\sum_{q \in Q} x_{wq} z_q = 1 \tag{2}$$

- for all words $w \in S_- - \{\lambda\}$ (counterexamples)

$$\sum_{q \in Q} x_{wq} z_q = 0 \tag{3}$$

- for all words $w \in P$, such that $w = a$, for $a \in \Sigma$

$$x_{wq} = y_{wq_0q} \tag{4}$$

- for all words $w \in P$, such that $w = va$, for $v \in P, a \in \Sigma$

$$x_{wq} = \sum_{p \in Q} x_{vp} y_{apq} \tag{5}$$

Let $k = |Q|$, $l = |\Sigma|$, $m = |P|$ and $n = |S|$. Then to make the description complete note that Eqs. (1)–(5) use $k(m + kl + 1)$ variables, out of which $k(kl + 1)$ are independent variables (these are variables y and z). Moreover the number of equations is equal to $km + n + 1$.

Let the symbols of the alphabet be sorted and let each symbol be identified by its position within this ordered set. Then an index from the range $0 \ldots k^2l - 1$ can be assigned to each y_{apq} variable. We assume that the assignment is such that a unit change in $a \in \Sigma$ changes the variable index by k^2, a unit change in $p \in Q$ changes the variable index by k and a unit change in $q \in Q$ changes the variable index by 1, as suggested by Wieczorek (priv. comm.). Note also that the index assignment is reversible, e.g. given an index i the symbol of the alphabet can be determined by performing an integer division of i by k^2.

3.2 Remarks

Let us now present some remarks related to Eqs. (1)–(5), which led to the construction of the parallel algorithm described in section 3.3.

Remark 1. *Out of Eqs. (1)–(5) only first three equations impose the constraints on the NFA construction. However, their role in the transformation into the family of CSPs is different. Firstly, Eq. (1) is optional in the meaning that it is valid if and only if $\lambda \in S$. Secondly, if present, it reduces the number of feasible CSPs, since it forces the value of z_0 variable. On the other hand, Eqs. (2) and (3) define the CSP constraints, which describe the consistency of the automaton with the input sample.*

Remark 2. *Equations (4) and (5) can be treated as definitions of respective x variables. Furthermore, any occurrence of an arbitrary x variable defined by Eq. (4) can be replaced with the corresponding y variable. The replacement relates to Eqs. (2), (3) and (5). The advantages of the replacement are that: (i) the length of Eqs. (2), (3) and (5) remains unchanged and (ii)* *the total number of variables is reduced by at most kl. Moreover it can be observed for all x variables defined by Eq. (5) that*

$$x = \sum \left(\prod_{0 \le i \le k^2 l - 1} y_i \right) \tag{6}$$

Finally, the variables defined by Eq. (6) can be substituted into Eqs. (2) and (3) effectively reducing the number of constraints to at most $n+1$ equations and $k(kl+1)$ variables as mentioned in section 3.1.

It is worth to comment on the exact form of Eq. (6). The sum of products of y variables given by Eq. (6) can be constructed by recursive replacement of x variables according to the increasing length of the prefix corresponding to given x. Moreover, y variables participating in any Eq. (6) are restricted to the variables corresponding to the symbols appearing in the prefix defined by x variable.

Example 1. Let $\Sigma = \{a, b, c\}$, $k = 2$, $w = abc$, $q = 1$ be given. Then Eq. (6) corresponding to $x_{wq} = x_{abc1}$ can be constructed as follows. From (5) it follows that $x_{abc1} = x_{ab0}y_{c01} + x_{ab1}y_{c11}$. Then x_{ab0}, x_{ab1} will be $x_{ab0} = x_{a0}y_{b00} + x_{a1}y_{b10}$, $x_{ab1} = x_{a0}y_{b01} + x_{a1}y_{b11}$ (by Eq. (5)). Finally, from (4), variables x_{a0}, x_{a1} can be substituted with $x_{a0} = y_{a00}$ and $x_{a1} = y_{a01}$. Considering the above substitutions the final form of x_{abc1} will become

$$x_{abc1} = y_{a00}y_{b00}y_{c01} + y_{a01}y_{b10}y_{c01} + y_{a00}y_{b01}y_{c11} + y_{a01}y_{b11}y_{c11},$$

which after numbering the y_{apq} variables as described in Sect. 3.1, will lead to Eq. (6) in the form

$$x_{abc1} = y_0y_4y_9 + y_1y_6y_9 + y_0y_5y_{11} + y_1y_7y_{11} \ .$$

Remark 3. *Let x be defined according to Eq. (6). Then the total length of an arbitrary Eq. (6) is $|w|k^{|w|-1}$, for any prefix $w \in P$, $|w| > 1$. It follows from the fact that the number of possible y variables at the first and the last position of a single product term is equal to k and each of these variables appears in $k^{|w|-2}$ terms. On the other hand there are k^2 different variables y appearing at positions $2 \dots |w| - 1$. Each of them appears in $k^{|w|-3}$ terms.*

