DFA Minimization in Map-Reduce

Gösta Grahne
Concordia University
Montreal, Canada, H3G 1M8
grahne@cs.concordia.ca

Shahab Harrafi
Concordia University
Montreal, Canada, H3G 1M8
s_harraf@alumni.concordia.ca

Iraj Hedayati *
Concordia University
Montreal, Canada, H3G 1M8
h_iraj@encs.concordia.ca

Ali Moallemi
Concordia University
Montreal, Canada, H3G 1M8
moa_ali@encs.concordia.ca

ABSTRACT

We describe Map-Reduce implementations of two of the most prominent DFA minimization methods, namely Moore's and Hopcroft's algorithms. Our analysis shows that the one based on Hopcroft's algorithm is more efficient, both in terms of running time and communication cost. This is validated by our extensive experiments on various types of DFA's, with up to 2^{17} states. It also turns out that both algorithms are sensitive to skewed input, the Hopcroft's algorithm being intrinsically so.

CCS Concepts

•Theory of computation → MapReduce algorithms; Formal languages and automata theory;

Keywords

Map-Reduce, Automata, DFA Minimization, Communication Cost

1. INTRODUCTION

Google introduced *Map-Reduce (MR)* [3] as a parallel programming model that can work over large clusters of commodity computers. Map-Reduce provides a high-level framework for designing and implementing such parallelism. MR is by now well established in academia and industry. While many single round (non-recursive) problems have been studied and implemented, recursive problems that require several rounds of Map and Reduce are still being explored. A growing number of papers (see e.g. [1, 10, 13]), deal with such multi-round MR algorithms.

The problem of *DFA minimization* provides an interesting vehicle for studying multi-round MR problems for several reasons. Firstly, because of its importance and wide use in applications. Secondly, because of the multi-round structure of this problem. In contrast with single-round problems such as the NFA intersections studied in [4], multi-round problems require coordination between rounds. Thirdly, although DFA minimization is fairly straightforward to express in Datalog, the results of the present paper show that designing DFA minimization algorithms in MR is far from trivial. Thus, we hope that DFA minimization can shed some light on the more general problem of implementing arbitrary Datalog programs in the MR framework.

We note that DFA minimization has been studied for various parallel architectures (see e.g. [6, 11, 14, 15]). However, all of these architectures contain some kind of shared memory, while the Map-Reduce model corresponds to a *shared-nothing* architecture. To the best of our knowledge, ours is the first paper studying DFA minimization in MR. However, somewhat similar problem of graph reductions based on bi-simulation has recently been the object of MR implementations, see e.g. [8] and [12].

In the serial context, the two major algorithms for DFA minimization are the one of Moore [9] and the one of Hopcroft [5]. Of these, Hopcroft's algorithm is considered superior due to its running time, which is less than that of Moore's algorithm. In this paper we derive MR versions of Moore's and Hopcroft's algorithms. The resulting algorithms are called *Moore-MR* and *Hopcroft-MR*, respectively.

We analyze Moore-MR and Hopcroft-MR in terms of total *communication cost*, i.e. the size of all data communicated by mappers and reducers over the commodity cluster. These turn out to be $O(k^2n^2 \log n)$ and $O(kn^2 \log n)$ bits, respectively. Here n is the number of states of the automaton to be minimized, and k is the size of its alphabet. We have implemented both algorithms in Apache Hadoop, and performed extensive experiments on synthetically generated DFA's, with various graph topologies. These topologies allow us to compare the communication cost and running time of the algorithms with respect to worst and average inputs, as well as to see the effect of skewed input. Our experiments validate the analysis and show that Moore-MR and Hopcroft-MR are comparable in performance, but for larger alphabets Hopcroft-MR is more efficient. The experiments also show that the performance of both algorithms deteriorate when the input is skewed.

The rest of this paper is organized as follows. Section 2 provides the necessary technical definitions, and Section 3 describes Moore's and Hopcroft's algorithms. In Section 4 we adapt the two algorithms for the MR framework along with the technical analysis of the communication cost for the two adapted algorithms. Section 5 describes the experimental results. Conclusions are drawn in the last section.

*Contact author.

BeyondMR'16, June 26-July 01 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2926534.2926537

2. PRELIMINARIES

In this section we introduce the basic technical preliminaries and definitions. A *Finite Automaton (FA)* is a 5-tuple $A = (Q, \Sigma, \delta, q_s, F)$, where Q is a finite set of states, Σ is a finite set of alphabet symbols, $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation, $q_s \in Q$ is the start state, and $F \subseteq Q$ is a set of final states. By Σ^* we denote the set of all finite strings over Σ. Let $w = a_1 a_2 \dots a_\ell$ where $a_i \in \Sigma$, be a string in Σ^*. An *accepting computation path* of w in A is a sequence $(q_s, a_1, q_1), (q_1, a_2, q_2), \dots, (q_{\ell-1}, a_\ell, q_f)$ of tuples of δ, where $q_f \in F$. The *language* accepted by A, denoted $L(A)$, is the set of all strings in Σ^* for which there exists an accepting computation path in A.

An FA A is said to be *deterministic* if for all $p \in Q$ and $a \in \Sigma$ there is a unique $q \in Q$, such that $(p, a, q) \in \delta$. Otherwise, the FA is *non-deterministic*. Deterministic FA's are called DFA's, and non-deterministic ones are called NFA's. By the well known subset construction, any NFA A can be turned into a DFA A^D, such that $L(A^D) = L(A)$. For a DFA $A = (Q, \Sigma, \delta, q_s, F)$, we also write δ in the functional format, i.e. $\delta(p, a) = q$ iff $(p, a, q) \in \delta$. For a state $p \in Q$, and string $w = a_1 a_2 \dots a_\ell \in \Sigma^*$, we denote by $\hat{\delta}(p, w)$ the unique state $\delta(\delta(\cdots \delta(\delta(p, a_1), a_2) \cdots, a_{\ell-1}), a_\ell)$. Then, for a DFA its language $L(A)$ can be defined as $\{w : \hat{\delta}(q_s, w) \in F\}$.

A DFA A is said to be *minimal*, if all DFA's B, such that $L(A) = L(B)$, have at least as many states as A. For each regular language L, there is a unique (up to isomorphism of their graph representations) minimal DFA that accepts L.

The DFA minimization problem has been studied since the 1950's. A taxonomy of DFA minimization algorithms was created by B. W. Watson in 1993 [16]. The taxonomy reveals that most of the algorithms are based on the notion of equivalent states (to be defined). The sole exception is Brzozowski's algorithm [17], which is based on *reversal* and determinization of automata. Let $A = (Q, \Sigma, \delta, q_s, F)$ be a DFA. Then the reversal of A is the NFA $A^R = (Q, \Sigma, \delta^R, F, \{q_s\})$, where $\delta^R = \{(p, a, q) : (q, a, p) \in \delta\}$. Brzozowski showed in 1962 that if A is a DFA, then $(((A^R)^D)^R)^D$ is a minimal DFA for $L(A)$. This rather surprising result does not however yield a practical minimization algorithm since there is a potential double exponential blow-up in the number of states, due to the two determinization steps.

The rest of the algorithms are based on equivalence of states. Let $A = (Q, \Sigma, \delta, q_s, F)$ be a DFA, and $p, q \in Q$. Then p is *equivalent* with q, denoted $p \equiv q$, if for all strings $w \in \Sigma^*$, it holds that $\hat{\delta}(p, w) \in F$ iff $\hat{\delta}(q, w) \in F$. The relation $\equiv$ is an equivalence relation, and by $Q/_\equiv$ we denote the induced partition of the state-space Q. The *quotient* DFA $A/_\equiv = (Q/_\equiv, \Sigma, \gamma, q_s/_\equiv, F/_\equiv)$ where $\gamma(p/_\equiv, a) = \delta(p, a)/_\equiv$, is then a minimal DFA, such that $L(A/_\equiv) = L(A)$. An important observation is that $\equiv$ can be computed iteratively as a sequence $\equiv_0, \equiv_1, \equiv_2, \dots, \equiv_m = \equiv$, where $p \equiv_0 q$ if $p \in F \Leftrightarrow q \in F$, and $p \equiv_{i+1} q$ if $p \equiv_i q$ and $\delta(p, a) \equiv_i \delta(q, a)$, for all $a \in \Sigma$. For each DFA A the sequence converges in at most ℓ steps, where ℓ is the length of the longest simple path from the start state to a final state. This means that $Q/_\equiv$ can be computed iteratively, each step refining $Q/_{\equiv_i}$ to $Q/_{\equiv_{i+1}}$, and eventually converging to $Q/_\equiv$.

