Parallel Computing: Technology Trends
I. Foster et al. (Eds.)

doi:10.3233/APC200039

On Superlinear Speedups of a Parallel NFA Induction Algorithm

Tomasz JASTRZĄB [a,1]
[a] *Institute of Informatics, Silesian University of Technology, Gliwice, Poland*

Abstract. The parallel induction algorithm discussed in the paper finds a minimal nondeterministic finite automaton (NFA) consistent with the given sample. The sample consists of examples and counterexamples, i.e., words that are accepted and rejected by the automaton. The algorithm transforms the problem to a family of constraint satisfaction problems solved in parallel. Only the first solution is sought, which means that upon finding a consistent automaton, the remaining processes terminate their execution. We analyze the parallel algorithm in terms of achieved speedups. In particular, we discuss the reasons of the observed superlinear speedups. The analysis includes experiments conducted for the samples defined over the alphabets of different sizes.

Keywords. parallel algorithm, superlinear speedup, nondeterministic automata induction, constraint satisfaction

1. Introduction

Deterministic and nondeterministic finite automata play a crucial role in various practical applications, including artificial intelligence, grammatical inference, and bioinformatics [1,2,3]. The last field of application is particularly interesting, as also stated in [4], since the automata can be used to detect patterns hidden in bioinformatics data. In this context, automata can act as classifiers for previously unseen sequences, or as generators, producing new sequences that may bear some biological meaning.

A nondeterministic finite automaton (NFA) is given by a tuple $A = (Q, \Sigma, \delta, q_0, Q_F)$, where Q is a finite set of states, Σ is an alphabet, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $q_0 \in Q$ is an initial state and $Q_F \subseteq Q$ is a set of final states [5]. A sample $S = (S_+, S_-)$ consists of two sets of words, where a word w is a finite sequence of symbols defined over the alphabet Σ, set S_+ contains examples, while set S_- contains counterexamples.

The aim of the parallel induction algorithm is to find a minimal NFA consistent with the given sample S. The automaton is *consistent* with S iff it accepts all the examples and rejects all the counterexamples. A word w is accepted by the automaton A iff there exists a sequence of transitions between state q_0 and at least one state $q \in Q_F$ on which the word is read. Otherwise, the word is rejected. The automaton is consistent and *minimal* iff no two states can be merged together without losing the consistency.

[1]Corresponding Author: Tomasz Jastrząb, Institute of Informatics, Silesian University of Technology, ul. Akademicka 16, 44-100 Gliwice, Poland; E-mail: Tomasz.Jastrzab@polsl.pl.

The induction of minimal consistent NFA is known to be hard. It was shown that NFA minimization is impossible from polynomial time and data [6]. It was also shown that even if the sample is given in the form of a deterministic finite automaton, the problem is PSPACE-complete [7]. Hence, the use of parallel computing is vital for efficient solving of the problem at hand.

In the paper, we study a parallel algorithm for solving the minimal NFA induction problem. The algorithm constructs first a set of independent constraint satisfaction problems (CSP), described in detail in Section 2, and then solves them in parallel. We also consider a modified version of the algorithm in which each CSP is solved independently according to a number of different variable orderings. Furthermore, we discuss the implications of the use of shared and distributed memory models, including the issues of distributed computation termination and the overhead of interprocess communication.

The main contribution of the paper is the analysis of the achieved speedups. We focus in particular on the superlinear speedups[2] observed for certain samples. We provide explanations of the anomalies. We analyze the speedups in the function of the number of processes, but also with respect to the input samples. They differ in the sizes of the alphabets, the lengths of the examples and counterexamples and the sizes of the resulting automata. We consider different samples including the ones presented in the literature [9] and randomly generated based on the publicly available resources [10].

The rest of the paper is organized as follows. In Section 2 we present the problem formulation considered in the paper. In Section 3 we discuss the basic and modified parallel induction algorithms. Section 4 contains the results of the experiments and the discussion of the superlinear speedups. Finally, in Section 5 we present the conclusions.