Remark 4. *It follows from Remark 3 that after substitution of x variables into Eqs. (2) and (3) the length of any of these equations becomes equal to $(|w| + 1)k^{|w|}$. This clearly means that unless k and $|w|$ are bounded by some small constant, the resulting equation lengths indicate that it will be impossible to store all the expressions in memory. However, as we will show in section 3.3 the substitution can be performed efficiently on-the-fly reducing the memory requirements significantly. Let us also observe that for a single Eq. (2) or (3) the substituted x variables differ only in the last y variable of the product terms, thus it is enough to compute the product terms for a single x and then replace the last term with a proper y variable.*

Example 2. Continuing Example 1 let us assume that word $w = abc$ belongs to the set S_+. Then Eq. (2) for this word will be $x_{abc0}z_0 + x_{abc1}z_1 = 1$, which after computing x_{abc0} by analogy to x_{abc1} and substituting both variables will become:

$$\begin{gathered} y_0y_4y_8z_0 + y_1y_6y_8z_0 + y_0y_5y_{10}z_0 + y_1y_7y_{10}z_0 + \\ y_0y_4y_9z_1 + y_1y_6y_9z_1 + y_0y_5y_{11}z_1 + y_1y_7y_{11}z_1 = 1 \end{gathered}$$

Remark 5. *The number of CSPs resulting from the transformation of the NFA induction problem depends only on the number of states k. This is because the algorithm presented in Sect. 3.3 considers all possible combinations of final/non-final states, i.e. all possible assignments of values to z variables. The number of CSPs ranges from $2^{k-1} - 1$ up to $2^k - 1$, which comes from the following facts: (i) if $\lambda \in S$ then the number of feasible combinations is automatically cut by half, (ii) the combination of $z_q = 0$ for all $q \in Q$ can never be true. Depending on the particular assignment of z variables, the number of product terms to be generated can be further reduced, since $z_q = 0$ for an arbitrary $q \in Q$ makes the product terms defined by Eq. (6) irrelevant for the constraint satisfiability.*

To explain the significance of the assignment $z_q = 0$ let us recall Examples 1 and 2 once again. It can be easily observed that if e.g. $z_0 = 0$ then the first four terms of the expression from Example 2 will always be zero and so will be all x_{w0} variables, for any $w \in S$. Thus these variables may be removed from further processing. On the other hand, if $z_q = 1$ then the satisfiability of Eqs. (2) or (3) will depend only on y variables corresponding to this z_q variable.

It can be concluded from the above remarks that the problem of NFA induction can be transformed into a family of CSPs described by $n = |S|$ equations over $k^2 l$ variables y, where $l = |\Sigma|$ and $k = |Q|$. CSP equations take the final form of:

$$\sum \left(\prod_{0 \le i \le k^2 l - 1} y_i \right) = 1 \ , \tag{7}$$

for the examples and

$$\sum \left(\prod_{0 \le i \le k^2 l - 1} y_i \right) = 0 \ , \tag{8}$$

for counterexamples. The product terms appearing in Eqs. (7) and (8) correspond to these z_q variables for which $z_q = 1$, as explained above.

3.3 The Algorithm

Let us now present the parallel algorithm responsible for building and solving the CSPs. Since each CSP can be built and solved independently we assume that the theoretical upper limit on the number of threads used corresponds directly to the number of possible CSPs, as given by Remark 5. The algorithm determines first the number of feasible combinations of z values and constructs the CSPs resulting from different value assignments. Each CSP is then dynamically assigned to a separate thread, which uses backtracking according to the description that follows. The backtracking procedure (Algorithm 1) takes as arguments the current state of the CSP, and either the index of y variable for which the assignment $y = 1$ was performed, or the value of -1, if the value of $y = 0$ was established. Variable k, representing the number of states, is assumed to be global.