We reserve the letter n for the number of states in a DFA and the letter k for the size of its alphabet. We assume that $\Sigma = \{a_1, \dots, a_k\}$, and use a and b as metalinguistic variables denoting arbitrary alphabet symbols.

3. THE CLASSICAL ALGORITHMS

3.1 Moore's Algorithm

The first iterative algorithm was proposed by E. F. Moore in 1956 [9]. The algorithm computes the partition $Q/_\equiv$ by iteratively refining the initial partition $\pi = \{F, Q \setminus F\}$. After termination, $\pi = Q/_\equiv$. In the algorithm, the partition π is encoded as a mapping that assigns to each state $p \in Q$ a block-identifier π_p, which is a bit-string. The number of blocks in π, i.e. the cardinality of the range of π, is denoted $|\pi|$. The value of π at the i:th iteration of lines 8–11 is denoted π^i. The symbol "$\cdot$" on line 10 of the algorithm denotes concatenation (of strings).

Algorithm 1 Moore's DFA minimization algorithm

Input: DFA $A = (Q, \{a_1, \dots, a_k\}, \delta, q_s, F)$
Output: $\pi = Q/_\equiv$

```
 1: i ← 0
 2: for all p ∈ Q do                          ▷ The initial partition
 3:     if p ∈ F then π_p^i ← 1
 4:     else π_p^i ← 0
 5:     end if
 6: end for
 7: repeat
 8:     i ← i + 1
 9:     for all p ∈ Q do
10:         π_p^i ← π_p^{i-1} · π_{δ(p,a_1)}^{i-1} · π_{δ(p,a_2)}^{i-1} · ... · π_{δ(p,a_k)}^{i-1}
11:     end for
12: until |π^i| = |π^{i-1}|
```

The minimal automaton is now obtained by replacing Q by π, q_s by π_{q_s}, F by $\{\pi_f : f \in F\}$, and replacing each transition $(p, a, q) \in \delta$ by transition (π_p, a, π_q), and then removing duplicate transitions.

3.2 Hopcroft's Algorithm

This algorithm was proposed by J. E. Hopcroft in 1971 [5]. We start with some definitions. Let $A = (Q, \Sigma, \delta, q_s, F)$ be a DFA, $S, P \subseteq Q$, $S \cap P = \emptyset$, and $a \in \Sigma$. Then $\langle S, a \rangle$ is called a *splitter*, and

$$\begin{aligned} P \div \langle S, a \rangle &= \{P_1, P_2\}, \text{ where} \\ P_1 &= \{p \in P : \delta(p, a) \in S\} \\ P_2 &= \{p \in P : \delta(p, a) \notin S\}. \end{aligned}$$

If either of P_1 or P_2 is empty, we set $P \div \langle S, a \rangle = P$. Otherwise $\langle S, a \rangle$ is said to *split* P. Furthermore, for $S \subseteq Q$ and $a \in \Sigma$, we define

$$a\langle S \rangle = \{s \in S : \delta(p, a) = s \text{ for some } p \in Q\}.$$

That is, $a\langle S \rangle$ consists of those states in S that have an incoming transition labeled a. As in Moore's algorithm, we encode the current partition with a function π that maps states to integers. We also define

$$B_j = \{p \in Q : \pi_p = j\}.$$

Algorithm 2 Hopcroft's DFA minimization algorithm

Input: DFA $A = (Q, \Sigma, \delta, q_s, F)$
Output: $\pi = Q/_{\equiv}$

1: Execute lines 1 – 6 of Algorithm 1 ▷ Compute π^0
2: $\mathcal{Q} \leftarrow \emptyset$ ▷ Queue set
3: **for all** $a \in \Sigma$ **do**
4: **if** $|a\langle B_0\rangle| < |a\langle B_1\rangle|$ **then** Add $\langle B_0, a\rangle$ to $\mathcal{Q}$
5: **else** Add $\langle B_1, a\rangle$ to $\mathcal{Q}$.
6: **end if**
7: **end for**
8: $i \leftarrow 0$
9: **while** $\mathcal{Q} \neq \emptyset$ **do**
10: $i \leftarrow i + 1$; $size \leftarrow |\pi^{i-1}|$
11: Pick and delete a splitter $\langle S, a\rangle$ from $\mathcal{Q}$.
12: **for each** $B_j \in \pi^{i-1}$ which is split by $\langle S, a\rangle$ **do**
13: $size \leftarrow size + 1$
14: **for all** $p \in B_j$, where $\delta(p, a) \in S$ **do**
15: $\pi_p^i \leftarrow size$ ▷ New block-number
16: **end for**
17: **for all** $b \in \Sigma$ **do** ▷ Update $\mathcal{Q}$
18: **if** $\langle B_j, b\rangle \in \mathcal{Q}$ **then**
19: Add $\langle B_{\pi_p^i}, b\rangle$ to $\mathcal{Q}$
20: **else**
21: **if** $|b\langle B_j\rangle| < |b\langle B_{\pi_p^i}\rangle|$ **then**
22: add $\langle B_j, b\rangle$ to $\mathcal{Q}$
23: **else** add $\langle B_{\pi_p^i}, b\rangle$ to $\mathcal{Q}$.
24: **end if**
25: **end if**
26: **end for**
27: **end for**
28: **end while**
29: $\pi \leftarrow \pi^i$

At line 10, the size of the current partition (the number of equivalence classes) is stored in variable *size*. At lines 12–16 we examine all blocks $B_j \in \pi^{i-1}$ that are split by $\langle S, a\rangle$, and all states $p \in B_j$, where $\delta(p, a) \in S$, are given the new block identifier *size* for π^i at line 15. At the end, the minimal automaton $A/_{\equiv}$ can be obtained as in Moore's algorithm.

4. THE ALGORITHMS IN MAP-REDUCE

We assume familiarity with the Map-Reduce model (see e.g. [7]). Here we will only recall that the initial data is stored on the DFS, and each mapper is responsible for a chunk of the input. In a round of Map and Reduce, the mappers emit key-value pairs $\langle K, V\rangle$. Each reducer K receives and aggregates key-value lists of the form $\langle K, [V_1, \ldots, V_\ell]\rangle$. where the $\langle K, V_i\rangle$ pairs were emitted by the mappers.

4.1 Moore's Algorithm in Map-Reduce

Our Map-Reduce version of Moore's algorithm, called Moore-MR, consists of a pre-processing stage, and one or more rounds of map and reduce functions.

Pre-processing: Let $A = (Q, \{a_1, \ldots, a_k\}, \delta, q_s, F)$. We first build a set Δ from δ. The set Δ will consist of annotated transitions of the form (p, a, q, π_p, D), where π_p is a bit-string representing the initial block where the state p belongs, $D = +$ indicates that the tuple represents a transition (an outgoing edge), and $D = -$ indicates that the tuple is a "dummy" transition carrying in its fourth position the information of the initial block of the aforementioned state q. More specifically, for each $(p, a, q) \in \delta$ we insert into Δ

$$\begin{cases} (p, a, q, \mathtt{1}, +) \text{ and } (p, a, q, \mathtt{1}, -) & \text{when } p, q \in F \\ (p, a, q, \mathtt{1}, +) \text{ and } (p, a, q, \mathtt{0}, -) & \text{when } p \in F, q \notin F \\ (p, a, q, \mathtt{0}, +) \text{ and } (p, a, q, \mathtt{1}, -) & \text{when } p \notin F, q \in F \\ (p, a, q, \mathtt{0}, +) \text{ and } (p, a, q, \mathtt{0}, -) & \text{when } p, q \notin F. \end{cases}$$

Recall that Moore's algorithm is an iterative refinement of the initial partition $\{F, Q \setminus F\}$. In the MR version each reducer will be responsible for one or more states $p \in Q$. Since there is no global data structure, we need to annotate each state p with the identifier π_p of the block it currently belongs to. This annotation is kept in all transitions $(p, a_i, q_i), i = 1, \ldots, k$. Hence tuples $(p, a_i, q_i, \pi_p, +)$ are in Δ. In order to generate new block identifiers, the reducer also needs to know the current block of all the above states q_i, which is why tuples $(p, a_i, q_i, \pi_{q_i}, -)$ are in Δ. Furthermore, the new block of state p will be needed in the next round when updating the block annotation of states $r_1, \ldots, r_m$, where $(r_1, a_{i_1}, p), \ldots (r_m, a_{i_m}, p)$ are all transitions leading to p. Thus tuples $(r_i, a_{i_j}, p, \pi_p, -)$ are in Δ.

