On the performance of automata minimization algorithms

Marco Almeida Nelma Moreira Rogério Reis
{mfa,nam,rvr}@ncc.up.pt
DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

Abstract. Apart from the theoretical worst-case running time analysis not much is known about the average-case analysis or practical performance of finite automata minimization algorithms. On this paper we compare the running time of four minimization algorithms based on experimental results. We applied these algorithms to both deterministic and non-deterministic random automata.

Keywords deterministic finite automata, non-deterministic finite automata, minimal automata, minimization algorithms, random generation

1 Introduction

The problem of writing efficient algorithms to find the minimal DFA equivalent to a given automaton can be traced back to the 1950's with the works of Huffman [Huf55] and Moore [Moo58]. Over the years several alternative algorithms were proposed. Authors typically present the running time worst-case analysis of their algorithms, but the practical experience is sometimes different. The comparison of algorithms performance is always a difficult problem, and little is known about the practical running time performance of automata mininimization algorithms. In particular, there are no studies of average-case analysis of these algorithms, an exception being the work of Nicaud [Nic00], where it is proved that the average-case complexity of Brzozowski's algorithm is exponential for group automata. Lhoták [Lho00] presents a general data structure for DFA minimization algorithms to run in $O(kn \log n)$. Bruce Watson [Wat95] presents some experimental results but his data sets were small and biased. Tabakov and Vardi [VT05] compared Hopcroft's and Brzozowski's algorithms. Baclet *et al.* [BP06] analysed different implementations of the Hopcroft's algorithm. More recently, Bassino *et al.* [BDN07] compared Moore's and Hopcroft's algorithms.

Using the `Python` programming language, we implemented the algorithms due to Hopcroft (**H**) [Hop71], Brzozowski (**B**) [Brz63a], Watson (**W**) [Wat01], and also using full memoization (**WD**) [WD03]. The choice of the algorithms was due to the disparate worst-case complexities and doubts about the practical behaviour of each algorithm.

The input data was obtained with random automata generators. For the (initially-connected) deterministic finite automata we used a uniform random generator and thus our results are statistically accurate. Lacking an equivalent uniform random generator for NFAs, we implemented a non-uniform one. Although not statistically significant, the results in this case are still fairly interesting.

The text is organised as follows. In Section 2 we present some definitions and the notation used throughout the paper. In Section 3 we describe each of the algorithms, briefly explain them, and present the respective pseudo-code. In Section 4 we describe the generation methods of the random automata. In Section 5 we present the experimental results and in Section 6 we expose our final remarks and possible future work.

2 Preliminaries

A *deterministic finite automaton* (DFA) $\mathcal{D}$ is a tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is a finite set of *states*, Σ is the *input alphabet* (any non-empty set of symbols), $\delta : Q \times \Sigma \to Q$ is the *transition* *function*, q_0 is the *initial state* and $F \subseteq Q$ is the set of *final states*. When the transition function is total, the automaton $\mathcal{D}$ is said to be *complete*. Any finite sequence of alphabet symbols $a \in \Sigma$ is a *word*. Let $\Sigma^\star$ denote the set of all words over the alphabet Σ and ϵ denote the *empty word*. We define the *extended transition function* $\hat{\delta} : Q \times \Sigma^\star \to Q$ in the following way: $\hat{\delta}(q, \epsilon) = q$; $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$. A state $q \in Q$ of a DFA $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ is called *accessible* if $\hat{\delta}(q_0, w) = q$ for some $w \in \Sigma^\star$. If all states in Q are accessible, a complete DFA $\mathcal{D}$ is called (complete) *initially-connected* (ICDFA). The *language* accepted by $\mathcal{D}$, $L(\mathcal{D})$, is the set of all words $w \in \Sigma^\star$ such that $\hat{\delta}(q_0, w) \in F$. Two DFAs $\mathcal{D}$ and $\mathcal{D}'$ are *equivalent* if and only if $L(\mathcal{D}) = L(\mathcal{D}')$. A DFA is called *minimal* if there is no other equivalent DFA with fewer states. Given a DFA $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$, two states $q_1, q_2 \in Q$ are said to be *equivalent*, denoted $q_1 \approx q_2$, if for every $w \in \Sigma^\star$, $\hat{\delta}(q_1, w) \in F \Leftrightarrow \hat{\delta}(q_2, w) \in F$. Two states that are not equivalent are called *distinguishable*. The equivalent minimal automaton $\mathcal{D}/\approx$ is called the *quotient automaton*, and its states correspond to the equivalence classes of $\approx$. It is proved to be unique up to isomorphism.