2. Problem Formulation

The problem of minimal consistent NFA induction can be viewed from two different perspectives. Namely:

1. It is an *optimization problem*, if we first induce any consistent NFA, and later reduce it by merging redundant states.
2. It is a *decision problem*, if we first fix the number of states, and later search for a consistent automaton with the given number of states.

Note that in the first case, the final size of the automaton depends on the order in which the merges are performed. As a consequence, the resulting automaton need not be minimal. With the second approach, by taking the number of states to be $k = 1, 2, \ldots$, we not only find the consistent automaton for the given k, but we can also prove that it is indeed minimal, if no consistent NFA exists for $k-1$ states. However, even for the decision problem, the solution (i.e., the induced NFA) does not have to be unique.

There exists a number of algorithms following the first approach towards NFA induction mentioned above. They include the *DeLeTe2* algorithm [11], Nondeterministic Regular Positive Negative Inference (NRPNI) [12], and the state merging algoriths based on the notions of unambiguous [13] or universal [14] automata. The algorithm discussed in [14] has been extended in [15], to produce an algorithm that is independent of the or-

[2]A superlinear speedup occurs when the achieved speedup is greater than the number of used processes. For more information see [8].

der in which the merges are performed. Yet another approach was taken in [16], in which subautomata consistent with the set S_- were generated for each member of the set S_+, and were later removed when an example was accepted by a different subautomaton.

The decision problem formulation was pursued in [17], in which the basic encoding of the induction problem as a CSP was proposed. The encoding was later improved in [18,19], which allowed for a significant reduction of the solution space size. The impact of the selected variable ordering schemes on the performance of parallel induction algorithms was also investigated in [20,4]. Finally, some considerations related to the possibility of using multiple variable orderings at the same time were presented in [21]. In the current paper, we further elaborate on this possibility in terms of achieved speedups.

Let us now recall the CSP-based formulation of the induction problem solved by the parallel algorithms described in Section 3. The description is based on [19] and corresponds to the decision problem stated before. Let k be the given number of states and l be the size of the alphabet. We assume two types of binary variables y and z. Variables y_i, $i = 0, 1, \ldots, k^2 l - 1$, denote the elements of the transition function δ, and variables z_j, $j = 0, 1, \ldots, k-1$, mark the states as final or non-final. Let Σ be ordered lexicographically and let $loc(a)$ denote the zero-based position of a symbol a within Σ. Then each index i of a variable y_i, corresponding to a transition $q_m \xrightarrow{a} q_n$, $q_m, q_n \in Q$, is given by [17]:

$$i = k^2 \cdot loc(a) + k \cdot m + n. \tag{1}$$

Given the variables defined above, the consistency of the automaton with the sample $S = (S_+, S_-)$ is defined as follows:

1. If set S_+ or set S_- contains the empty word λ, then $z_0 = 1$, for $\lambda \in S_+$ (the empty word is accepted), and $z_0 = 0$, for $\lambda \in S_-$ (the empty word is rejected).
2. For all examples, the word w is accepted by the NFA iff there exists a sequence of transitions over which word w is spelled out, provided that this sequence ends in a final state. Therefore, for each $w \in S_+ \setminus \{\lambda\}$, it holds that:

$$\bigvee_{j=0..k-1} \left(\bigvee_{1..k^{|w|-1}} (y_{i_1} \wedge y_{i_2} \wedge \ldots \wedge y_{i_{|w|}}) \right) \wedge z_j = 1, \tag{2}$$

   where $i_1, i_2, \ldots, i_{|w|}$ are the indices of y_i variables computed according to Eq. (1), for $0 \leq m, n < k$ and $a \in \Sigma$ appearing in word w.
3. For all counterexamples, the word w is rejected by the NFA iff no sequence of transitions over which word w is spelled out exists, or such a sequence ends in a non-final state. Therefore, for each $w \in S_- \setminus \{\lambda\}$, it holds that:

$$\bigvee_{j=0..k-1} \left(\bigvee_{1..k^{|w|-1}} (y_{i_1} \wedge y_{i_2} \wedge \ldots \wedge y_{i_{|w|}}) \right) \wedge z_j = 0, \tag{3}$$

   where $i_1, i_2, \ldots, i_l$ are defined as before.

