Vector Compaction Using
Dynamic Markov Models

Radu Marculescu, Diana Marculescu,
and Massoud Pedram

CENG 96-14

Department of Electrical Engineering - Systems
University of Southern
Los Angeles, California 90089-2562
(213) 740-4458

February 1996
Vector Compaction Using Dynamic Markov Models

Abstract
Evaluation of power dissipation is a critical step in the design of today’s ICs. Power dissipation is strongly input pattern dependent and hence, to obtain accurate power values, one must simulate the circuit with a large number of input vectors that typify the application data. The goal of this paper is to present an effective and robust technique for compacting a large sequence of input vectors into a much smaller input sequence so as to reduce the circuit/gate level simulation time by orders of magnitude and maintain the accuracy of the power estimates. In particular, this paper introduces and characterizes a family of dynamic Markov trees that can model complex spatiotemporal correlations which occur during power estimation both in combinational and sequential circuits. This new framework is very effective and flexible: the Markov model itself is derived through a one-pass traversal of the initial sequence and it can be used after that with any available simulator to derive power consumption. As the results demonstrate, large compaction ratios of 1-2 orders of magnitude can be obtained without significant loss (less than 3% on average) in the accuracy of power estimates.

1. Introduction

CAD tools have played a significant role in the efficient design of the high-performance digital systems. In the past, time and area were the primary concerns of the CAD community during the optimization phase. More recently, circuit testability was added as yet another important consideration during the design process. With the growing need for low-power electronic circuits and systems, power analysis and low-power synthesis have become crucial tasks that must also be addressed. It is expected that, in the forthcoming years, power issues will receive increasing attention due to the widespread use of portable applications and the desire to reduce packaging and cooling costs of high-end systems.

Power estimation is in general a difficult problem; the key task in this process is the accurate and fast estimation of average switching activity. To date, both simulative [1]-[4] and nonsimulative approaches [5]-[10] have been tried, each one having its own advantages and limitations [11]. More specifically, general simulation techniques provide sufficient accuracy, but at high computational cost; it is simply expensive to simulate thousands of vectors. On the other hand, nonsimulative approaches (best represented by probabilistic power estimation techniques) are in general faster, but less accurate than those based simulation; usually, the input correlations and the reconvergent fan-out in the target circuit make things very complicated and simplifying assumptions (like input independence) become mandatory. During the last years, the gap between simulative and nonsimulative approaches remained basically the same, despite remarkable advances made in both directions.

As a conclusion, a number of issues appear to be important for power estimation and low-power synthesis. The input statistics which must be properly captured and the length of the input sequences which must be applied are two such issues. Generating a minimal-length sequence of input vectors that satisfies these statistics in not trivial. More precisely, LFSRs which have traditionally found use in testing or functional verification [12], are of little or no help
here. The reason is that more elaborate set of input statistics must be preserved or reproduced during sequence generation for use by power simulators. One such attempt is [13] where authors use deterministic FSMs to model user-specified input sequences. Since the number of states in the FSM is equal to the length of the sequence to be modeled, the ability to characterize anything else but short input sequences is severely limited. A more elaborate and effective technique was presented in [14] where, based on stochastic sequential machines, the authors succeed in compacting large sequences without significant loss in accuracy. However, in the present research, the limitations of that approach are pointed out and overcome by the proposed technique.

The present paper improves the-state-of-the-art by providing an original solution for vector compaction problem which potentially reduces the gap between simulative and nonsimulative approaches. Traditionally, data compression techniques were aimed to lossless compression, that is the ability to encode a body of data, transmit it eventually over a communication line and finally, decode it uniquely, without any loss of information during this process. Our objective in this paper is slightly different: having an initial sequence (assumed representative for some target circuit), we target lossy compression [15], that is the process of transforming an input sequence into a smaller one, such that the new body of data represents a good approximation as far as total power consumption is concerned. We use throughout the paper the term ‘compaction’ instead of ‘compression’, because we think it better suits our intentions.

The foundation of our approach is probabilistic in nature; it relies on adaptive (dynamic) modeling of binary input streams as first-order Markov sources of information and is applicable both to combinational and sequential circuits. The adaptive modeling technique itself (best known as Dynamic Markov Chain or DMC modeling) was introduced very recently in the literature on data compression [16] as a candidate to solve various compression problems. From the very beginning, this technique looked very promising and indeed, in most practical situations, has been more effective than any other compression technique available to date. However, the original model introduced in [17] is not completely satisfactory for our purpose. In this paper, we thus extend the initial formulation to manage not only correlations among adjacent bits that belong to the same input vector, but also correlations between successive input patterns.