A *non-deterministic finite automaton* (NFA) is also a tuple $(Q, \Sigma, \Delta, I, F)$, where I is a *set of initial states* and the *transition function* is defined as $\Delta : Q \times \Sigma \to 2^Q$. Just like with DFAs, we can define the *extended transition function* $\hat{\Delta} : 2^Q \times \Sigma^\star \to 2^Q$ in the following way: $\hat{\Delta}(S, \epsilon) = S$; $\hat{\Delta}(S, xa) = \bigcup_{q \in \hat{\Delta}(S,x)} \delta(q, a)$. The *language* accepted by $\mathcal{N}$ is the set of all words $w \in \Sigma^\star$ such that $\hat{\Delta}(I, w) \cap F \neq \emptyset$. Every language accepted by some NFA can also be described by a DFA. The *subset construction* method takes a NFA $\mathcal{A}$ as input and computes a DFA $\mathcal{D}$ such that $L(\mathcal{A}) = L(\mathcal{D})$. This process is also referred to as *determinization* and has a worst-case running time complexity of $O\left(2^{|Q|}\right)$.

Following Leslie [Les95], we define the *transition density* of an automaton $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ as the ratio $\frac{t}{|Q|^2|\Sigma|}$, where t is the number of transitions in $\mathcal{A}$. This density function is normalised, giving always a value between 0 and 1. We also define *deterministic density* as the ratio of the number of transitions t to the number of transitions of a complete DFA with the same number of states and symbols, i.e., $\frac{t}{|Q||\Sigma|}$.

The *reversal* of a word $w = a_0 a_1 \cdots a_n$, written w^R, is $a_n \cdots a_1 a_0$. The reversal of a language $L \subseteq \Sigma^\star$ is $L^R = \{w^R \mid w \in \mathrm{L}\}$. Further details on regular languages can be found in the usual references (Hopcroft [HMU00] or Kozen [Koz97], for example).

3 Minimization algorithms

Given an automaton, to obtain the associated minimal DFA we must compute the equivalence relation $\approx$ as defined in Section 2. The computation of this relation is the key difference of the several minimization algorithms. Moore's algorithm and its variants, for example, aim to find pairs of distinguishable states. **H**, on the other hand, computes the minimal automaton by refining a partition of the states' set.

Of the three minimization algorithms we compared, **H** has the best worst-case running time analysis. **B** is simple and elegant, and, despite its exponential worst-case complexity, it is supposed to frequently outperform other algorithms (including **H**). **B** also has the particularity of being able to take both DFAs and NFAs as input. **W** can be halted at any time yielding a partially minimized automaton. The improved version, **WD**, includes the use of full memoization. Because one of our motivations was to check the minimality of a given automaton, not to obtain the equivalent minimal one, this algorithm was of particular interest.

3.1 Hopcroft's algorithm (H)

H [Hop71], published in 1971, achieves the best known running time worst-case complexity for minimization algorithms. It runs on $O(kn \log n)$ time for a DFA with n states and an alphabet of size k. Let $\mathcal{D} = (Q, \Sigma, \delta, q_0, F)$ be a DFA. **H**, proceeds by refining the coarsest partition until no more refinements are possible. The initial partition is $P = \{F, Q - F\}$ and, at each step of the algorithm, a block $B \in P$ and a symbol $a \in \Sigma$ are selected to *refine* the partition. This refinement process *splits* each block B' of the partition according to whether the states of B', when consuming the symbol a, go to a state which is in B or not. Formally, we call this procedure `split` and define it by

$$\mathtt{split}(\mathtt{B'},\mathtt{B},\mathtt{a}) = (\mathtt{B'} \cap \check{\delta}^{-1}(\mathtt{B},\mathtt{a}), \mathtt{B'} \cap \overline{\check{\delta}^{-1}(\mathtt{B},\mathtt{a})})$$

where $\check{\delta}(S,a) = \bigcup_{q \in S} \delta(q,a)$.

The algorithm terminates when there are no more blocks to refine. In the end, each block of the partition is a set of equivalent states. Because, for any two blocks $B, B' \in P$, every state $q \in B$ is distinguishable from any state $q' \in B'$, the elements of P represent the states of a new minimal DFA. The complete pseudo-code is presented in Algorithm 1.1.

```
def hopcroft():
    L = {}
    if |F| < |Q-F|:
        P = {Q-F, F}; L = {F}
    else:
        P = {F, Q-F}; L = {Q-F}
    while L $\neq \emptyset$:
        S = extract(L)
        for a in $\Sigma$:
            for B in P:
                ($B_1$, $B_2$) = split(B, S, a)
                P = P - {B}; P = P $\cup$ {$B_1$}; P = P $\cup$ {$B_2$}
                if |$B_1$| < |$B_2$|:
                    L = L $\cup$ {$B_1$}
                else:
                    L = L $\cup$ {$B_2$}
    return P
```

Algorithm 1.1. Hopcroft's algorithm (**H**).