Example 1. To clarify Eqs. (2) and (3) let us consider the following example. Let the sample be $S = (\{a, aa, ba, bba\}, \{\lambda, b, ab\})$ and let $k = 2$. Since $\lambda \in S_-$ we have $z_0 = 0$.

Since $z_1 = 0$ cannot lead to a valid solution (no word would be accepted), we set $z_1 = 1$. Equations (2) and (3), after applying values of z_0 and z_1, take the following form:

for word *a*:	$y_1 = 1$
for word *aa*:	$y_0 \wedge y_1 \vee y_1 \wedge y_3 = 1$
for word *ba*:	$y_4 \wedge y_1 \vee y_5 \wedge y_3 = 1$
for word *bba*:	$y_4 \wedge y_4 \wedge y_1 \vee y_4 \wedge y_5 \wedge y_3 \vee y_5 \wedge y_6 \wedge y_1 \vee y_5 \wedge y_7 \wedge y_3 = 1$
for word *b*:	$y_5 = 0$
for word *ab*:	$y_0 \wedge y_5 \vee y_1 \wedge y_7 = 0$

After solving the above equations we get that $y_1 = 1$, $y_4 = 1$, $y_0 \vee y_3 = 1$, $y_5 = 0$, and $y_7 = 0$. It means that the resulting automaton contains the transitions $q_0 \xrightarrow{a} q_1$, $q_0 \xrightarrow{b} q_0$ and at least one of the transitions $q_0 \xrightarrow{a} q_0$ or $q_1 \xrightarrow{a} q_1$. Moreover, the automaton cannot contain the transitions $q_0 \xrightarrow{b} q_1$ and $q_1 \xrightarrow{b} q_1$. The existence of transitions related to variables y_2 (transition $q_1 \xrightarrow{a} q_0$) and y_6 (transition $q_1 \xrightarrow{b} q_0$) cannot be determined based on the given sample S. The example solutions are shown in Figure 1.

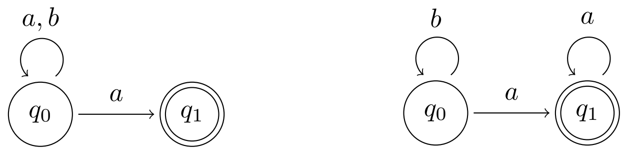

Figure 1. Automata consistent with sample S, in which $y_0 = 1, y_1 = 1, y_4 = 1$ (left), and $y_1 = 1, y_3 = 1, y_4 = 1$

3. Parallel Algorithms

Let us now discuss the basic parallel algorithm for solving the induction problem [21]. As already stated, it aims at solving independent CSPs in parallel to speed up the computation. Note that the algorithm, shown in Figure 2, searches for one solution only.

The algorithm BASICPARINDUCTION starts by checking if the value of variable z_0 can be established based on the presence of the empty word (line 2). Depending on the outcome of this check, it sets the number of possible CSPs n as follows: (i) $n = 2^k - 1$, for $\lambda \notin (S_+ \cup S_-)$, (ii) $n = 2^{k-1}$, for $\lambda \in S_+$, (iii) $n = 2^{k-1} - 1$, for $\lambda \in S_-$. These CSPs are then distributed among processes (line 3). Each process employs a backtracking procedure (lines 5–10), to find the assignments of values to y variables.