As demonstrated and supported by practical evidence, this new framework is extremely effective in power estimation. The basic idea is illustrated in Fig.1. To evaluate the total power consumption of a target circuit for a given input sequence \( L_0 \) (Fig.1a), we derive first the Markov model of the input sequence through a one-pass traversal technique and after that, having this compact representation, we generate a much shorter sequence \( L \), equivalent with \( L_0 \), which can be used with any available simulator to derive accurate power estimates (Fig.1b).
We point out here that the present approach can be used without any difficulty to generate benchmark data for power estimation, that is, input sequences with different lengths that satisfy a set of user-prescribed characteristics in terms of word-level transitions or conditional probabilities.

To conclude, both simulation-based approaches and probabilistic techniques for power estimation may benefit from this research. The issues brought into attention in this paper are new and represent an important step toward reducing the gap between the simulative and probabilistic techniques commonly used in power estimation. Finally, the concept of DMC modeling itself might find useful applications in other CAD fields.

The paper is organized as follows: Section 2 reviews the basic concepts of DMC modeling technique. Section 3 formalizes the power-oriented vector compaction problem and discusses parameters which makes this approach effective in practice. Section 4 presents a DMC-based procedure for vector compaction. In sections 5 and 6, we give some practical considerations and experimental results, respectively. Finally, we conclude by summarizing our main contribution.

2. Background on Dynamic Markov models

Modeling for data compaction in digital systems involves essentially the derivation of certain source-string events and their contexts (the set of bits or words surrounding some bit or word under consideration), which uniquely describe the original source string. We consider therefore the model as having two parts: 1) the structure which is the set of events and their contexts and 2) the parameters which are probabilities assigned to the events. The structure is intended to capture the characteristics of the entire set of sequences under consideration while the parameters are tailored to each individual sequence.

Without loss of generality, in what follows we restrict ourselves to finite binary strings, that is, finite sequences consisting only of 0's and 1's. The set of events of interest is the set $S$ of all finite binary sequences on $k$ bits. A particular sequence $S_1$ in $S$ consists of vectors $v_1, v_2, ..., v_n$ (which may be distinct or not), each having a positive occurrence probability. Indices $1, 2, ..., n$ represent the discrete time steps when a particular vector is applied to a

1. Throughout the paper, we may refer occasionally to vectors $v_1, v_2, ..., v_n$ as 'symbols' or 'states'.
target circuit. Imposing a total ordering among bits, such a sequence may be conveniently viewed as a binary tree (for reasons that will become obvious later on, let's call it $DMT_0$ from Dynamic Markov Tree of order zero) where nodes at level $j$ correspond to bit $j$ ($1 \leq j \leq k$) in the original sequence; each edge that emerges from a node is labelled with a positive count (and therefore with a positive probability) that indicates how many times the substring from the root to that particular node, occurred in the original sequence. For clarity, let's consider the following example.

**Example 1**: For the following 4-bit sequence consisting of 8 non-distinct vectors: $(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8) = (0000, 0001, 1001, 1100, 1001, 1100, 1001, 1100)$ the construction of the tree $DMT_0$ is shown step-by-step in Fig.2a. Obviously, the whole Markov tree that models this sequence must have four levels because the original sequence is a 4-bit sequence. Without any loss in generality, we assume a left-to-right order among bits that is, the leftmost bit in any vector $v_1$ to $v_8$ is considered as being bit number one (and consequently represented at level one in $DMT_0$ as shown in Fig.2a), the next bit is considered as being bit number two and so on. Every time when a vector is completely scanned (that corresponds to reaching the level four in the tree), we come back to the root and start again with the next vector in the sequence. While the input sequence is scanned, the actual counts on the edges are dynamically updated (as shown in Fig.2a for the first three vectors) such that, for this particular example, they finally become as indicated in Fig.2b. The Markov tree in Fig.2b contains in a compact form all the spatial information about the original sequence $v_1, v_2, ..., v_8$. We point out that this sparse structure is possible only by using the dynamic (adaptive) fashion of growing the tree $DMT_0$ just illustrated. Another approach would have been to consider a static binary tree capable to model any 4-bit sequence and just to update the counts on the edges while scanning the original sequence (Fig.3). By doing so, we would end up with the obvious disadvantage of having 15 instead of 9 nodes in the structure for the same amount of information; this reason alone is sufficient for considering from now on only dynamically grown models.
Definition 1. We define the information source, to be the pair $<S,P>$, where $P$ is a function from $S$ into $[0,1]$ satisfying the condition:

$$P(v) = \sum_{x \in S} P(vx)$$

(1)

for all $v$ in $S$, where $vx$ represents the event corresponding to the joint occurrence of the strings $v$ and $x$.