The set L contains blocks of P not yet treated. The `extract` procedure removes one element of L to be used in the splitting process. The choice of the element does not influence the correctness of the algorithm.

3.2 Brzozowski's algorithm (B)

B [Brz63b] is based on two successive reverse and determinization operations and the full pseudo-code is presented (in one single line!) on Algorithm 1.2.

```
def brzozowski(fa):
    return det(rev(det(rev(fa))))
```

Algorithm 1.2. Brzozowski's algorithm (**B**).

Having to perform two determinizations, the worst-case running time complexity of **B** is exponential. Watson's thesis, however, presents some surprising results about **B** practical performance, usually outperforming **H**. As for the peculiar way that this algorithm computes a minimal DFA, Watson assumed it to be unique and, in his taxonomy, placed it apart all other algorithms. Later, Champarnaud et al. [CKP02] analysed the way the sequential determinizations perform the minimization and showed that it does compute state equivalences.

3.3 An incremental algorithm (W)

In 2001 Watson presented an incremental DFA minimization algorithm (**W**) [Wat01]. This algorithm can be halted at any time yielding a partially minimized DFA that recognises the same language as the input DFA. Later, Watson and Daciuk presented an improved version of the same algorithm (**WD**) [WD03] which makes use of full memoization. While the first algorithm has a worst-case exponential running time, the memoized version yields a $O(n^2)$ algorithm (for all *practical* values of n, i.e., $n \leq 2^{2^{16}}$). It was not clear, however, that this algorithm would outperform **H** as the experimental results in [WD03] seemed to point to. Since the use of memoization introduces some considerable overhead in the algorithm, we wanted to discover at what point this extra work begins to payoff.

W uses an auxiliary function, `equiv`, to test the equivalence of two states. The third argument, an integer k, is used to control the recursion depth only for matters of efficiency. Also for matters of efficiency, a variable S that contains a set of presumably equivalent pairs of states is made global. The pseudo-code for a non-memoized, specialized for ICDFAs, implementation of `equiv` is presented in Algorithm 1.3. The memoized algorithm (**WD**) is quite extensive and can be found in Watson and Daciuk's paper [WD03].

```
def equiv(p, q, k):
    if k = 0:
        return (p in F and q in F) or (not p in F and not q in F)
    elif (p,q) in S:
        return True
    else:
        eq = (p in F and q in F) or (not p in F and not q in F)
        S = S ∪ {(p,q)}
        for a in $\Sigma$:
            if not eq:
                return False
            eq = eq and equiv($\delta(p,a)$, $\delta(q,a)$, k−1)
        S = S − {(p,q)}
    return eq
```

Algorithm 1.3. Pairwise state equivalence algorithm.

Having a method to verify pairwise state equivalence, it is possible to implement a test that calls `equiv` for every pair of states and returns $False$ if some pair is found to be equivalent.

4 Random automata generation

Even if we consider only (non-isomorphic) ICDFAs, the number of automata with n states over an alphabet of k symbols grows so fast [RMA05a] that trying to minimize every one is not feasible, even for small values of n and k. The same applies to NFAs. In order to compare the practical performance of the minimization algorithms, we must have an arbitrary quantity of "good" randomly generated automata available, i.e. the samples can not be unbiased.

A uniform random generator obviously produces unbiased samples. We used the DFA string representation and random generation method proposed by Almeida et al. [AMR07]. This approach, unlike the one proposed by Bassino et al. [BN07], does not require a rejection step. The generator produces a string of the form

$$\underbrace{[[a_1 \cdots a_k] \cdots [b_1 \cdots b_k]]}_{n},$$

where $a_i, b_i \in [0, n-1]$. This string is a canonical representation of an ICDFA with n states and k symbols without final states information, as described by Reis et al. [RMA05b,AMR07]. Given an order over Σ, it is possible to define a canonical order over the set of states by traversing the automaton in a breadth-first way choosing at each node the outgoing edges using the order of Σ. In the string representation, each of the i blocks, for $1 \leq i \leq n$, corresponds to the transitions from the state $i-1$. The following string, for example, represents the ICDFA from Fig. 1,

$$[[1, 2], [0, 1], [1, 0]].$$

The random generator produces the random strings from left to right, taking into account the number of ICDFAs that, at a given point, would still be possible to produce with a given prefix. The set of final states is computed by generating an equiprobable bitstream of size n and considering final all the states that correspond to a non-zero position.

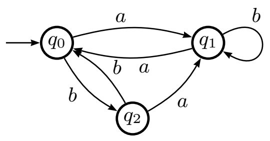

Fig. 1. A ICDFA with 3 states and 2 symbols, but no final states.