There are a few points about the algorithm shown in Figure 2 that are worth mentioning. First of all, structure Z_s is a k-element vector of z_j variables' values. Based on these values, Eqs. (2) and (3) are simplified by removing the terms for which $z_j = 0$ (see Example 1). Secondly, the way in which the CSPs are distributed among processes in the **parfor** loop (line 3) depends on the used memory model. For a shared memory model, new processes may be forked by a master process, while for a distributed memory model, the processes may be assigned to the Z_s vectors based on their ranks. In either case the interprocess communication overhead at this point is minimal. Thirdly, for each CSP both the y_i variables and their values are selected according to the given ordering scheme (line 6), which is the same for each CSP (see [21] for a discussion of other possibilities). Finally, since we search for the first solution, upon finding it the computation

```
1: procedure BasicParInduction($S, k$)
2:     if $\lambda \in (S_+ \cup S_-)$ then set $z_0$ accordingly
3:     parfor $s \leftarrow 1$ to $n$ do
4:         $Z_s \leftarrow$ assignment of values to $z_j$ variables, $0 \leq j < k$
5:         ▷ start backtracking procedure
6:         select next $y_i$ and assign value according to the given ordering
7:         evaluate Eqs. (2) and (3)
8:         if contradition found then change value or return to the previous $y_i$
9:         if solution found then notify other processes and terminate
10:        ▷ end backtracking procedure
11:    end parfor
12: end procedure
```

Figure 2. The basic parallel induction algorithm

terminates. The way in which the termination procedure is realized, depends again on the memory model used. In case of the shared memory model, it is enough to use a global Boolean flag protected against simultaneous read-write access by a mutex. In case of the distributed memory model, a message has to be sent to other processes, indicating that they may terminate their execution. However, to receive the message, each process has to periodically check for message arrival. Hence, the distributed memory model incurs some time overhead resulting from channel probing and interprocess communication.

The modified version of the parallel induction algorithm is shown in Figure 3. It applies multiple ordering schemes to each of the analyzed CSPs. The intuition behind this approach is that the "best" ordering is not known in advance, and it may differ between respective CSPs. Thus, to increase the chances for efficient computation, we employ multiple orderings to the same instance of the CSP. This way we also capitalize on the negative results, i.e., when the process using some ordering determines that no solution exists for the given CSP (given Z_s), it notifies the other processes working on the same CSP, that they should terminate their execution. This way, the time to solve a given CSP is shorter, and equal to the run time of the process using the "best" ordering.

Let us discuss the effects of the memory models on the MultiVOParInduction algorithm. The distribution of computation ocurrs in lines 3 and 5. Let n be the number of CSPs and m be the number of ordering schemes. Then in the shared memory model, we can fork nm processes, divide them into n groups working on the Z_s vectors and for each process in the given group apply a different ordering scheme for the same Z_s. In case of the distributed memory model, we can still use the process ranks, but this time we have to group the processes working on the same vector Z_s. As to the termination procedure, for the shared memory model, we need a set of global Boolean flags, one for each group of processes working on the same CSP, to indicate negative results. For the distributed memory model, we need to introduce a different message type for each group of processes, to indicate group termination, as opposed to global termination when the solution is found. Therefore, the overhead of interprocess communication does not change for the distributed memory model, while it increases for the shared memory model, due to the need for access synchronization to the group termination flag.

Figures 4 and 5 show the work distribution and interprocess communication related to the termination procedure for the two parallel algorithms. For the basic algorithm, we

```
1: procedure MultiVOParInduction($S, k$)
2:     if $\lambda \in (S_+ \cup S_-)$ then set $z_0$ accordingly
3:     parfor $s \leftarrow 1$to $n$ do
4:         $Z_s \leftarrow$ assignment of values to $z_j$ variables, $0 \leq j < k$
5:         parfor $t \leftarrow 1$to $m$ do
6:             ▷ start backtracking procedure
7:             select next $y_i$ and assign value according to the given ordering $t$
8:             evaluate Eqs. (2) and (3)
9:             if contradition found then change value or return to the previous $y_i$
10:            if solution found then notify other processes and terminate
11:            ▷ end backtracking procedure
12:        end parfor
13:        if solution not found then notify other processes working on $Z_s$ and terminate
14:    end parfor
15: end procedure
```

Figure 3. The modified parallel induction algorithm

assumed that the number of processes is equal to *n*, while for the modified version, this number is equal to *nm*.

