Offering Memory Efficiency utilizing Cellular Automata for Markov Tree based Web-page Prediction Model

Ruma Dutta, Anirban Kundu, and Debajyoti Mukhopadhyay
Web Intelligence & Distributed Computing Research Lab (WIDiCoReL)
Techno India Group (West Bengal University of Technology)
EM 4/1, Sector V, Salt Lake, Calcutta 700091, India
email : {rumadutta2006, anik76in, debajyoti.mukhopadhyay}@gmail.com

Abstract

In this paper, an approach for storing Markov tree, used in various versions of PPM model while predicting next Web-page is proposed. Markov tree requires huge amount of memory. This problem is solved using Cellular Automata which is considered as a fast and inexpensive mechanism. The proposed technique utilizes non-linear Single Cycle Multiple Attractor Cellular Automata (SMACA) which replaces Markov tree for minimizing the memory requirement.

Index Terms - Cellular Automata (CA), Single Cycle Multiple Attractor Cellular Automata (SMACA), Rule Vector (RV), Self Cycle Loop Attractor (SLA), Prediction by Partial Match (PPM), LRS (Longest Repeating Sequence)

1. Introduction

World Wide Web (WWW) has brought revolution to millions of people who wish to access the huge information stored in WWW irrespective of the country they belong to. In doing so, minimizing access latency is an important requirement, users are looking for. Markov Model plays an important role in this area. Markov model is a prediction model where probability of going into next state depends on the previous states. The order of Markov model depends on the number of previous states. In 1st order Markov model only the probability of next state is determined by current state. For second, third, ....nth order the number of previous states are 2, 3, ..., n states respectively including current state. 1st order Markov Model is easy to implement but the prediction accuracy is low. To improve prediction accuracy, higher order Markov Model has to be used. But implementing higher order Markov Model is extremely complicated. To ease the implementation of Markov Model, Markov tree has come into picture. But the Models using Markov tree require huge memory. This problem is solved by many researchers by different methods of pruning the tree efficiently. Here the problem is solved by SMACA where only rules have to be stored instead of whole tree, thus minimizes the memory requirement. For 1st order Markov model the CA implementation has been published in [1]. This paper handles CA implementation for the higher order Markov tree.

This paper is organized as follows:
Section 1 is Introduction
Section 2 discusses Related Work in the field of Web Prefetching
Section 3 discusses about Cellular Automata Preliminaries
Section 4 proposes Our Approach
Section 5 is Conclusion

2. Related Work

There are several Markov predictor tree models which store URLs in tree structure. Two of them are Standard PPM model and LRS PPM Model [3, 5][10]-[15]. The Standard PPM Model uses multiple Markov models to store historical URL paths. Each Markov model partially represents a client session. The model structure is a tree, and each branch is a Markov model with multiple URL predictors. The standard PPM model is shown in Figure 1. Node 0 represents the root of the tree. When a client accesses URL A, the server builds a new branch with root A and sets the access counter to 1. When B comes, it creates another branch with root B. Because A is followed by B in the same session, another node for B as a child node of A has been generated. The process completes until the server has placed all the URLs accessed in the three sessions.

Another approach in Figure 2 is LRS PPM Model. This model keeps the longest repeating subsequences and stores only long branches with frequently accessed URL predictors. A longest sequence is a frequently repeating sequence
in which at least one occurrence of one subsequence belongs to an independent access session. A sequence of URLs that a client accesses more than once is considered a repeated sequence. The server builds the tree in the same way as in Figure 1, but it then scans each branch to eliminate nonrepeating sequences, such as A1/1, B1/1, C1/1.

CA cell (Fig 3(b)) can be represented as a rule as defined in Table 3 [7]. First row of Table 1 represents \(2^3 = 8\) possible present states of 3 neighbours of \(i^{th}\) cell - (i-1), i, (i+1) cells. Each of the 8 entries (3 bit binary string) represents a minterm of a 3 variable boolean function for a 3 neighbourhood CA cell.