The backtracking algorithm (Algorithm 1) is constructed according to the following rules:

1. Whenever value 1 is assigned to any y variable it is required to evaluate equations defined by (8) that correspond to words containing the symbol resulting from y (lines 4–7). If any product term appearing in Eq. (8) evaluates to 1, we need to backtrack, as the assignment cannot lead to a valid CSP solution (line 5).

```
1  Procedure ParallelInduction($csp$, $idx$)
       input: $csp$ – current state of Eqs. (7) and (8) and $y$ variables
              $idx$ – index of most recently set variable $y = 1$, or $-1$ if $y = 0$
2      if $idx <> -1$ then
3          $s \leftarrow idx/k^2$;                // finds symbol for given variable index
4          evaluate Eqs. (8) for words $w$ containing $s$;
5          if any Eq. (8) evaluates to 1 then
6              return;
7          end
8          evaluate Eqs. (7) for words $w$ containing $s$;
9          if all Eq. (7) evaluate to 1 then
10             $found \leftarrow$ true; return;
11         end
12     end
13     $idx \leftarrow$ next unset $y$; $s \leftarrow idx/k^2$; $y_{idx} \leftarrow 0$;
14     evaluate Eqs. (7) for words $w$ containing $s$;
15     if any Eq. (7) evaluates to 0 then
16         return;
17     end
18     ParallelInduction($csp$, $-1$);
19     if $found$ = false then
20         $y_{idx} \leftarrow 1$; ParallelInduction($csp$, $idx$);
21     end
22     if $found$ = false then
23         unset $y_{idx}$;
24     end
```

Algorithm 1: The backtracking algorithm

2. Whenever value 1 is assigned to any y variable and backtracking was not performed in step 1, it is also required to evaluate equations defined by (7) that correspond to words containing the symbol resulting from y (lines 8–11). If any product term appearing in Eq. (7) evaluates to 1, we can remove the equation from the model, as it is already satisfied. If no unsatisfied equations (7) remain, the solution is found (line 10).

3. Whenever value 0 is assigned to any y variable it is required to evaluate equations defined by (7) that correspond to words containing the symbol resulting from y (lines 14–17). If for any Eq. (7) all product terms evaluate to 0, we need to backtrack, as the assignment cannot lead to a valid CSP solution (line 15).

4. Since all equations defined by Eq. (7) have to be satisfied, we order the y variables based on the examples only. The variable ordering scheme selects Eq. (7) with the fewest product terms. The initial number of product terms can be evaluated basing on Remarks 3 and 4. Then, from the y variables appearing in the selected equation we choose the variable with the largest number of occurrences in this equation and we set it to 0. To follow the scheme we have to know the current counts of all y variables for each Eq. (7) and the current number of product terms in the equation (these counts are stored in *csp* variable being the argument of ParallelInduction procedure). The counts change when value 0 is assigned to some variable y (line 14).

Let us provide some details of the evaluation of product terms in steps 1–2 of the backtracking algorithm. As mentioned in Remark 4 the product terms have to be evaluated on-the-fly due to memory limitations. To make the evaluation fast, we propose the following technique:

- Let I denote the set of position(s) within word w of the symbol corresponding to the most recently set y variable.
- For each $i \in I$
  - if $i < |w|/2$ evaluate first the product terms for the prefix of word w ending at position $i-1$. If the prefix part evaluates to 1, evaluate the product terms for the suffix of word w starting at position $i+1$.
  - if $i \geq |w|/2$ reverse the order of evaluation.

  The evaluation of the prefix (resp. suffix) product terms stops whenever at least one product term evaluates to 1.

The advantage of the proposed evaluation technique is that it can significantly reduce the number of product terms that have to be computed as compared to direct on-the-fly evaluation of all possible product terms for given word. This effect is achieved thanks to the evaluation of the "shorter" part first (either the prefix or the suffix part, whichever is shorter).