Map function: Let ν be the number of reducers available, and $h : Q \to \{0, 1, \ldots, \nu - 1\}$ a hashing function. At round i each mapper gets a chunk of Δ as input, and for each tuple $(p, a, q, \pi_p^{i-1}, +)$ it emits the key-value pair $\langle h(p), (p, a, q, \pi_p^{i-1}, +)\rangle$, and for each tuple $(p, a, q, \pi_q^{i-1}, -)$ it emits $\langle h(p), (p, a, q, \pi_q^{i-1}, -)\rangle$ and $\langle h(q), (p, a, q, \pi_q^{i-1}, -)\rangle$.

Reduce function: Each reducer $\rho \in \{0, 1, \ldots, \nu - 1\}$ receives, for all $p \in Q$ where $h(p) = \rho$, all outgoing transitions

- $(p, a_1, q_1, \pi_p^{i-1}, +), \ldots, (p, a_k, q_k, \pi_p^{i-1}, +)$,

as well as dummy transitions

- $(p, a_1, q_1, \pi_{q_1}^{i-1}, -), \ldots, (p, a_k, q_k, \pi_{q_k}^{i-1}, -)$, and
- $(r_1, a_{i_1}, p, \pi_p^{i-1}, -), \ldots, (r_m, a_{i_m}, p, \pi_p^{i-1}, -)$.

The reducer can now compute

$$\pi_p^i \leftarrow \pi_p^{i-1} \cdot \pi_{q_1}^{i-1} \cdot \pi_{q_2}^{i-1} \cdot \ldots \cdot \pi_{q_k}^{i-1},$$

corresponding to line 10 in Algorithm 1 and write the new value π_p^i in the tuples $(p, a, q_j, \pi_p^{i-1}, +)$, for $j \in \{1, \ldots, k\}$, and $(r_j, a, p, \pi_p^{i-1}, -)$, for $j \in \{1, \ldots, m\}$, which it then outputs.

The reducer also outputs a "change" tuple $(p, true)$ if the new value of π_p^i means that the number of blocks in π^i has increased. In order to see how reducer $h(p)$ can determine this internally, consider for example the DFA $A = (\{p, q, r\}, \{a\}, \delta, p, \{r\})$, where $\delta(p, a) = q$, $\delta(q, a) = r$, and $\delta(r, a) = r$.

Round i	π_p^i	π_q^i	π_r^i
0	0	0	1
1	0·0	0·1	1·1
2	00·01	01·11	11·11
3	0001·0111	0011·1111	1111·1111

After round 1 a new block was created for state q, and after round 2 a new block was created for state p. This can be seen from the fact that after round 1, π_p consists of two equal parts 0 and 0, whereas π_q consists of two distinct parts 0 and 1. Similarly, after round 2, π_q still consists of two distinct parts, whereas π_p has changed from two similar

parts to two distinct parts. We conclude that a new block has been created for p but not for q. After round 3, all block-identifiers consist of two distinct parts, as they did after round 2. No new blocks have been created, and the algorithm can terminate.

The above can be generalized as follows: If $|\Sigma| = k$, then a block-identifier π_p is a bit-string that consists of $k+1$ bit-string components from the previous round. If the number of components have increased, it means that a new block-identifier has been created. The salient point is that the increase can be detected inside reducer $h(p)$, which then can emit a "change" tuple $(p, true)$.

Termination detection: At he end of each MR-round, if any change tuples $(p, true)$ have been emitted, it means that another round of MR is needed.

PROPOSITION 1. *At round i of Moore-MR, $\pi_p^i = \pi_q^i$ if and only if $p \equiv_i q$.*

PROOF. We have $\pi_p^0 = \pi_q^0$ iff $p \equiv_0 q$ by construction. Assume that $\pi_p^{i-1} = \pi_q^{i-1}$ iff $p \equiv_{i-1} q$, and suppose that $\pi_p^i = \pi_q^i$. In the i:th Map-Reduce round,

$$\begin{aligned} \pi_p^i &= \pi_p^{i-1} \cdot \pi_{\delta(p,a_1)}^{i-1} \cdot \ldots \cdot \pi_{\delta(p,a_k)}^{i-1}, \text{ and} \\ \pi_q^i &= \pi_q^{i-1} \cdot \pi_{\delta(q,a_1)}^{i-1} \cdot \ldots \cdot \pi_{\delta(q,a_k)}^{i-1}. \end{aligned}$$

By the inductive hypothesis, all the above $(k+1)$ components are equal iff $p \equiv_{i-1} q$ and $\delta(p, a_j) \equiv_{i-1} \delta(q, a_j)$ for all $j \in \{1, \ldots, k\}$ iff $p \equiv_i q$. □

For proving termination of the algorithm, we denote by $sub(\pi_p^i)$ the set of substrings obtained by dividing π_p^i into $(k+1)$ contiguous substrings of equal length.

PROPOSITION 2. *$|sub(\pi_p^i)| = |sub(\pi_p^{i-1})|$ for all states p if and only if $\equiv_{i-1} = \equiv_{i-2}$.*

PROOF. Suppose $|sub(\pi_p^i)| \neq |sub(\pi_p^{i-1})|$ for some state p. We have

$$\begin{aligned} \pi_p^i &= \pi_p^{i-1} \cdot \pi_{\delta(p,a_1)}^{i-1} \cdot \ldots \cdot \pi_{\delta(p,a_k)}^{i-1}, \text{ and} \\ \pi_p^{i-1} &= \pi_p^{i-2} \cdot \pi_{\delta(p,a_1)}^{i-2} \cdot \ldots \cdot \pi_{\delta(p,a_k)}^{i-2}. \end{aligned}$$

From this it is easy to see that if, say, $\pi_{\delta(p,a_j)}^{i-2} \neq \pi_{\delta(p,a_m)}^{i-2}$, then necessarily $\pi_{\delta(p,a_j)}^{i-1} \neq \pi_{\delta(p,a_m)}^{i-1}$. Thus, $|sub(\pi_p^i)| \neq |sub(\pi_p^{i-1})|$ entails that $|sub(\pi_p^i)| > |sub(\pi_p^{i-1})|$. Consequently there are $j, m \in \{1, \ldots, k\}$, such that $\pi_{\delta(p,a_j)}^{i-2} = \pi_{\delta(p,a_m)}^{i-2}$ and $\pi_{\delta(p,a_j)}^{i-1} \neq \pi_{\delta(p,a_m)}^{i-1}$. (Note: $\pi_{\delta(p,a_j)}^{i-1}$ and $\pi_{\delta(p,a_j)}^{i}$ could also be π_p^{i-1} and π_p^i.) Proposition 1 now entails that $\delta(p, a_j) \equiv_{i-2} \delta(p, a_m)$ and $\delta(p, a_j) \not\equiv_{i-1} \delta(p, a_m)$. Consequently, $\equiv_{i-1} \neq \equiv_{i-2}$.