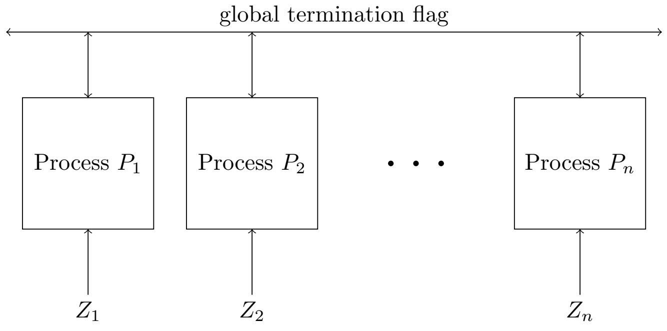

Figure 4. Work distribution and computation termination procedure for BasicParInduction algorithm

4. Experiments

The parallel algorithms were implemented in Java and executed on a pair of 12-core Intel Haswell 2.3 GHz processors with 128 GB RAM. The read-write access to the shared memory was protected using the `synchronized` keyword. The time measurements were performed using `System.nanoTime()` function.

The experiments were conducted for the selected Tomita languages [9] and for the samples built from the peptides listed in WALTZ-DB database [10]. The Tomita languages are defined over the alphabet $\{0, 1\}$, while the peptides are based on an alphabet of up to 20 symbols, representing amino acids. The summary of the differences between these two sample sources is shown in Table 1.

The experiments aimed at observing the speedups obtained by the basic and modified parallel algorithms. The algorithms used three different ordering schemes, namely

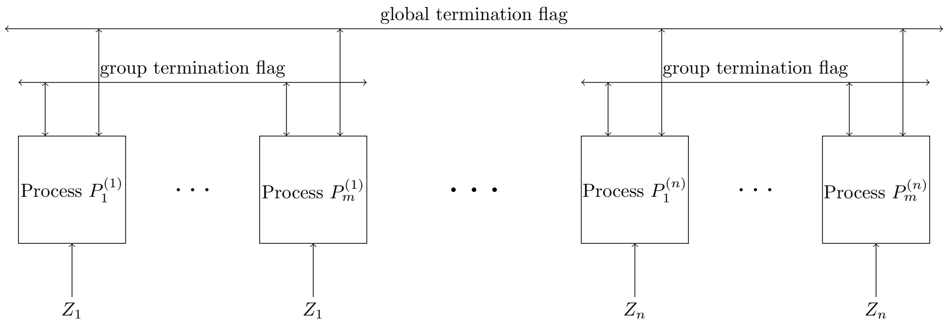

Figure 5. Work distribution and computation termination procedure for MULTIVOPARINDUCTION algorithm

Table 1. Comparison of sample characterisitcs based on Tomita languages and WALTZ-DB samples

Sample characteristic	Tomita languages	WALTZ-DB samples
Number of samples N	10	50
Number of states k	3–4	2–3
Alphabet size $\|\Sigma\|$	2	18–20
Sample size $\|S_+\| + \|S_-\|$	20–25	50
Word length $\|w\|$	0–18	5–6
Contains empty word $\lambda \in (S_+ \cup S_-)$	yes	no

the *deg* scheme [22], as well as the *min-max-ex* and *min-max-cex* schemes [4]. The *deg* scheme uses static ordering based on variable degree, while the other two schemes use dynamic ordering based on the examples and counterexamples, respectively.

In the first experiment we compared the basic parallel algorithm executed by a single process and by the number of processes corresponding to n. The distribution of obtained speedups, for different variable orderings, is shown in Figure 6. The box plots show the minimum and maximum speedup values (marked by the lines extending from the box), together with the first, second, and third quartile (marked by the box itself).