![Figure 1. PPM Model for (ABCA1B1C1), (ABC), (A1B1C1)](image)

![Figure 2. LRS PPM Model for (ABCA1B1C1), (ABC), (A1B1C1)](image)

**3. Cellular Automata Preliminaries**

An \(n\) cell Cellular Automata (CA) consists of \(n\) cells (Fig 3(a)) with local interactions. It evolves in discrete time and space. The next state function of three neighbourhood

In subsequent discussions, each of the 8 entries in Table 1 is referred to as a Rule Min Term (RMT). The decimal equivalent of 8 minterms are 0, 1, 2, 3, 4, 5, 6, 7 noted within () below the three bit string. Each of the next five rows of Table 1 shows the next state (0 or 1) of \(i^{th}\) cell. Hence there can be \(2^8 = 256\) possible bit strings. The decimal counterpart of such an 8 bit combination is referred to as a CA rule [7]. The rule of a CA cell represents its next state logic as illustrated in Table 2 for a few example rules.
It can be derived from the truth table (Table 1) of the $i^{th}$ cell, where $q_i^{t+1}$ is the next state of $i^{th}$ cell, while $q_{i-1}^{t}$, $q_{i}^{t}$, and $q_{i+1}^{t}$ are the current states of $(i-1)^{th}$, $i^{th}$, and $(i+1)^{th}$ cells respectively; the $\oplus$ represents XOR logic and ($) denotes AND function. If a CA cell is configured with a specific rule, its next state function implements the truth table as illustrated for sample rules in Table 2. The first two rules 90 and 150 of Table 2 are linear rules employing XOR logic while remaining non-linear rules employ AND logic in addition to XOR. Out of 256 possible rules, as shown in Table 3, there are 7 rules with XOR logic and another 7 rules employ XNOR logic. The Rule 0 sets the cell to state ‘0’ for each of the 8 minterms. The remaining rules are non-linear rules employing AND/OR/NOT logic. Linear and Additive CA employing XOR/XNOR logic have been characterized with matrix algebraic formulation [9].

3.1. Definitions

Definition 1: Reachable state - A state having 1 or more predecessors is a reachable state.
Definition 2: Non-reachable state - A state having no predecessor is termed as non-reachable.
Definition 3: Cyclic state - A state in a cycle of the state transition behavior of a CA. Cyclic states are also referred to as Stable (semi stable) states.
Definition 4: Transient state - A non-cyclic state of a non-group CA is referred to as a transient state.
Definition 5: Attractor Cycle - The set of states in a cycle is referred to as an attractor cycle.
Definition 6: Self-Loop Attractor(SLA) - A single cycle attractor state with self-loop is referred to as SLA.
Definition 7: Rule Vector(RV) - The sequence of rules $< R_{0} R_{1} \cdots R_{i} \cdots R_{n-1} >$, where $i^{th}$ cell is configured with rule $R_i$.
Definition 8: Self Loop State(SLS)-A state forming a cycle

![Table 1. Truth Table of sample rules of a CA cell showing the next state logic for the Minterms of a 3 variable boolean function - The 8 minterms having decimal values 0, 1, 2, 3, 4, 5, 6, 7 are referred to as Rule Minterms (RMTs)](image)