Suppose then that $|sub(\pi_p^i)| = |sub(\pi_p^{i-1})|$, for all states p. This means that for all states p and all $j, m \in \{1, \ldots, k\}$, $\pi_{\delta(p,a_j)}^{i-1} = \pi_{\delta(p,a_m)}^{i-1}$ if and only if $\pi_{\delta(p,a_j)}^{i-2} = \pi_{\delta(p,a_m)}^{i-2}$. Then, by Proposition 1,

$$\delta(p, a_j) \equiv_{i-1} \delta(p, a_m) \Leftrightarrow \delta(p, a_j) \equiv_{i-2} \delta(p, a_m) \qquad (*)$$

Thus $q/_{\equiv_{i-1}} = q/_{\equiv_{i-2}}$ for every state q that is of the form $\delta(p, a_j)$ for some p and a_j. Since we assume that the DFA has no inaccessible states, this is true for all states except possibly the initial state q_s. For the initial state q_s, if $q_s/_{\equiv_{i-1}} \neq q_s/_{\equiv_{i-2}}$, there would have to be an $a_j \in \Sigma$ such that $|sub(\pi_{\delta(q_s,a_j)}^{i-1})| > |sub(\pi_{\delta(q_s,a_j)}^{i-2})|$, implying, as in the first part of the proof, that $\delta(q_s, a_j)/_{\equiv_{i-1}} \neq \delta(q_s, a_j)/_{\equiv_{i-2}}$, contradicting $(*)$. Thus necessarily $\equiv_{i-1} = \equiv_{i-2}$. □

Dealing with large block-labels: It is easy to see that the bitstrings π_p, encoding the block-identifiers, grow with a factor of $k+1$ at each round, meaning that the length of each π_p is $O((k+1)^i \log n)$ bits at round i. In order to avoid such excessive growth we apply the technique of Perfect Hashing Functions (PHF). A PHF is a total injective function which takes a subset of size s of a set S and maps it to a set T where $|T| = s$ and $s \ll |S|$.

However, PHF requires some shared data structures which is not applicable in Map-Reduce environment. In order to implement PHF in MR, in each reducer ρ, a new block number is assigned from the range $\{\rho \cdot n, \ldots, \rho \cdot n + n - 1\}$. Since there can be at most n reducers, the resulting block numbers will be in the range $\{0, \ldots, n^2 - 1\}$, and our hashing scheme is thus guaranteed to be injective. We call the scheme *Parallel PHF (PPHF)*. Technically, applying the PPHF is a separate MR job, following the main map and reduce functions in each round.

4.2 Hopcroft's Algorithm in Map-Reduce

This section describes Hopcroft-MR, a Map-Reduce version of Hopcroft's algorithm. It consists of a pre-processing stage, and rounds of a sequence of two map-reduce jobs and a wrap-up job.

Pre-processing. We first produce the set

$$\Delta = \{(p, a, q, \pi_p^0, +) : (p, a, q) \in \delta\},$$

where $\pi_p^0 = \texttt{1}$ if $p \in F$, and $\pi_p^0 = \texttt{0}$ otherwise. In this stage we also build a set Γ_0, containing information about $a\langle B\rangle$, for all $a \in \Sigma$, and blocks B in the current partition π^0. More specifically,

$$\Gamma_0 = \bigcup_{B_j \in \pi^0} \{(p, a, q, \pi_q^0, -) : (p, a, q) \in \delta, q \in B_j\},$$

Again, the initial value of π_q is `1` or `0` as in Moore-MR. The sets Δ and Γ_0 are stored in the DFS. After the pre-processing we execute rounds of two map-reduce jobs and a wrap-up. The main idea is that, in each round, for each splitter $\langle S, a\rangle \in \mathcal{Q}$, and each q, where $q \in S$, the reducer $h(q)$ in the first MR-job calculates a new block-id for all states p, where $(p, a, q, \pi_p, +) \in \Delta$. It will be the responsibility of the reducer $h(p)$ in the second MR-job to update the block-id in all relevant tuples.

First map function. At round i, each mapper is assigned a chunk of Δ and chunk of Γ_{i-1} according to $\mathcal{Q}$. Note that before the second and subsequent rounds, the set Γ_{i-1} is updated by the second job of the first round, according to the partition π^{i-1}. We have

$$\Gamma_{i-1} = \bigcup_{B_j \in \pi^{i-1}} \{(p, a, q, \pi_q^{i-1}, -) : (p, a, q) \in \delta, q \in B_j\}.$$

For each tuple $(p, a, q, \pi_p^{i-1}, +)$ in its chunk of Δ the mapper emits key-value pair $\langle h(q), (p, a, q, \pi_p^{i-1}, +)\rangle$, and for each tuple $(p, a, q, \pi_q^{i-1}, -)$ in its Γ_{i-1}, the mapper emits

$\langle h(q), (p, a, q, \pi_q^{i-1}, -)\rangle$. After mapping all splitters, the Master removes all elements of $\mathcal{Q}$.

First reduce function. Reducer $h(q)$ then receives, for each $\langle S, a\rangle$ in $\mathcal{Q}$, where $q \in S$, the tuples

- $(p_1, a, q, \pi_{p_1}^{i-1}, +), \ldots, (p_m, a, q, \pi_{p_m}^{i-1}, +)$,

$m \leq n$, of incoming a-transitions to q, and a subset of tuples

- $(p_1, a, q, \pi_q^{i-1}, -), \ldots, (p_m, a, q, \pi_q^{i-1}, -)$,

namely those that were in the current Γ_{i-1}. If state q had no incoming a-transitions, or if q doesn't participate in any of the current splitters, the reducer does nothing. Otherwise, some of the tuples $(p_j, a, q, \pi_q^{i-1}, -)$, where $j \in \{1, \ldots, m\}$ were received, and for each such tuple, the state p_j needs to move to a new block. The identifier of this new block will be computed from the values of $\pi_{p_j}^{i-1}, \pi_q^{i-1}$, and $\beta_{p_j}^a$, where $\beta_{p_j}^a$, is a bitvector of length $k = |\Sigma|$, such that $\beta_{p_j}^a(i) = \mathtt{1}$ if $a = a_i$, and $\beta_{p_j}^a(i) = \mathtt{0}$, otherwise. The reducer will output *update tuples*

- $(p_j, \pi_{p_j}^{i-1}, \beta_{p_j}^a, \pi_q^{i-1})$

Note that reducer $h(q)$ will possibly also receive transitions tuples $(p', a', q', \pi_{p'}^{i-1}, +)$, in case $h(q') = h(q)$. These states p' will be updated only if $(p', a', q', \pi_{q'}^{i-1}, -)$ also was received.

Second map function. Each mapper is assigned its chunks of Δ as before, a chunk of Γ_{i-1}, and a chunk of the update tuples produced by the first map-reduce job. The mapper emits key-value pairs $\langle h(p), (p, a, q, \pi_p^{i-1}, +)\rangle$ from Δ, and $\langle h(p), (p, \pi_p^{i-1}, \beta_p^a, \pi_{\delta(p,a)}^{i-1})\rangle$ from its assigned update tuples.

Second reduce function. Each reducer $h(p)$ will receive tuples

- $(p, a_1, \delta(p, a_1), \pi_p^{i-1}, +), \ldots, (p, a_k, \delta(p, a_k), \pi_p^{i-1}, +)$

from Δ, and possibly update tuples

- $(p, \pi_p^{i-1}, \beta_p^{a_{i_1}}, \pi_{\delta(p,a_{i_1})}^{i-1}), \ldots, (p, \pi_p, \beta_p^{a_{i_m}}, \pi_{\delta(p,a_{i_m})})$,

where $m \leq k$. If the reducer received the update tuples, it first computes $\beta_p = \bigvee_{j=1}^m \beta_p^{a_{i_j}}$, and will then write the value

$$\pi_p^i = \pi_p^{i-1} \cdot \beta_p \cdot \pi_{\delta(p,a_{i_1})}^{i-1} \cdot \ldots \cdot \pi_{\delta(p,a_{i_m})}^{i-1},$$

in the above tuples from Δ and Γ_{i-1}. Finally, the reducer outputs all its updated (or not) Δ tuples and updated (or not) Γ_{i-1} tuples as Γ_i.