Based on the results shown in Figure 6 we noticed two kinds of anomalies. On the one hand, we observed slowdowns present mostly for the Tomita languages. On the other hand, we noted the superlinear speedups (up to 8500) in case of WALTZ-DB samples.

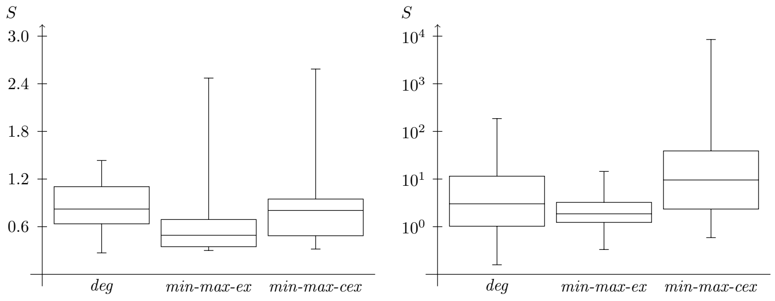

Figure 6. Speedups achieved by the BASICPARINDUCTION algorithm for the Tomita languages (left) and WALTZ-DB samples

The reason for the negative anomalies is that when the solution found during the sequential and parallel execution of the algorithm is the same, the latter approach introduces an overhead resulting from parallelism. Since the processes read from and write to the shared memory, access synchronization occurs. Furthermore, the NUMA architecture of the processors also affects the distribution of memory access times. This in turn introduces certain delays to the overall execution time. The reasons for positive anomalies are two-fold. Either there exists more than one solution for the given sample, or the solution is found for a CSP that is not the first one analyzed. In the former case, the algorithm executed in parallel allows to find the “simplest” solution, i.e., the solution that can be found in the shortest time. In the latter case, the parallel algorithm is able to bypass the “hard” CSP instances that have to be solved during the sequential run of the algorithm.

Example 2. Let us assume that a two-state automaton is sought. We consider three different CSPs resulting from the pairs of assignments $Z_0 = \langle 0,1 \rangle$, $Z_1 = \langle 1,0 \rangle$, and $Z_2 = \langle 1,1 \rangle$. Let us assume that the execution times are $\tau_0 = 10$ s, $\tau_1 = 1$ s, and $\tau_2 = 25$ s, and that a solution exists for the cases Z_0 and Z_1. The sequential execution takes 10 s (solution for Z_0 found), while the parallel execution for $n = 3$ processes takes only 1 s (solution for Z_1 found), which gives a speedup of 10.

Example 3. Let us assume that a two-state automaton is sought. We consider three different CSPs resulting from the pairs of assignments $Z_0 = \langle 0,1 \rangle$, $Z_1 = \langle 1,0 \rangle$, and $Z_2 = \langle 1,1 \rangle$. Let us assume that the execution times are now $\tau_0 = 25$ s, $\tau_1 = 1$ s, and $\tau_2 = 10$ s, and that a solution exists for the case Z_1. The sequential execution takes 26 s (cases Z_0 and Z_1 considered), while the parallel execution for $n = 3$ processes takes only 1 s (after solution for Z_1 is found all processes terminate), which gives a speedup of 26.

In the experiments performed for the WALTZ-DB samples, we counted 25, 34, and 34 cases in which a different solution was found by the sequential and parallel algorithm using *deg*, *min-max-ex*, and *min-max-cex*, respectively. Out of these cases, there were 12, 19, and 11 cases which resulted in superlinear speedups. Additionally, for the cases in which the sequential and parallel execution provided the same solution, there were 13, 14, and 5 cases, in which we observed superlinear speedups.

In the second experiment we used the MULTIVOPARINDUCTION algorithm to observe how the use of multiple ordering schemes affects the execution times. In particular, we compared the run times of the modified algorithm with the sequential executions of the basic algorithm. The summary of obtained speedups is shown in Table 2.