<table>
<thead>
<tr>
<th>Rule No.</th>
<th>Next state function with XOR logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 60</td>
<td>$q_i(t+1) = q_{i-1}(t) \oplus q_i(t)$</td>
</tr>
<tr>
<td>Rule 90</td>
<td>$q_i(t+1) = q_{i-1}(t) \oplus q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 102</td>
<td>$q_i(t+1) = q_i(t) \oplus q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 150</td>
<td>$q_i(t+1) = q_{i-1}(t) \oplus q_i(t) \oplus q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 170</td>
<td>$q_i(t+1) = q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 204</td>
<td>$q_i(t+1) = q_i(t)$</td>
</tr>
<tr>
<td>Rule 240</td>
<td>$q_i(t+1) = q_{i-1}(t)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rule No.</th>
<th>Next state function with XNOR logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 195</td>
<td>$q_i(t+1) = q_{i-1}(t) \oplus q_i(t)$</td>
</tr>
<tr>
<td>Rule 165</td>
<td>$q_i(t+1) = q_{i-1}(t) \oplus q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 153</td>
<td>$q_i(t+1) = q_i(t) \oplus q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 105</td>
<td>$q_i(t+1) = q_{i-1}(t) \oplus q_i(t) \oplus q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 85</td>
<td>$q_i(t+1) = q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 51</td>
<td>$q_i(t+1) = q_{i+1}(t)$</td>
</tr>
<tr>
<td>Rule 15</td>
<td>$q_i(t+1) = q_{i-1}(t)$</td>
</tr>
</tbody>
</table>

Table 2. Linear/additive CA rules employing Next State Function with XOR/XNOR logic

Note: Rule 0 sets the cell to state ‘0’ for each of the 8 minterms.

Note: Set of minterms $T = \{ 7, 6, 5, 4, 3, 2, 1, 0 \}$ represented as $\{ T(7), T(6), T(5), T(4), T(3), T(2), T(1), T(0) \}$ (TM=$m, m=0\,to\,7$) in the text, are noted simply as $q$. 

![Table 2. Linear/additive CA rules employing Next State Function with XOR/XNOR logic](image)
of length 1, that is a self loop with the successor as the state itself.

4. Our Approach

In our approach, SMACA is being used. Typically, a non-linear SMACA consists of $2^n$ number of states where $n$ is the size of SMACA. The structure of a non-linear SMACA has attractors (self-loop or single length cycle), non-reachable states, and transient states. The attractors form unique classes (basins). All other states reach the attractor basins after certain time steps.

In this approach, the root of Markov tree is the attractor in SMACA. The $n^{th}$ order markov tree has $n$ levels in SMACA. Each non-reachable and transient states are the Web-pages. If there are 9 records in Web-log, for $2^{nd}$ order Markov tree, 9-1 = 8 subsequences should be formed considering 3 records in one subsequence. 1 subsequence has been subtracted, as last subsequence does not give any information and last but one subsequence take only 2 records. So, total states in SMACA replacing Markov tree are $3^7 + 2=23$. So, minimum 5 bytes are required to represent the 23+1(for root) states in SMACA as, in CA, states are represented in binary form. If there are 4 distinct Web-pages (excluding the attractor state) then 2 bytes are required to represent these 4 distinct Web-pages. This 2 bytes are merged with 5 bytes calculated previously. So total 7 bytes are required to represent all 24 states. For example, there is a sequence ABACD. In that case there are 4 distinct Web-pages A, B, C, D. In our case 00 represents A, 01 represents B, 10 represents C and 11 represents D. There are 5 records. So, there are 4 subsequences. So, total number of states are $3^3+2+1=12$. total bytes required $4+2=6$. For this sequence, Fig. 6 shows the State transition diagram for $2^{nd}$ order Markov tree.

Algorithm 1 The Algorithm for SMACA generation
Input: Web log
Output: Generated SMACA replacing $n^{th}$ order Markov tree
Step 1 : Scan through the Web log
Step 2 : Calculate the number of states and number of bytes required to represent all states.
Step 3 : Repeat Step 4 thru Step 6 until end of Web log is reached
Step 4 : Take the subequence of $n^{th}$ length for $n^{th}$ order Markov tree
Step 5 : Make 1st item in sequence as non-reachable state and generate unique state value
Step 6: Make other items as transient states and generate unique state value
Step 7: Store generated SMACA rule [2]
Step 8: Stop

Following two Algorithms provides the step necessary to get the next state from SMACA rule.