PROPOSITION 3. *At the end of i:th round of Hopcroft-MR, $\pi_p^i = \pi_q^i$ if and only if $p \equiv_i q$.*

PROOF. We have $\pi_p^0 = \pi_q^0$ iff $p \equiv_0 q$ by definition. Assume that $\pi_p^{i-1} = \pi_q^{i-1}$ iff $p \equiv_{i-1} q$, and suppose that $\pi_p^i = \pi_q^i$. Then necessarily $\pi_p^{i-1} = \pi_q^{i-1}$, $\beta_p = \beta_q$, and $\pi_{\delta(p,a_{i_j})}^{i-1} = \pi_{\delta(q,a_{i_j})}^{i-1}$ for all $i_j \in \{i_1, \ldots, i_m\}$. By the inductive hypothesis $p \equiv_{i-1} q$ and $\delta(p, a_{i_j}) \equiv_{i-1} \delta(q, a_{i_j})$, for all $i_j \in \{i_1, \ldots, i_m\}$. The remaining possibility is that $\delta(p, a_j) \not\equiv_{i-1} \delta(q, a_j)$, for some j such that $\beta_p(j) = \beta_q(j) = \mathtt{0}$. Then, for all splitters $\langle S, a_j\rangle$ in $\mathcal{Q}$ at round i, there are no states $r \in S$, such that $\delta(p, a_j) = r$ or $\delta(q, a_j) = r$. On the other hand, since $p \equiv_{i-1} q$ it must be that $\delta(p, a_j) \equiv_{i-2} \delta(q, a_j)$. Since we had $\delta(p, a_j) \not\equiv_{i-1} \delta(q, a_j)$, it follows that either $\delta(p, a_j)$ or $\delta(q, a_j)$ was moved to new block at round $i-1$, which means that a sub-block of $\pi_{\delta(p,a_j)}^{i-1}$ or $\pi_{\delta(q,a_j)}^{i}$ was put in $\mathcal{Q}$ for round i, contradicting our assumption that $\beta_p(j) = \beta_q(j) = \mathtt{0}$.

Suppose then that $p \equiv_i q$. Then $p \equiv_{i-1} q$ and by the inductive hypothesis, $\pi_p^{i-1} = \pi_q^{i-1}$. Also, for all $j \in \{1, \ldots, k\}$, $\delta(p, a_j) \equiv_{i-1} \delta(q, a_j)$, and by the inductive hypothesis $\pi_{\delta(p,a_j)}^{i-1} = \pi_{\delta(q,a_j)}^{i-1}$. Denote $\pi_{\delta(p,a_j)}^{i-1}$ by S. We have two cases to consider: In the first case $\langle S, a_j\rangle$ is in $\mathcal{Q}$ at round i. Then $\beta_p(j) = \beta_q(j) = \mathtt{1}$ and reducer $h(p)$ will in the second job concatenate $\pi_{\delta(p,a_j)}^{i-1}$ generated in the first job by reducer $h(\delta(p, a_j))$ to π_p^i and reducer $h(q)$ will concatenate $\pi_{\delta(q,a_j)}^{i-1}$ to π_q^i. Since $\pi_{\delta(p,a_j)}^{i-1} = \pi_{\delta(q,a_j)}^{i-1}$, it follows that $\pi_p^i = \pi_q^i$. In the second case, $\langle S, a_j\rangle$ is not in $\mathcal{Q}$ at round i. Then $\beta_p(j) = \beta_q(j) = \mathtt{0}$, meaning that $\pi_{\delta(p,a_j)}^{i-1}$ and $\pi_{\delta(q,a_j)}^{i-1}$ were not part of π_p^i and π_q^i, and that therefore $\pi_p^i = \pi_q^i$. □

Wrap-up. The Master now executes lines 17 – 26 of Algorithm 2. If there is any split, Master will put new splitters into $\mathcal{Q}$ and initiates another round of the two map-reduce jobs.

Dealing with large block-labels: In Hopcroft-MR the block-identifiers π_p consist of $(k+2)^i$ bits after round i. In order to avoid long bitstrings, between rounds we use a PHF with range $\{0, \ldots, n^2 - 1\}$, as in Moore-MR.

4.3 Communication Cost of the Algorithms

It is known that, in the worst case, $\equiv = \equiv_{n-2}$, where n is the number of states in the DFA [9]. This happens in "linear" DFAs, such as $A = (\{p, q, r\}, \{a\}, \delta, \{r\})$ from Section 4.1. This means that Algorithm Moore-MR needs $n-1$ rounds in the worst case. It follows from Lemma 9 in [2], that Hopcroft-MR requires the same number of rounds as Moore-MR.

We need to account for the amount of data communicated in each round. In Moore-MR each transition $(p, a, q) \in \delta$ gives rise to tuples $(p, a, q, \pi_p, +)$ and $(p, a, q, \pi_q, -)$ in Δ. The first tuple is mapped to $h(p)$ and the second to $h(p)$ and $h(q)$. This gives a replication rate of 3. A state can be encoded using $\log n$ bits and an alphabet symbol using $\log k$ bits. Since block-labels are in the range $\{0, \ldots, n^2 - 1\}$ using PPHF, these require $2 \log n$ bits. We note that since PPHF actually is a Map-Reduce job, the mappers in this job emit updated block numbers which require $(k+1) \log n$ bits as a result of the concatenation operations in the main Map-Reduce job. Thus a tuple $(p, a, q, \pi_p/\pi_q, +/-)$ requires $\log n + \log k + \log n + (k+1) \log n + 1 = \Theta(k \log n)$ bits. Since there are kn transitions, a total of $\Theta(k^2 n \log n)$ bits need to be communicated.

The reducers in Moore-MR output the above updated two type of tuples, and possibly an update tuple (p, true). It follows from [5] that the total number of update tuples ever emitted is bounded by $O(n \log n)$, which however is absorbed in one round by $O(k^2 n \log n)$. We therefore have

PROPOSITION 4. *The communication cost for Moore's MR-algorithm is $O(k^2 n^2 \log n)$*

The analysis for Hopcroft-MR is similar, except that there are two MR jobs in each round, and only two tuples are

mapped for every transition in the input DFA. As in Moore-MR, there can be at most $n \log n$ update tuples in total. All of this yields $O((\log n + \log k)kn)$ bits of communication per round. We can assume $k < n$, which gives us $O(kn \log n)$ bits per round.

PROPOSITION 5. *The communication cost for Hopcroft's MR-algorithm is* $O(kn^2 \log n)$

The effect of skewed input. One of the problems with the Map Reduce framework is that the reducers might not be evenly utilized. This happens when the input is "skewed" in some way. We note that in Moore-MR, each reducer $h(p)$ receives k tuples representing outgoing transitions, and k dummy tuples. However, reducer $h(p)$ also receives dummy tuples representing transitions leading into p. There is at least one, and at most n incoming transitions for each state. Thus we can expect that those reducers $h(p)$, where p has a large number of incoming transitions, have to deal with larger inputs than those reducers $h(p')$, where p' has only a few incoming transitions. Thus the reducers $h(p')$ might have to wait for the $h(p)$ reducers.

The same phenomenon occurs in the first Reduce job in Hopcroft-MR, where each reducer $h(q)$ receives tuples corresponding to transitions leading into state q. However, in the second Reduce job all reducers receives tuples corresponding to outgoing transitions. However, Hopcroft-MR is *inherently* sensitive to skewness, as the splitting is determined by a state q, for all blocks B that contain a state p, such that $(p, a, q) \in \delta$. On the other hand, Moore-MR determines a new block for a state p based on all its outgoing transitions, and each sate has exactly k outgoing transitions. The possible skewness in number of incoming transitions is only manifested in the way we update the dummy tuples representing transitions incident upon p. We are currently exploring ways of circumventing this skewness sensitivity in Moore-MR.

5. EXPERIMENTAL RESULTS

In this section we examine the behavior of Moore-MR and Hopcroft-MR by implementing them on Apache Hadoop 2.7.2 on a cluster of two machines, and then running the algorithms on various types of DFA's. The number of reducers in the experiments was set to 128.