Lacking a uniform random generator for NFAs, we implemented one which combines the van Zijl bit-stream method, as presented by Champarnaud et al. [CHPZ04], with one of Leslie's approaches [Les95], which allows us both to generate initially connected NFAs (with one initial state) and to control the transition density. Leslie presents a "generate-and-test" method which may never stop, so we added some minor changes that correct this situation. A brief explanation of the random NFA generator follows. Suppose we want to generate a random NFA with n states over an alphabet of k symbols and a transition density d. Let the states (respectively the symbols) be named by the integers $0, \ldots, n-1$ (respectively $0, \ldots, k-1$). A sequence of n^2k bits describes the transition function in the following way: the occurrence of a non-zero bit at the position $ink + jk + a$ denotes the existence of a transition from state i to state j labelled by the symbol a. Consider the following bitstream, which represents the NFA of the Fig. 2.

$$\underbrace{\overbrace{0\,0}^{q_0} \mid \overbrace{1\,1}^{q_1} \mid \overbrace{1\,0}^{q_2}}_{q_0} \parallel \underbrace{\overbrace{0\,0}^{q_0} \mid \overbrace{0\,0}^{q_1} \mid \overbrace{0\,0}^{q_2}}_{q_1} \parallel \underbrace{\overbrace{0\,1}^{q_0} \mid \overbrace{1\,0}^{q_1} \mid \overbrace{0\,1}^{q_2}}_{q_2}$$

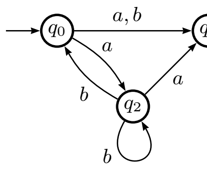

Fig. 2. The NFA built from the bitstream above.

Starting with a sequence of zero bits, the first step of the algorithm is to create a connected structure and thus ensure that all the states of the final NFA will be accessible. In order to do so, we define the first state as 0, mark it as visited, generate a transition from 0 to any not-visited state i, and mark i as visited. Next, until all states are marked as visited, randomly choose an already visited state q_1, randomly choose a not-visited state q_2, add a transition from q_1 to q_2 (with a random), and mark q_2 as visited. At this point we have an initially connected NFA and proceed by adding random transitions. Until the desired density is achieved, we simply select one of the bitstream's zero bits and set it to one. By maintaining a list of visited states on the first step and keeping record of the zero bits on the second step, we avoid generating either a disconnected NFA or a repeated transition and guarantee that the algorithm always halts. The set of final states can be easily obtained by generating an equiprobable bitstream of size n and considering final all the states that correspond to a non-zero position in the bitstream.

5 Experimental results

To compare algorithms is always a difficult problem. The choice of the programming language, implementation details, and the hardware used may harm the rigour of any benchmark. In order to produce realistic results, the input data should be random so that it represents a typical usage of the algorithm and the test environment should be identical for all benchmarks. We implemented all the algorithms in the `Python 2.4` programming language, using similar data structures whenever possible. All the tests were executed in the same computer, an Intel® Xeon® 5140 at 2.33GHz with 2GB of RAM. We used samples of 10.000 automata, with $5 \leq n \leq 100$ states and alphabets with $k \in \{2, 5, 10, 20\}$ symbols. For the data sets obtained with the uniform random generator, the size of each sample is sufficient to ensure a 95% confidence level within a 1% error margin. It is calculated with the formula $n = (\frac{z}{2\epsilon})^2$, where z is obtained from the normal distribution table such that $P(-z < Z < z)) = \gamma$, ϵ is the error margin, and γ is the desired confidence level.

5.1 Random ICDFA minimization

On his thesis, Watson used a fairly biased sample. It consisted of 4833 DFAs of which only 7 had 23 states. As Watson himself states, being constructed from regular expressions, the automata "... are usually not very large, they have relatively sparse transition graphs, and the alphabet frequently consists of the entire ASCII character set.". On their paper on the incremental minimization algorithm [WD03], Watson and Daciuk also present some performance comparisons of automata minimization algorithms. They used four different data sets, one from experiments on finite-state approximation of context-free grammars and three that were automatically generated. These are not, however, uniform random samples, and thus, do not represent a typical usage of the algorithms.

The following graphics show the running times for the three algorithms while minimizing a sample of 10.000 random ICDFAs. The running time limit for all algorithms was set to 24 hours.