**Algorithm 2** Binary to RMT

**Input:** Binary bit String $<b_0b_1...b_i...b_{n-1}>$

**Output:** RMT string $<T_0T_1...T_i...T_{n-1}>$ where $T_i \in \{T(0), T(1), T(2), T(3), T(4), T(5), T(6), T(7)\}$ ($i = 0, 1, ..., (n-1)$)

1. $T^0 = <0b_0b_1>$ and $T^{n-1} = <b_{n-2}b_{n-1}0>$
2. For $i = 1$ to $n-2$, set $T^i = jb_{i-1}b_ib_{i+1} >$
3. Output RMT String
4. Stop

**Algorithm 3** The Algorithm for Generating next State Value

**Input:** Current State as decimal value, CA-Size, set of CA rule, number of rules among set of CA rule

**Output:** Next State as decimal value

1. Convert the current state decimal value to binary
2. Convert the binary number to RMT using Algorithm 2
3. Take the particular rule, taken as input from the set of CA rule
4. Convert the rule to binary form
5. Take the value of the particular position of RMT from the rule
6. Decrement CA-Size
7. Repeat Step 2 to Step 6 until CA-Size=0
8. Stop

Once the rule has been stored as per Algorithm 1, the next state value can be calculated according to Algorithm 3 [1]. For $n^{th}$ order $(n-1)$ times the Algorithm 3 has to be applied to get the next state value. From this next state value, next Web-page is predicted. Following Algorithm gives the steps required to predict the next Web-page.

**Algorithm 4** The Algorithm for Predicting next State from Generated SMACA

**Input:** SMACA rule, order n, previous sequence

**Output:** Predicted Web-page

1. Take one of the non-reachable state which tallies with 1st Web-page of order n
2. For $i=1$ to $(n-1)$ repeat Step 3 thru Step 4
3. Check whether next state tallies with the required sequence
4. If Step 3 != TRUE break the loop go to Step 1
5. Stop

### 4.1. Analysis of Storage requirement

The comparison of the memory requirement of Markov tree and that of proposed method is shown in Figure 8. Let us suppose, there are 50 records in Web-log and 10 of them are distinct. For 2nd order Markov tree, $150*10$ (on average 10 bytes per URL) = 1500 bytes are required whereas in SMACA approach for 10 distinct states 4 bytes (as $2^4 = 16$) and for 150 states i.e., $2^8$, so $(8+4)*8=96$ bytes are required. 2nd order Markov tree is considered as it has been seen, for client-based prediction, 2nd order Markov tree gives the optimum prediction accuracy [5]. For another case, let us assume that there are 100 records in Web log and 25 are the distinct Web-pages. 25 distinct Web-pages (states) require 5 bytes and for 300 states 9 bytes are required. So in this case for SMACA $(9+5)*8$ bytes i.e. 112 bytes are required compared to the Markov tree case, where $300*10=3000$ bytes are required. So substantial amount of memory are saved after taking the approach of SMACA implementation.

For the case of Standard PPM Model of order 6 (it has been found experimentally that beyond 6th order, Standard PPM does not show any improvement in prediction accuracy), the storage requirement is more and if it is implemented in SMACA it will give better result. The no. of nodes of 50 records, will be $(6+5+4+3+2+1)*6+2+1= 123$. The no. of bytes required is $123*10=1230$. Figure 9 is showing the graph of comparison between the storage requirement of PPM and SMACA.
5. Conclusion

This paper discussed the mechanism of replacing Markov tree models with SMACA in predicting next Web-page. This approach solves the problem of huge memory requirement for storing Markov tree. As this is a replacement of Markov tree based model, here memory requirement is less with prediction accuracy same as per the predictive model chosen. The proposed mechanism has been analysed for client-based prediction particularly, but in server-based model it is equally advantageous as the Markov tree for this case has to store many number of records and to achieve better predictive accuracy higher order markov tree is required and higher order means more nodes of the tree.

References