5.1 Data Generation

We first briefly describe the sample data we ran the algorithms on. In a DFA $(Q, \Sigma, \delta, q_s, F)$ where $|Q| = n$ and $|\Sigma| = k$, we assume that $Q = \{1, \ldots, n\}$, q_s is state 1, and $\Sigma = \{1, \ldots, k\}$. We will generate four types of DFA's:

• **Slow DFA's.** This family of automata was described in [5]. Slow DFA's behave badly for Hopcroft's and Moore's (serial) algorithms. Here each newly generated block has a constant number of states and at each iteration only one block can be split. The prototypical slow DFA is a "linear" one, where $F = \{n\}$ and $\delta(i, j) = i + 1$, for all $i \in Q \setminus \{n\}$ and $j \in \Sigma$. Additionally, $\delta(n, j) = n$ for all $j \in \Sigma$. Linear DFA's are already minimal.

• **Circular DFA's.** These DFA's were also utilized in original paper of Hopcroft. For all states i and alphabet symbols j, we compute $\delta(i, j)$ as follows: [1]

$$\delta(i,j) = \begin{cases} (i+j) \bmod n & \text{if } j \leq \lceil \frac{k}{2} \rceil \\ (i + \lfloor \frac{k}{2} \rfloor - j) \bmod n & \text{otherwise.} \end{cases}$$

We also have $F = \{i \in Q \ : \ i \bmod k = 0\}$. This type of DFA does not exhibit any skew. If we assume that n is a multiple of k, we will have $i \equiv j$ iff $i \bmod k = j \bmod k$, so the minimal version of the DFA is a similar circular DFA, only with k states.

• **Replicated-Random DFA's.** We first generate a random DFA and then make k copies of it. We add a "global" start state 0, from which symbol i takes us to (what was) the start state of the ith copy DFA. The minimized version of the Replicated-Random DFA is equal to the minimized version of the original random DFA plus the new start state 0, and its transitions to the old start state on all symbols.

• **Star DFA's.** The last type of DFA has a star shape, where $F = \{n\}$. For all states i and alphabet symbols j, we compute $\delta(i, j)$ as follows:

$$\delta(i,j) = \begin{cases} \big(i \bmod (n-1)\big) + 1 & \text{if } i \neq n \text{ and } j = 1 \\ n & \text{if } i \neq n \text{ and } j \neq 1 \\ n & \text{if } i = n. \end{cases}$$

This type of DFA exhibits drastic skew, and will be minimized to $(\{1, n\}, \{1, \ldots, k\}, \gamma, 1, \{n\})$, where $\gamma(1, j) = n$ for all $j \in \{2, \ldots, k\}$, $\gamma(1, 1) = 1$, and $\gamma(n, j) = n$ for all $j \in \{1, \ldots, k\}$.

5.2 Experiments

• **Slow DFA's.** We compared Slow DFA's, each with 4 to 512 states and alphabet size $k = 2$. As we know, Moore-MR updates all transitions in each round. According to Table 1, in this particular dataset, the minimization takes exactly n rounds for Moore-MR. In each round all n states are involved. Thus, this requires $n^2 \times O(k \log n) = O(n^2 k \log n)$ bits. Since $k = 2$, we can consider it as a constant, so the bound becomes $O(n^2 \log n)$. On the other hand, in Hopcroft-MR the total number of updated states is independent from the number of rounds and is bounded by $O(n \log n)$. Figure 1 clearly shows the above polynomial relation between these two algorithms. However, Figure 2 shows that Hopcroft-MR takes more time than Moore-MR. This is due to the fact that Hopcroft-MR has one more job in each round. The relation here is linear and exhibits the combined impact of number of map-reduce jobs and rounds on execution time.

[1] Since initial state is labeled as 1, we map $\delta(i, j) = 0$ to n.

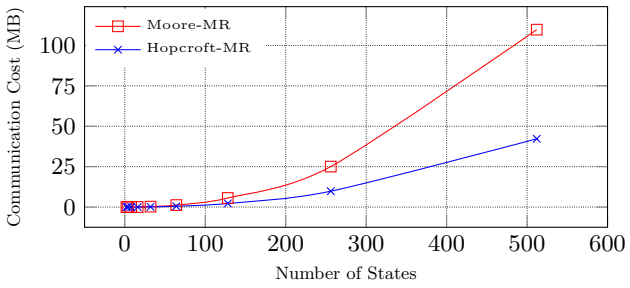

Figure 1: Communication cost on slow DFA for the alphabet size $k = 2$

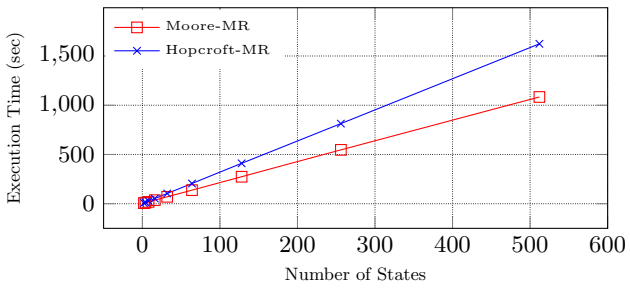

Figure 2: Execution time on slow DFA for the alphabet size $k = 2$

• **Circular DFA's.** The main feature of this class of automata is that each state has k incoming transitions. It creates a uniform distribution of transitions both in mapping schemas based on source state (the Key-Value pairs $\langle h(p), (p, a, q, \pi_p, +)\rangle$ in Moore-MR and in the second map function in Hopcroft-MR) and in mapping schemas based on target state ($\langle h(q), (p, a, q, \pi_q, -)\rangle$ in Moore-MR and the first map function in Hopcroft-MR).

As can be seen from Figure 3, both Moore-MR and Hopcroft-MR have the same communication cost on small alphabets. On the other hand, similarly to the Slow DFA's, when the size of the alphabet is fixed and relatively small, we see from Figure 4 that Hopcroft-MR requires more execution time. This is mainly because the overhead around I/O operations and job execution, as was the case for Slow DFA's. We note that when the number of states is increased, both algorithms perform similarly.

In order to see the effect of the alphabet size, we ran experiments using increasing alphabet size and a fixed number ($n = 1024$) of states. Figures 5 and 6 show that the communication cost and running time of Moore-MR are larger than that of Hopcrpft-MR by a factor of k, as stated earlier in Section 4.3.