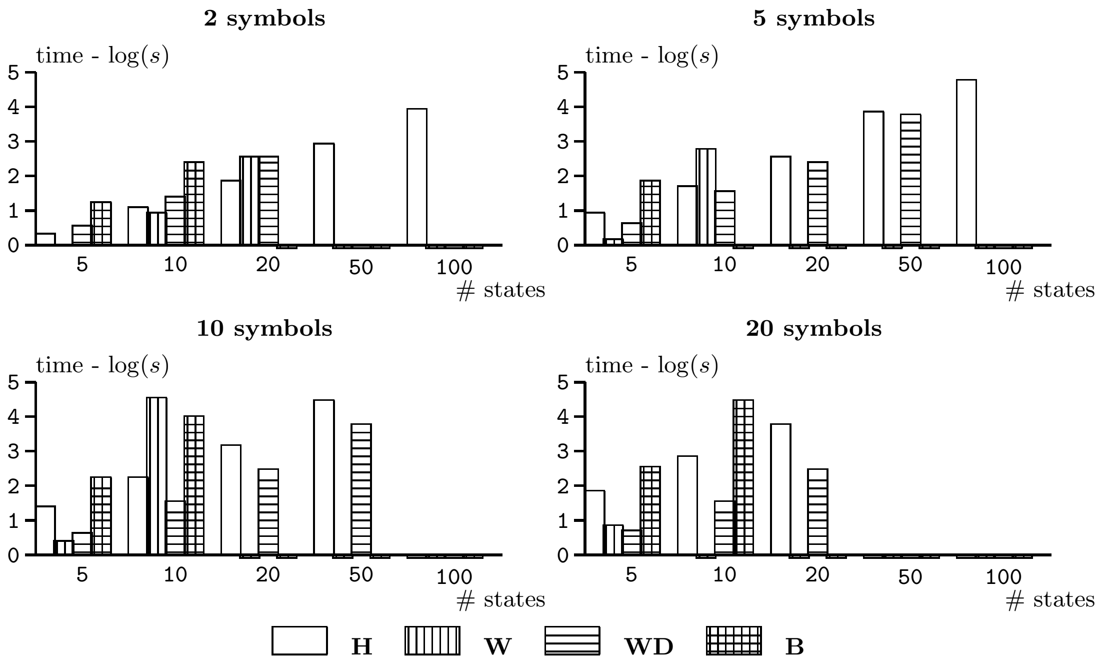

Fig. 3. Running time results for the minimization of 10.000 ICDFAs with $k \in \{2, 5, 10, 20\}$.

For small alphabets ($k = 2$), **H** is always the fastest. When the alphabet size grows ($k \geq 5$), however, **H** is clearly outperformed by the **WD**, which was over twice as fast as **H** when minimizing

ICDFAs with an alphabet of size $k \geq 10$. **W** showed itself quite slow in all tests. It is important to point out that for $k \geq 5$ all the automata were already minimal and so the speed of the incremental algorithm can not be justified by the possibility of halting whenever two equivalent states are found. The fact that almost all ICDFAs are minimal was observed by several authors, namely Almeida *et al.* [AMR06] and Bassino *et al.*[BDN07]. As Watson himself stated, the incremental algorithm may show exponential performance for some DFAs. This was the case in one of our tests. For the sample of 5 symbols and 15 states **WD** took an unusual amount of time. **B** is never the fastest algorithm. In fact, even for small alphabets it was not possible to use it on ICDFAs with more than 15 states.

5.2 Random NFAs minimization

The next set of graphics shows the execution times of the three algorithms when applied to a set of 10.000 random NFAs. The running time limit for all algorithms was set to 15 hours. It is important to note that the NFA generator we used is not a uniform one, and so we can not prove that each sample is actually a good representative of the universe. Because we are dealing with NFAs, the transition density is an important factor and so each sample was generated with three different transition densities (d): 0.2, 0.5, and 0.8. For both the **WD** and **H**, which are only able to minimize DFAs, we also accounted for the time spent in the subset construction method.