Let us also comment on the proposed variable ordering scheme (see rule 4). We conjecture that ordering the equations according to the number of their product terms should be perceived as a fail-fast technique. Since each Eq. (7) has to be satisfied it is best to start with the equation with the fewest possibilities. Furthermore, selecting the most common y variable for such an equation allows to quickly remove majority of the product terms and also leads to fail-fast behavior. The main limitation of this proposal could be the requirement for an up-to-date information on the counts of variables and product terms. To satisfy this requirement, the CSP model would have to be recomputed after each assignment of the form $y = 0$ (since this makes some of the product terms irrelevant for the equation satisfiability). However, this problem may be easily solved by applying the following updating technique:

- For all states $q' \in Q$ compute the counts of y variables appearing in the prefix of word w and store in an array cnt_p. The length of the prefix should be equal to $|w|/2$. The starting state of the first y variable should always be 0, while the ending state of the last y variable should be q'. Store the total number of generated product terms in variable $total_p$.
- For all states $q'' \in Q$ compute the counts of y variables appearing in the suffix of word w and store in an array cnt_s. The suffix should not include the last symbol of word w. Therefore the length of the suffix should be equal to $|w|/2 - 1$. The starting state of the first y variable should be q', while the ending state of the last y variable should be q''. Store the total number of generated product terms in variable $total_s$.
- Compute and store the variable counts $var^{q''}[i] = cnt_p[i] \cdot total_s + (total_p - cnt_p[i]) \cdot cnt_s[i]$, for $0 \leq i \leq k^2 l - 1$, separately for each q''. Compute and store the total number of product terms $total^{q''} = total_p \cdot total_s$, separately for each q''.
- Compute the final counts by summing up the counts obtained for particular states $q'' \in Q$. However, if the summation relates to a variable y corresponding to the last symbol of word w take the count to be equal to the total number of product terms for this q'' instead of the computed count for y.

The greatest advantage of the proposed updating method is that it significantly reduces the computational effort. This is because instead of computing at most $k^{|w|}$ product terms it is enough to compute at most $(2k^{|w|/2} + k^2)$ product terms. What is more, the method can be easily extended to further divide the prefix and/or suffix parts if required. As the experiments have also shown, without this technique it would be impossible to solve majority of the benchmarks in reasonable time (i.e. time of the order of minutes).

It should be clear that the proposed algorithm and optimization techniques minimize the computational effort. It should be also noted that by evaluating each feasible CSP concurrently, one should be able to reduce the execution time significantly.

4 Experiments

The experiments were conducted using a Java-based implementation of the parallel algorithm described in Section 3.3. The test machine was equipped with Intel Xeon E5-2640 2.60GHz processor with 8 physical (16 logical) cores and 8 GB of RAM memory. The time measurements were performed using *System.nanoTime()* function. Each experiment was repeated five times and the minimum observed values were taken as the final value reported in the paper. The time was measured from the moment the first thread started execution to the moment the last thread finished its computation.

The benchmarking set used in the experiments was selected to be the set of 14 Tomita languages reported in [17], including 7 basic problems and 7 inverse problems, denoted in the sequel with superscripts b and i, respectively. The languages were defined by the examples and counterexamples over a two-symbol alphabet, and were referred to as the right- and wrong-list in [17]. The inverse problems were constructed from the basic problems by swapping the right- and wrong-lists. The benchmarking set was selected because of two reasons. First, the sizes of minimal NFAs are known, since exemplary solutions have been shown in [17]. Second, the results for the set of basic examples were also reported in [13, 18]. Therefore our current approach can be compared with the ones presented in the literature.

Table 1 summarizes the characteristics of the 14 problems considered. We use Σ to denote the input alphabet, $n^b_+, n^i_+, n^b_-, n^i_-$ to denote the cardinalities of the set of examples S_+ and counterexamples S_- for the basic and inverse problems, $l^b_+, l^i_+, l^b_-, l^i_-$ to denote the total lengths of examples and counterexamples, and k^b, k^i to denote the size of the minimal NFA.

Problem	Σ	$n^b_+ = n^i_-$	$n^b_- = n^i_+$	$l^b_+ = l^i_-$	$l^b_- = l^i_+$	k^b	k^i
P1	$\{0,1\}$	9	8	36	29	1	2
P2	$\{0,1\}$	6	10	34	35	2	3
P3	$\{0,1\}$	13	12	59	67	4	5
P4	$\{0,1\}$	11	9	44	67	3	4
P5	$\{0,1\}$	10	12	52	54	4	4
P6	$\{0,1\}$	10	12	47	56	3	3
P7	$\{0,1\}$	13	8	58	61	4	5

Table 1: Benchmark set characteristics

Table 2 contains the execution times (expressed in seconds) obtained for the parallel algorithm runs. Columns 2–6 denote the times measured for the execution with N threads. The number of threads was related to k, i.e. the size of the NFA, so that either all, half or only one CSP was considered simultaneously. For completeness, columns labelled $W(N)$ and $JCW(N)$

contain the times reported in [18] and [13], respectively, with the number of processors used, given in brackets. The '–' entries in Tab. 2 mean that the experiments for the given problem and/or the number of processors/threads were not conducted.