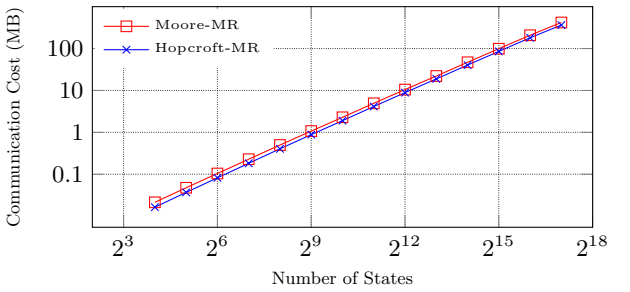

Figure 3: Communication cost on circular DFA for the alphabet size $k = 4$

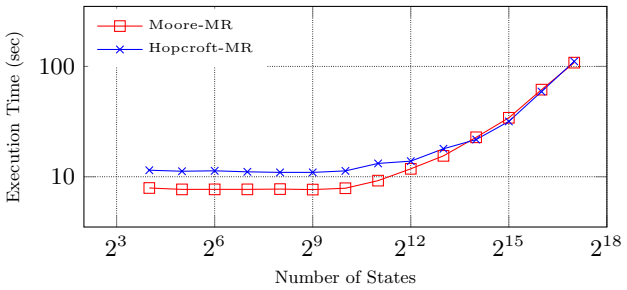

Figure 4: Execution time on circular DFA for the alphabet size $k = 4$

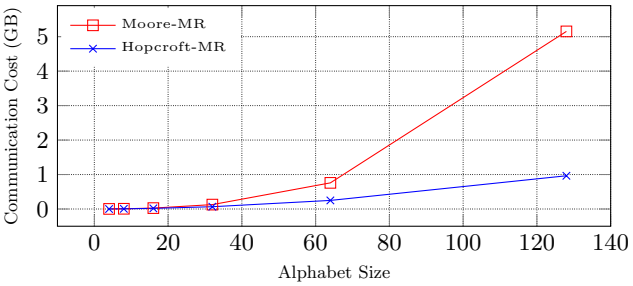

Figure 5: Communication cost on circular DFA for $n = 2^{10}$

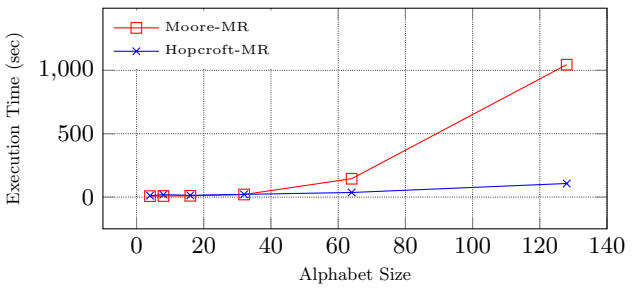

Figure 6: Execution time on circular DFA for $n = 2^{10}$

• **Replicated-Random DFA's.** Here the size of alphabet is fixed and equal to 4. The number of states in the Replicated-Random DFA's varies from 2^4+1 to $2^{17}+1$. Each DFA is replicated four times and the copies are connected using an auxiliary start state. Results are plotted in Figures 7 and 8. These figures show that Replicated-Random DFA's behave similarly to Circular DFA's. We note that the small standard deviation of the distribution of tuples among reducers in the Replicated-Random DFA's (see Table 2) does not affect the performance significantly. The irregular value for the DFA in Figure 8 with 2^5+1 states is, as shown in Table 1, due to the fact that the set of final states for this DFA is empty and it converges at the beginning of the process.

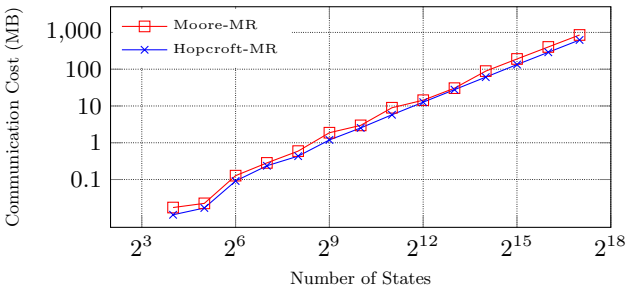

Figure 7: Communication cost on replicated-random DFA for alphabet size $k = 4$

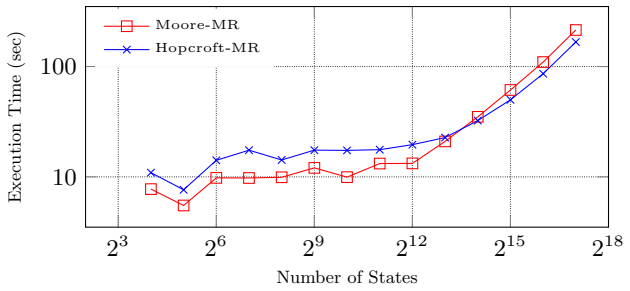

Figure 8: Execution time on replicated-random DFA for alphabet size $k = 4$

• **Star DFA's.** The last dataset is designed to highlight the sensitivity of the algorithms to data skewness. The main property of this dataset is that $(k-1)n$ out of kn transitions will be mapped to one reducer. Interestingly, both algorithms demonstrate the same performance which shows both have the same sensitivity to skewness, as this is a clear consequence of their mapping schemas. Results are plotted in Figures 9 and 10. In conclusion, these two algorithms behave equally when k is fixed and small.

On the other hand, when the number of states is fixed and the alphabet size increases, there is a noticeable difference between Moore-MR and Hopcroft-MR. Figure 11 shows the difference in communication cost between these algorithms when k increases, while, Figure 12 shows this difference in execution time.

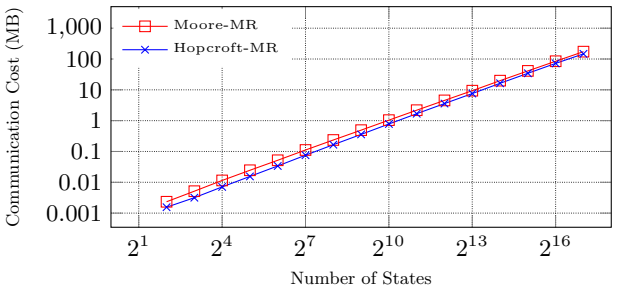

Figure 9: Communication cost on star DFA for alphabet size $k = 4$

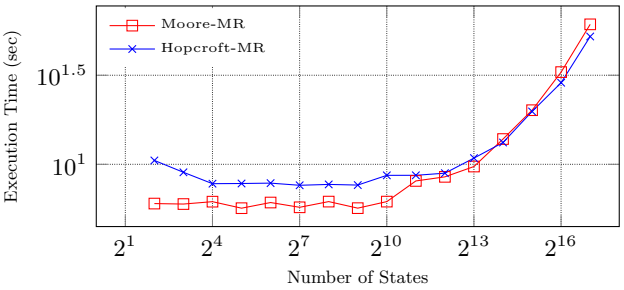

Figure 10: Execution time on star DFA for alphabet size $k = 4$

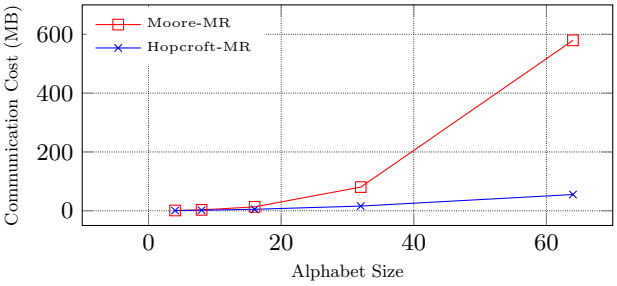

Figure 11: Communication cost on star DFA for $n = 2^{10}$

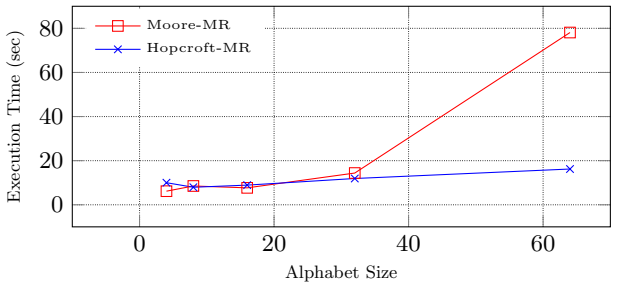

Figure 12: Execution time on star DFA for $n = 2^{10}$

5.3 Summary

The number of rounds required for each algorithm is categorized by the data sets and is listed in Table 1. For the *Slow DFA's*, Moore-MR and Hopcroft-MR both required exactly n rounds, as it is expected. On the other hand, for *Circular DFA's*, both algorithms accomplished the task in a fixed number of rounds, independent from the size of the alphabet. The same results are observed for *Star DFA's*. Lastly, running the algorithms on *Replicated-Random DFA's* shows that Hopcroft-MR has finished the task in fewer rounds than Moore-MR.

Table 1: Number of rounds for minimizing DFA using Moore-MR and Hopcroft-MR with different datasets

Dataset	k	n	M-MR [a]	H-MR [b]
Slow	2	2 to 2^{10}	n	n
Circular	4	2^4 to 2^{17}	3	2
	2^2 to 2^7	2^{10}	3	2
Random	4	2^4+1	3	2
	4^c	2^5+1^c	2	1
	4	2^6+1	4	3
	4	2^7+1	4	4
	4	2^8+1	4	3
	4	2^9+1	5	4
	4	$2^{10}+1$	4	4
	4	$2^{11}+1$	5	4
	4	$2^{12}+1$	4	4
	4	$2^{13}+1$	4	4
	4	$2^{14}+1$	5	4
	4	$2^{15}+1$	5	4
	4	$2^{16}+1$	5	4
	4	$2^{17}+1$	5	4
Star	4	2^2 to 2^{17}	2	1
	2^2 to 2^7	2^{10}	2	1

[a] Moore-MR
[b] Hopcroft-MR
[c] This randomly generated DFA does not have any final state and will minimize into a single state machine.