Based on the results shown in Table 2, we note that the use of multiple orderings sometimes fails to bring any improvement in the execution time, regardless of the type of sample (see min columns). It is caused by even more frequent synchronization between processes, occurring also within the groups solving the same CSP. However, we observe that the MULTIVOPARINDUCTION algorithm allows also for large superlinear speedups

Table 2. Speedups achieved by the MULTIVOPARINDUCTION algorithm with respect to the basic algorithm

	deg			*min-max-ex*			*min-max-cex*		
Sample source	min	max	avg	min	max	avg	min	max	avg
Tomita	0.1	2.8	0.7	0.2	17.6	2.8	0.2	23.1	3.8
WALTZ-DB	0.1	115.1	13.6	0.9	38101.7	2688.8	0.5	96.8	11.6

(see max columns in WALTZ-DB row). These speedups are observed for the ordering schemes different than the one that found the solution. It is so because, the “best” ordering can produce the solution in much shorter time than the other orderings, bypassing also their problems in solving certain CSPs.

We noted that in case of the WALTZ-DB samples, the *deg* ordering scheme was usually the one that allowed to find the solution in the shortest time (for 30 out of 50 samples). The same trend was also preserved for Tomita languages, for which the *deg* ordering scheme was the fastest in 7 out of 10 cases.

5. Conclusions

We analyzed the speedups obtained by the basic and modified parallel algorithms for NFA induction. For the Tomita languages, defined over two-symbol alphabet, we usually observed negative anomalies, i.e., the algorithms slowed down with the increase of the number of processes. Furthermore, these samples turned out to be easy enough to be solved efficiently even by a single process. For the peptide-based samples of WALTZ-DB database, the parallelism was expolited to a larger extent. Firstly, there were 3 cases in which the sequential execution of the algorithm failed to find the solution within the time limit of 8 hours. And secondly, we observed superlinear speedups of up to 8500 for no more than 7 processes, and over 38 000, for up to 21 processes.

To explain the differences between the two kinds of samples, let us note that the solution space is given by $2^{k^2 l}$, where k is the number of states and l is the alphabet size. Therefore, for the Tomita languages we need to consider at most $2^{32} \approx 4 \cdot 10^9$ different assignments of values to y variables. For the WALTZ-DB samples, we get $2^{180} \approx 10^{18}$ possible assignments. Therefore, the bigger solution space allows for better use of the parallelism and increases also the probability that more than one solution exists. Hence, it allows to achieve superlinear speedups in the cases discussed in Examples 2 and 3.

In the future, we plan to investigate the performance of the algorithms in the cases in which more than one solution is sought. In particular, we are interested in analyzing how the number and types of used variable ordering schemes would affect the speedups. Moreover, we plan to investigate deeper the reasons for the observed slowdowns.

Acknowledgments

The research was supported by National Science Centre Poland, project registration no. 2016/21/B/ST6/02158. The computations were carried out using the computer cluster Tryton at the Academic Computer Center in Gdańsk.

References

[1] C. de la Higuera. A bibliographical study of grammatical inference. *Pattern Recognition*, 38(9):1332–1348, 2005.

[2] W. Wieczorek and O. Unold. Induction of directed acyclic word graph in a bioinformatics task. In A. Clark, M. Kanazawa, and R. Yoshinaka, editors, *Proceedings of the 12th International Conference on Grammatical Inference*, volume 34 of *JMLR Workshop and Conference Proceedings*, pages 207–217. Proceedings of Machine Learning Research, 2014.

[3] W. Wieczorek and O. Unold. Use of a novel grammatical inference approach in classification of amyloidogenic hexapeptides. *Computational and Mathematical Methods in Medicine*, 2016:1–10, 2016.

[4] T. Jastrząb. A comparison of selected variable ordering methods for NFA induction. In J.M.F. Rodrigues et al., editors, *Computational Science – ICCS 2019*, volume 11540 of *LNCS*, pages 741–748. Springer, Cham, 2019.

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

[6] C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27:125–138, 1997.