The algorithms proposed in [18, 13] were implemented in C++ and used Message Passing Interface (MPI) for communication. They used computer clusters Reef, Zeus and Galera (Poland), equipped mostly with Dual- and Quad-Core Intel Xeon 2.33 GHz processors connected with Infiniband network. The results for problems P3^b, P5^b and P7^b in column $W(N)$ were obtained using parallel tabu search heuristic algorithm.

Problem	$N = 1$	$N = 2$	$N = 4$	$N = 8$	$N = 16$	$W(N)$	$JCW(N)$
P1^b	0.034	–	–	–	–	< 1 (4)	< 1 (4)
P1^i	0.037	0.038	–	–	–	–	–
P2^b	0.041	0.036	–	–	–	< 1 (4)	< 1 (4)
P2^i	0.044	0.059	0.050	–	–	–	–
P3^b	2.749	–	2.237	2.296	–	1757 (80)	34 (30)
P3^i	0.266	–	–	0.358	0.465	–	–
P4^b	0.070	0.080	0.115	–	–	6 (120)	< 1 (4)
P4^i	0.466	–	0.583	0.518	–	–	–
P5^b	0.706	–	0.647	0.636	–	1688 (64)	155 (20)
P5^i	0.935	–	0.511	0.359	–	–	–
P6^b	0.054	0.053	0.057	–	–	1 (80)	< 1 (4)
P6^i	0.076	0.071	0.073	–	–	–	–
P7^b	5.280	–	3.378	3.746	–	1433 (72)	7287 (40)
P7^i	26.251	–	–	0.831	0.992	–	–

Table 2: Execution times for the benchmark languages (in sec.)

It can be observed from the obtained results that the proposed parallel algorithm brings substantial improvement to the efforts known from [18, 13]. We were able to find the minimal NFA for all considered examples in below half a minute, while in [18, 13] the times were usually much larger (of the order of minutes or even hours). In terms of speedups the results are not satisfactory, except for problems P5^i, P7^b and P7^i. However, poor speedup values can be explained by the following observation. Namely, we are looking for the first solution only. Since usually it is found for one of the first combinations of z values, increasing the number of threads does not bring much improvement. Nevertheless we believe that the parallelism can be important for other problems, in which more combinations of the final states have to be considered. The preliminary results of our research on hexapeptide sequences available in Waltz-DB database [3] support this belief.

Let us also comment on some of the anomalies that can be observed in Tab. 2.

It can be noticed that the algorithm shows some unexpected behavior for problems P3^i and P4^b, where we observe slowdowns as the number of threads increases. However, this behavior can be explained by the following observations. First, let us recall that the execution time is measured between the beginning of the first thread's execution and the ending of the last thread's execution. Thus, it can happen that when the solution is found in one of the threads, the remaining threads can still be performing their part of computation (e.g. evaluating some of the equations). Consequently it can take some time for them to terminate their execution, which will be reflected in an overall slowdown. Furthermore, as revealed by a detailed investigation of the experimental results, successive memory allocations become slower as the number of threads and the recursion level of the backtracking procedure grows. Thus, in some cases, the gain of finding the solution earlier may be lost due to the memory allocation overhead.

It can be also noticed that the algorithm achieves super-linear speedup in problem $P7^i$ for the transition between 1 and 8 threads. This anomaly can be justified as follows. The solution to the induction problem for given input sample does not have to be unique. Therefore, the algorithm running with 8 threads may find a different solution than its sequential version. Given that, and assuming that the other solution is found early, the super-linear speedup may be observed. Finally, following the explanations for problems $P3^i$ and $P4^b$, the slowdown that is observed for the change from 8 to 16 threads in problem $P7^i$ can again be explained by the memory allocation overhead and the way the time measurements were performed.

5 Summary and conclusions

In the paper we devise a parallel algorithm for solving the problem of minimal nondeterministic finite automata induction. We propose the method transforming the problem of NFA induction into a family of constraint satisfaction problems. We introduce two original techniques for efficient evaluation of the CSP model, which help to find the correct solution quickly. The algorithm is evaluated on a set of benchmarking languages proposed by Tomita.