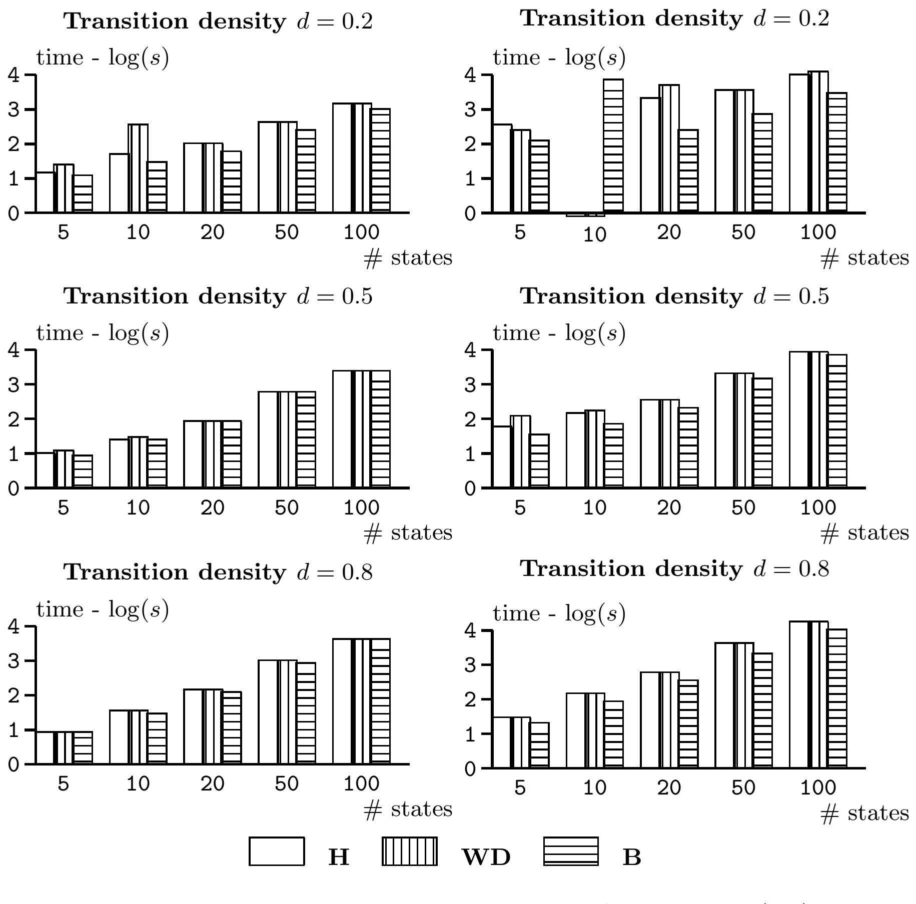

Fig. 4. Running time results for the minimization of 10.000 ICDFAs with $k = 2$ (left) and $k = 5$ (right).

For alphabets with two symbols there are no significant differences in any of the algorithms' general performance, although **B** is usually the fastest. **H** outperforms **B** for less than 4% only when $d = 0.5$ and $n \in \{50, 100\}$.

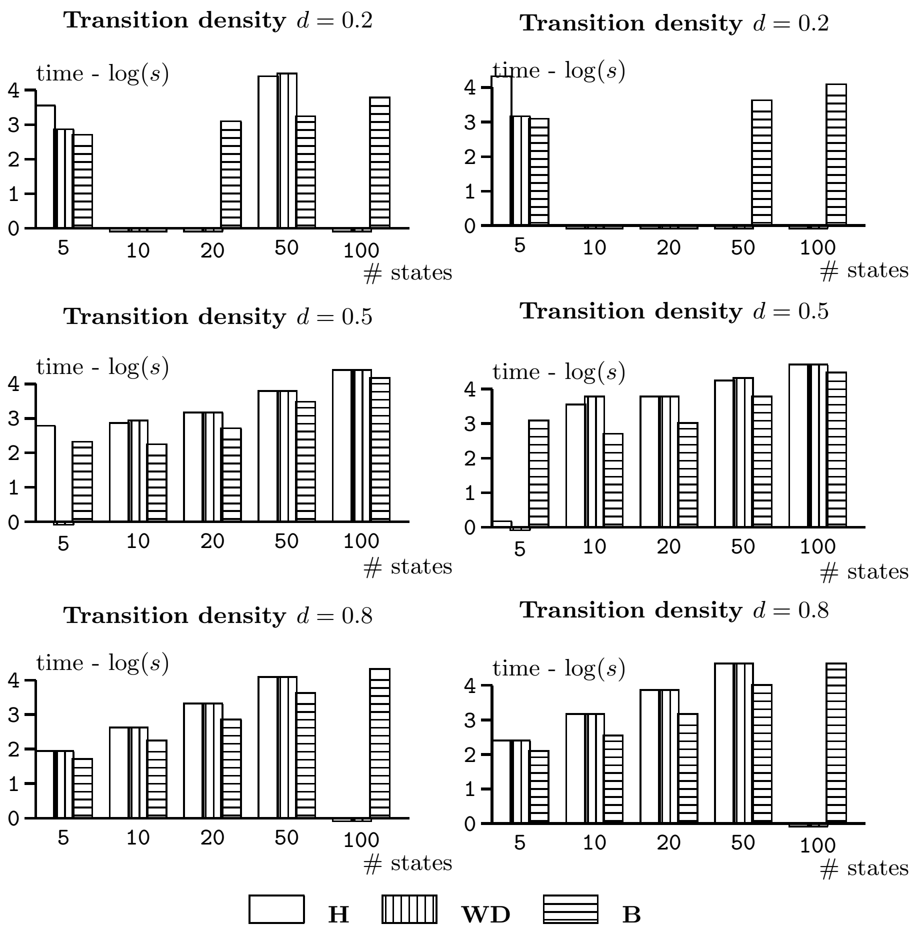

Fig. 5. Running time results for the minimization of 10.000 NFAs with $k = 10$ (left) and $k = 20$ (right).