Figures 13 – 16 represent the data distribution and execution time among all reducers only in the first round of execution. As is expected, for *Circular* and *Replicated-Random DFA's* the data is distributed and executed uniformly. However, for *Star DFA's* these results show that both Moore-MR and Hopcroft-MR are sensitive to skewness. The data distribution shows that one reducer receives most of the input while the rest of the input is evenly shared among all other reducers.

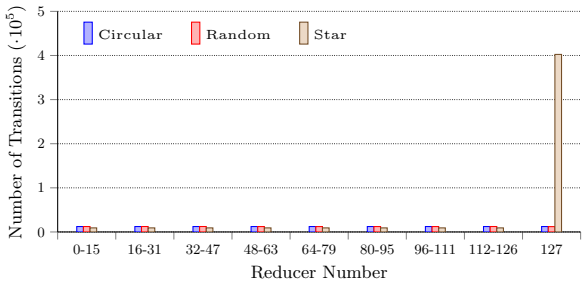

Figure 13: Maximum number of transitions in reducer groups for Moore-MR's algorithm

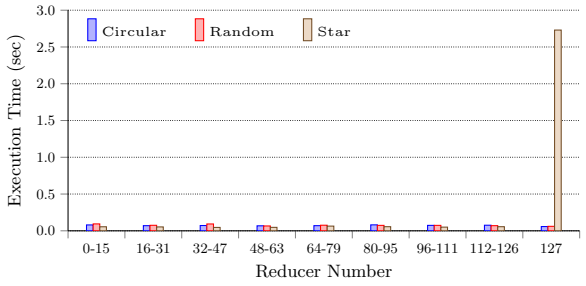

Figure 14: Maximum execution time in reducer groups for Moore-MR's algorithm

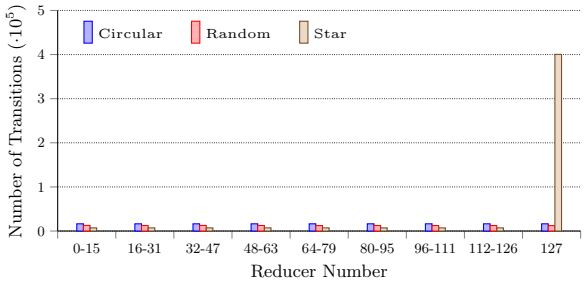

Figure 15: Maximum number of transitions in reducer groups for Hopcroft-MR's algorithm

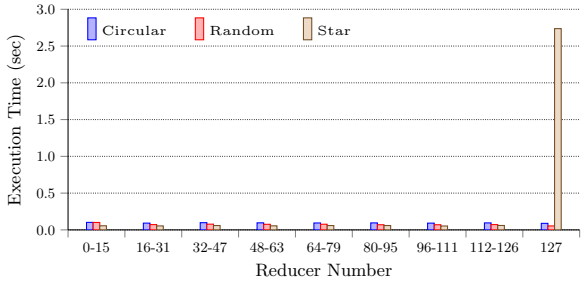

Figure 16: Maximum execution time in reducer groups for Hopcroft-MR's algorithm

Statistical summary of the distribution of the tuples over the reducers is shown in Table 2 for both algorithms. For all four datasets the size of alphabet is set to 4 and the number of states is 2^{17}. The columns "Min." and "Max." show the minimum and maximum number of tuples mapped to each reducer in every job of the all rounds of execution. For Hopcroft-MR we have two rows per DFA type describing the two jobs. In *Circular DFA's*, tuples are evenly distributed for both incoming and outgoing transitions, and the standard deviation for both algorithms is 0. However, a high degree of skewness is observed for *Star DFA's* (all states have one incoming transition except the final state, which has $(k-1)n$ incoming transitions).

Table 2: Statistical summary of the distribution of records among reducers

	Dataset	Min.	Max.	Avg.	Std. Dev.
Moore-MR	Circular	36864	36864	36864	0
	Random	57600	61640	60194	768
	Star	18432	804848	24576	69238
Hopcroft-MR	Circular	16384	16384	16384	0
		20480	20480	20480	0
	Random	24972	28656	26890	671
		35648	37712	36986	407
	Star	2048	788483	8192	69240
		6144	6147	6144	0.26412

6. CONCLUSION

In this paper we developed and studied implementations in Map-Reduce of two algorithms for minimizing DFA's. Our analytical and experimental results show that the Moore-MR algorithm and the Hopcroft-MR behave very similarly when the alphabet size is small, whereas Hopcroft-MR outperforms Moore-MR in communication cost when the cardinality of the alphabet is at least 16, and in wall-clock time when the cardinality is at least 32, both measured on DFA's with 1024 states. Moreover, our study shows that both algorithms are equally sensitive to skewness in the input data. However, Moore-MR (and serial Moore) determines the block of a state based on its (constant number of) outgoing transitions, while Hopcorft-MR (and serial Hopcroft) does it based on the transitions incident upon the state. This means that there is potential to reduce skew-sensitiveness in Moore-MR. In future work we will also investigate the average communication cost of Hopcroft-MR. Finally, the capacity of the reducers (independently of the mapping schema) may affect number of rounds needed. We intend to explore this trade-off in future work.

7. REFERENCES

[1] F. N. Afrati and J. D. Ullman. Transitive closure and recursive datalog implemented on clusters. In *15th International Conference on Extending Database Technology, EDBT, Berlin, Germany*, pages 132–143, 2012.

[2] J. David. Average complexity of Moore's and Hopcroft's algorithms. *Theor. Comput. Sci.*, 417:50–65, 2012.

[3] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[4] G. Grahne, S. Harrafi, A. Moallemi, and A. Onet. Computing NFA intersections in Map-Reduce. In *Proceedings of the Workshops of the EDBT/ICDT Joint Conference (EDBT/ICDT)*, pages 42–45, 2015.

[5] J. E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, New York, 1971.

[6] J. F. JáJá and K. W. Ryu. An efficient parallel algorithm for the single function coarsest partition problem. In *Proceedings of the fifth annual ACM symposium on Parallel algorithms and architectures*, pages 230–239. ACM, 1993.

[7] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[8] Y. Luo, G. H. Fletcher, J. Hidders, Y. Wu, and P. De Bra. External memory k-bisimulation reduction of big graphs. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 919–928. ACM, 2013.

[9] E. F. Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.

[10] V. Rastogi, A. Machanavajjhala, L. Chitnis, and A. D. Sarma. Finding connected components in Map-Reduce in logarithmic rounds. In *29th IEEE International Conference on Data Engineering, ICDE, Brisbane, Australia*, pages 50–61, 2013.

[11] B. Ravikumar and X. Xiong. A parallel algorithm for minimization of finite automata. In *The 10th International Parallel Processing Symposium*, pages 187–191, April 1996.

[12] A. Schätzle, A. Neu, G. Lausen, and M. Przyjaciel-Zablocki. Large-scale bisimulation of rdf graphs. In *Proceedings of the Fifth Workshop on Semantic Web Information Management*, page 1. ACM, 2013.

[13] M. Shaw, P. Koutris, B. Howe, and D. Suciu. Optimizing large-scale semi-naïve Datalog evaluation in Hadoop. In *Datalog in Academia and Industry - Second International Workshop, Datalog 2.0, Vienna, Austria*, pages 165–176, 2012.

[14] Y. N. Srikant. A parallel algorithm for the minimization of finite state automata. *International Journal of Computer Mathematics*, 32(1-2):1–11, 1990.

[15] A. Tewari, U. Srivastava, and P. Gupta. A parallel DFA minimization algorithm. In *High Performance Computing-HiPC*, pages 34–40. Springer, 2002.

[16] B. W. Watson. A taxonomy of finite automata minimization algorithms. Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands, 1993.

[17] M. Yoeli. Canonical representations of chain events. *Information and Control*, 8(2):180 – 189, 1965.