The algorithm proves efficient and is able to solve all analyzed problems in a few seconds. This should be considered an important improvement with respect to the previous works on the subject. Furthermore, preliminary research on languages of other characteristics, e.g. having much larger alphabet (composed of 20 amino acid symbols) and/or larger total length of the input sample (about 3 times larger) are also promising. Although the results are not satisfactory in terms of speedups, the overall performance of the algorithm should be considered good. We plan to continue our research on other languages, in particular on (hexa-)peptides. We believe that the proposed parallel algorithm can help in extraction of some information encoded in the structure of hexapeptide sequences.

Acknowledgement

Computations were carried out using the infrastructure supported by POIG.02.03.01-24-099/13 grant: "GeCONiI—Upper Silesian Center for Computational Science and Engineering". This research was supported by the Institute of Informatics research grant no. BKM-515/RAU2/2015.

The author would like to thank the anonymous reviewers for their valuable remarks and suggestions to improve the quality of the paper.

References

[1] G. I. Alvarez, J. Ruiz, A. Cano, and P. García. Nondeterministic regular positive negative inference nrpni. In *Proceedings of the XXXI Latin American Informatics Conference (CLEI'2005)*, pages 239–249, 2005.

[2] D. Angluin. *An application of the theory of computational complexity to the study of inductive inference*. PhD thesis, University of California, USA, 1976.

[3] J. Beerten, J. Van Durme, F. Rousseau, and J. Schymkowitz. WALTZ-DB. database of amyloid forming peptides. [online]. http://waltzdb.switchlab.org/, last access: 30/01/2016.

[4] F. Coste and D. Fredouille. Unambiguous automata inference by means of state merging methods. In *Machine Learning: ECML 2003: 14th European Conference on Machine Learning*, volume 2837 of *LNCS (LNAI)*, pages 60–71, Heidelberg, 2003. Springer.

[5] M. Vázquez de Parga, P. García, and J. Ruiz. A family of algorithms for non deterministic regular languages inference. In *Proceedings of the 11th International Conference on Implementation and Application of Automata (CIAA 2006)*, volume 4094 of *LNCS*, pages 265–274, Berlin Heidelberg, 2006. Springer-Verlag.

[6] F. Denis, A. Lemay, and A. Terlutte. Learning regular languages using rfsas. *Theoretical Computer Science*, 313(2):267–294, 2004.

[7] P. Dupont. Regular grammatical inference from positive and negative samples by genetic search: the gig method. In *ICGI '94 Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, pages 236–245, London, UK, 1994. Springer-Verlag.

[8] P. García, M. Vázquez de Parga, G. I. Alvarez, and J. Ruiz. Learning regular languages using non-deterministic finite automata. In *Proceedings of the 13th International Conference on Implementation and Application of Automata (CIAA 2008)*, volume 5148 of *LNCS*, pages 92–101, Berlin Heidelberg, 2008. Springer-Verlag.

[9] P. García, M. Vázquez de Parga, G. I. Alvarez, and J. Ruiz. Universal automata and nfa learning. *Theoretical Computer Science*, 407(1–3):192–202, 2008.

[10] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):203–320, 1978.

[11] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.

[12] K. Imada and K. Nakamura. Learning context free grammars by using sat solvers. In *ICMLA '09: International Conference on Machine Learning and Applications*, pages 267–272, 2009.

[13] T. Jastrzab, Z. J. Czech, and W. Wieczorek. Parallel induction of nondeterministic finite automata. In *Proceedings of the 11th International Conference on Parallel Processing and Applied Mathematics (PPAM 2015)*, 2015. in press.

[14] T. Jiang and B. Ravikumar. Minimal nfa problems are hard. *SIAM Journal on Computing*, 22:1117–1141, 1993.

[15] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In *Proceedings of the 4th International Colloquium on Grammatical Inference*, pages 1–12. Springer-Verlag, 1998.

[16] J. Oncina and P. García. Inferring regular languages in polynomial updated time. In *Pattern Recognition and Image Analysis*, pages 49–60, Singapore, 1992. World Scientific.

[17] M. Tomita. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, pages 105–108, 1982.

[18] W. Wieczorek. Induction of non-deterministic finite automata on super- computers. In *Proceedings of the Eleventh International Conference on Grammatical Inference (ICGI 2012)*, volume 21 of *JMLR Workshop and Conference Proceedings*, pages 237–242, 2012.

[19] W. Wieczorek and A. Nowakowski. Grammatical inference in the discovery of generating functions. In *Man-Machine Interactions 4*, volume 391 of *Advances in Intelligent Systems and Computing*, pages 627–637. Springer International Publishing, 2016.