[7] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[8] S. Ristov, R. Prodan, M. Gusev, and K. Skala. Superlinear speedup in HPC systems: why and when? In *Proceedings of FEDCSIS*, pages 889–898. IEEE, 2016.

[9] M. Tomita. Dynamic construction of finite automata from examples using hill-climbing. In *Proceedings of the 4th Annual Conference of the Cognitive Science Society*, pages 105–108. University of Michigan, USA, 1982.

[10] J. Beerten, J. Van Durme, R. Gallardo, E. Capriotti, L. Serpell, F. Rousseau, and J. Schymkowitz. WALTZ-DB: a benchmark database of amyloidogenic hexapeptides. *Bioinformatics*, 31(10):1698–1700, 2015.

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

[12] G.I. Alvarez, J. Ruiz, A. Cano, and P. Garcia. Nondeterministic regular positive negative inference NRPNI. In J.F. Diaz, C. Rueda, and A. Buss, editors, *Proceedings of the XXXI Latin American Informatics Conference*, pages 239–249. 2005.

[13] F. Coste and D. Fredouille. Unambiguous automata inference by means of state merging methods. In N. Lavrac et al., editors, *Proceedings of the 14th European Conference on Machine Learning*, volume 2837 of *LNAI*, pages 60–71. Springer-Verlag, Berlin, Heidelberg, 2003.

[14] P. Garcia, M. Vazquez de Parga, G.I. Alvarez, and J. Ruiz. Universal automata and NFA learning. *Theoretical Computer Science*, 407(1–3):192–202, 2008.

[15] P. Garcia, M. Vazquez de Parga, G.I. Alvarez, and J. Ruiz. Learning regular languages using nondeterministic finite automata. In O.H. Ibarra and B. Ravikumar, editors, *Proceedings of the 13th International Conference on Implementation and Application of Automata*, volume 5148 of *LNCS*, pages 92–101. Springer-Verlag, Berlin, Heidelberg, 2008.

[16] M. Vazquez de Parga, P. Garcia, and J. Ruiz. A family of algorithms for non deterministic regular languages inference. In O.H. Ibarra and H.-C. Yen, editors, *Proceedings of the 11th International Conference on Implementation and Application of Automata*, volume 4094 of *LNCS*, pages 265–274. Springer-Verlag, Berlin, Heidelberg, 2006.

[17] W. Wieczorek. Induction of non-deterministic finite automata on supercomputers. In J. Heinz, C. de la Higuera, and T. Oates, editors, *Proceedings of the 11th International Conference on Grammatical Inference*, volume 21 of *JMLR Workshop and Conference Proceedings*, pages 237–242. Proceedings of Machine Learning Research, 2012.

[18] T. Jastrząb, Z.J. Czech, and W. Wieczorek. Parallel induction of nondeterministic finite automata. In R. Wyrzykowski et al., editors, *Proceedings of the 11th International Conference on Parallel Processing and Applied Mathematics*, volume 9573 of *LNCS*, pages 248–257. Springer, Cham, 2016.

[19] T. Jastrząb. On parallel induction of nondeterministic finite automata. In I. Altintas et al., editors, *Proceedings of the International Conference on Computational Science*, volume 80 of *Procedia Computer Science*, pages 257–268. Elsevier, 2016.

[20] T. Jastrząb. Performance evaluation of selected variable ordering methods for NFA induction. In *Proceedings of the 14th International Conference on Grammatical Inference – Extended Abstracts*. 2018.

[21] T. Jastrząb. Two parallelization schemes for the induction of nondeterministic finite automata on PCs. In R. Wyrzykowski et al., editors, *Proceedings of the International Conference on Parallel Processing and Applied Mathematics*, volume 10777 of *LNCS*, pages 279–289. Springer, Cham, 2017.

[22] R. Dechter and I. Meiri. Experimental Evaluation of Preprocessing Techniques in Constraint Satisfaction Problems. In *Proc. of IJCAI'89*, pages 271–277, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.