For alphabets with $k = 5$, **B** is always the fastest and, except for occasional cases, **H** is slightly faster than the incremental algorithm.

When the alphabet size increases, **B**'s performance becomes quite remarkable. For an alphabet with size $k \in \{10, 20\}$, **B** is definitively the fastest, being the only algorithm to actually finish the minimization process of almost all random samples within the 15 hour limit. As for **H** and **WD**, except for two cases, there are no significant performance differences.

For $d = 0.2$ and $n = 10$ all the algorithms showed a particularly bad performance. For $k = 5$ only **B** finished the minimization process (taking an unusual high amount of time) and for $k \in \{10, 20\}$ none of the algorithms completed the minimization process within the 15 hour time limit. This result corroborates Leslie's conjecture [Les95] about the number of states obtained with the subset construction method for a given deterministic density d_d. Leslie's conjecture states that randomly generated automata exhibit the maximum execution time and the maximum number of states at an approximate deterministic density of 2.0. While generating the random NFAs, we considered the transition density $d = \frac{t}{n^2k}$, which is related to the deterministic density $d_d = \frac{t}{nk}$ by $d_d = nd$. It is easy to see that in our case $d_d = nd = 10 \times 0.2 = 2.0$, which will make the subset construction computationally expensive. In order to achieve the same exponential behaviour in the subset method for $d \in \{0.5, 0.8\}$ the number of states would have to be $n \in \{4, 2.5\}$, but for such a small number of states the exponential blowup is not meaningful. This explains why there are no similar results for the test batches with $d \in \{0.5, 0.8\}$. Considering we used a variation of one of Leslie's random NFA generators, this result does not come with any surprise.

6 Analisys of the results

In this work, we experimentally compare four automata minimization algorithms (**H**, **W**, **WD** and **B**). As data sets we use two different types of randomly generated automata (ICDFAs and NFAs) with a range of different number of states and symbols. The ICDFAs' data set was obtained using a uniform random generator and is large enough to ensure a 95% confidence level within a 1% error margin. For ICDFAs with only to symbols, **H** is the fastest algorithm. As the alphabet size grows, **WD** begins to perform better than **H**. With alphabets larger than 10, **WD** becomes clearly faster. As for **W** and **B** algorithms, we can safely conclude that neither performs well, regardless of the number of states and symbols. As for the NFAs, it is important to note that the random generator used was not a uniform one, and thus does not have the same statistical accuracy as the first one. **B** is definitively the fastest algorithm. Both **H** and **WD** consistently show equally slower results.

All this algorithms make use of the subset construction pass, at least once, which turns the reason for **B**'s good performance even less evident. It would be interesting to make an average-case running time complexity analysis for the DFA reversal, and thus possibly explain **B**'s behaviour with ICDFAs minimization.

7 Comparison with Related Work

In this final section we summarise and compare our experiments with some of the results of the works cited before.

Bruce Watson implemented five minimization algorithms: two versions of Moore's algorithm as well as **H**, **W** and **B**. As we have mentioned before, the data set then used was fairly biased and his results lead him to conclude that the two Moore based algorithms perform very poorly and that **B** normally outperforms the other two.

Baclet *et al.* [BP06] implemented **H** with two different queue strategies: LIFO and FIFO. The implementations were tested with several random (non-uniform) automata having thousands of states but very small alphabets ($k \leq 4$), concluding that the LIFO strategy is better, at least for alphabets of one or two symbols.

Bassino et al. [BDN07] used an ICDFAs' uniform random generator based on a Boltzmann sampler to create a data set and compared the performance of the **H** (with the two strategies refereed above) and Moore's algorithms. The automata generated have up to some thousands of states for alphabets of size 2. Their results are statistically accurate, and indicate that Moore's algorithm is, for the average case, more efficient than **H**. Moreover, no clear difference were found between the two queue strategies for **H**. These results are interesting because they neither corroborate the results of the works mentioned above nor the general idea that **H** outperforms Moore's algorithm in practice. It remains unstudied a comparison between Moore's algorithm with **WD** using a uniformly generated data set.

Finally, Tabakov and Vardi [VT05] studied **H** and **B** performance with a data set of random NFAs. The random model they used is similar to the one we describe in Section 4 but considering a deterministic density, d_d. They choose $0 \leq d_d \leq 2.5$ and the samples were relatively small: 200 automata with $n < 50$ and $k = 2$. The conclusion was that **H** is faster for low-density automata ($d_d < 1.5$), while **B** is better for high-density automata. For the example studied of $n = 30$ the deterministic density $d_d < 1.5$, corresponds to a normalised transition density that we used, $d < 0.05$. This phase transition may be due to the fact that for such a low normalised transition density the probability of a connected NFA being deterministic is very high.

References

[AMR06] M. Almeida, N. Moreira, and R. Reis. Aspects of enumeration and generation with a string automata representation. In H. Leung and G.Pighizzini, editors, *Proc. of DCFS'06*, pages 58–69, Las Cruces, New Mexico, 2006. NMSU.

[AMR07] M. Almeida, N. Moreira, and R. Reis. Enumeration and generation with a string automata representation. *Theoretical Computer Science*, 387(2):93–102, 2007.

[BDN07] Frédérique Bassino, Julien David, and Cyril Nicaud. A library to randomly and exhaustively generate automata. In *Implementation and Application of Automata*, volume 4783 of *LNCS*, pages 303–305. Springer-Verlag, 2007.

[BN07] F. Bassino and C. Nicaud. Enumeration and random generation of accessible automata. *Theoretical Computer Science*, 381(1-3):86–104, 2007.

[BP06] M. Baclet and C. Pagetti. Around hopcroft's algorithm. pages 114–125, Taipei, Taiwan, 2006. Springer.

[Brz63a] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In J. Fox, editor, *Proceedings of the Symposium on Mathematical Theory of Automata*, volume 12 of *MRI Symposia Series*, pages 529–561, New York, NY, April 24-26,1962 1963. Polytechnic Press of the Polytechnic Institute of Brooklyn, Brooklyn, NY.

[Brz63b] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In J. Fox, editor, *Proc. of the Sym. on Mathematical Theory of Automata*, volume 12 of *MRI Symposia Series*, pages 529–561, NY, 1963. Polytechnic Press of the Polytechnic Institute of Brooklyn.

[CHPZ04] J.-M. Champarnaud, G. Hansel, T. Paranthoën, and D. Ziadi. Random generation models for nfas. *J. of Automata, Languages and Combinatorics*, 9(2), 2004.

[CKP02] J.-M. Champarnaud, A. Khorsi, and T. Paranthoën. Split and join for minimizing: Brzozowski's algorithm. In M. Balík and M. Simánek, editors, *Proc. of PSC'02*, Report DC-2002-03, pages 96–104. Czech Technical University of Prague, 2002.

[HMU00] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 2000.

[Hop71] J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Proc. Inter. Symp. on the Theory of Machines and Computations*, pages 189–196, Haifa, Israel, 1971. AP.

[Huf55] D. A. Huffman. The synthesis of sequential switching circuits. *The Journal of Symbolic Logic*, 20(1):69–70, 1955.

[Koz97] D. C. Kozen. *Automata and Computability.* Undergrad. Texts in Computer Science. Springer-Verlag, 1997.

[Les95] T. Leslie. Efficient approaches to subset construction. Master's thesis, University of Waterloo, Ontario, Canada, 1995.

[Lho00] O. Lhoták. A general data structure for efficient minimization of deterministic finite automata. Technical report, University of Waterloo, 2000.

[Moo58] E. F. Moore. Gedanken-experiments on sequential machines. *The Journal of Symbolic Logic*, 23(1):60, 1958.

[Nic00] C. Nicaud. *Étude du comportement en moyenne des automates finis et des langages rationnels.* PhD thesis, Université de Paris 7, 2000.

[RMA05a] R. Reis, N. Moreira, and M. Almeida. On the representation of finite automata. In C. Mereghetti C. Mereghetti, B. Palano, G. Pighizzini, and D.Wotschke, editors, *Proc. of DCFS'05*, pages 269–276, Como, Italy, 2005.

[RMA05b] R. Reis, N. Moreira, and M. Almeida. On the representation of finite automata. In C. Mereghetti, B. Palano, G. Pighizzini, and D.Wotschke, editors, *7th International Workshop on Descriptional Complexity of Formal Systems*, number 06-05 in Rapporto Tecnico delo Departimento de Informatica e Comunicazione dela Università degli Studi di Milano, pages 269–276, Como, Italy, June 2005. International Federation for Information Processing.

[VT05] M. Vardi and D. Tabakov. Evaluating classical automata-theoretic algorithms. In *LPAR'05*, 2005.

[Wat95] B. W. Watson. *Taxonomies and toolkit of regular languages algortihms.* PhD thesis, Eindhoven Univ. of Tec., 1995.

[Wat01] B. W. Watson. An incremental DFA minimization algorithm. In *International Workshop on Finite-State Methods in Natural Language Processing*, Helsinki, Finland, August 2001.

[WD03] B. W. Watson and J. Daciuk. An efficient DFA minimization algorithm. *Natural Language Engineering*, pages 49–64, 2003.