The above condition, simply states that the sum of the counts attached to the immediate successors of node $v$ equals its own value $P(v)$. As we can easily see in Fig. 2, condition (1) is satisfied at every node in this representation. In addition, based on the counts of the terminal edges, we may easily compute the probability of occurrence for a particular vector in the sequence. For instance, the probability of occurrence for string ‘1001’ is $3/8$ (because the count on the terminal edge that corresponds to ‘1001’ is 3 and the length of the sequence is 8) while the probability of string ‘1111’ is zero, ‘1111’ being a ‘forbidden’ vector for this particular sequence.

Traditionally, data compression algorithms (e.g. Huffman, Ziv-Lempel, Cleary-Witten) [16] have been targeted to work at byte- or word-level. In contrast, DMC is normally used to process one bit of the input at a time rather than one symbol at a time. In principle, there is no reason why a symbol-oriented version of the method can not be used. However, for practical reasons (less storage required and more efficiency in computations), the bit version of DMC is preferred.

1. This is actually similar to Kirchoff’s law for currents.
3. Power-oriented data compaction

In this section, first we review the main concerns about preserving the input statistics to obtain accurate power estimates. Second, we present a new approach based on DMC by generalizing the model introduced in Section 2.

3.1 Problem formulation

Input pattern dependence has a dramatic impact on power dissipation estimates. If one ignores the input statistics (which give the actual correlations among the primary inputs), power estimation results can be seriously impaired. Capturing only signal probabilities at the primary inputs of the circuit is not enough for accurate estimates therefore, for power estimation purposes, it is critical to distinguish between sequences which exhibit the same signal probabilities on different bit lines, yet showing very different spatial and temporal correlations.

Assuming that a gate level implementation is available, to estimate the total power dissipation, one can sum over all the gates in the circuit the average power dissipation due to the capacitive switching currents, that is:

\[ P_{\text{avg}} = \frac{f_{\text{clk}}}{2} \cdot V_{DD}^2 \cdot \sum_n (C_n \cdot \bar{s}w_n) \]

where \( f_{\text{clk}} \) is the clock frequency, \( V_{DD} \) is the supply voltage, \( C_n \) and \( \bar{s}w_n \) are the capacitance and the average switching activity of gate \( n \), respectively. From here, the average switching activity per node (gate) is the key parameter that needs to be correctly determined, mostly if we are interested in a node-by-node basis power estimation. However, this parameter is highly sensitive to the input statistics, namely it depends significantly on transition and conditional probabilities among different signal lines.

Having these issues in mind, the vector compaction problem can be formulated as follows: for a \( k \)-bit sequence of length \( n \) (consisting of vectors \( v_1, v_2, \ldots, v_n \)), find another sequence of length \( m < n \) (consisting of the subset \( u_1, u_2, \ldots, u_m \) of the initial sequence), such that the average transition probability on the primary inputs is preserved wordwise. More formally, for any generic input \( v \) and \( u \) (seen as a collection of bits) in the original and in the compacted sequence, respectively, the following holds:

\[ \left| P(v^- = X \land v^+ = Y) - P(u^- = X \land u^+ = Y) \right| < \varepsilon \]

(2)

In relation (2), \( v^- \), \( v^+ \) (\( u^- \), \( u^+ \)) denote the current and the next vector, respectively, in the original (compacted) sequence and \( X \), \( Y \) are any two patterns that appear in the initial sequence. This condition simply requires that the joint transition probability for any group of bits is preserved within a given level of error.

3.2 A DMC-based approach

An attempt to solve the vector compaction problem for power estimation was recently presented in [14]. In that paper, the authors use elements from probabilistic automata theory to synthesize stochastic machines which can be used in a standalone mode for sequence compaction. From a practical point of view, however, this approach has two inherent limitations:

- The values in the initial transition matrix themselves are important in the decomposition process: some distributions of transition probabilities tend to favor a small number of degenerate matrices, as opposed to others which result in much longer decompositions. In these cases, the decomposition becomes the critical step as far as running time is
concerned and one should therefore allow limited precision in the calculations to simplify the decomposition process.

- The compaction technique on stochastic machines is a multiple-step compaction technique. An initial pass through the sequence is performed to extract the statistics of interest; after that, the stochastic machine is synthesized and then the new sequence is generated. This is especially disadvantageous for large sequences when the on-line computer memory and time requirements become prohibitive.

The disadvantages mentioned above can be eliminated by using DMC modeling. To this end, in what follows we introduce an original framework for power-oriented data compaction.

From Section 3.1, it follows that the spatiotemporal correlations that characterize a particular sequence are the key factor in power estimation. Differently stated, not only a particular vector \( v_i \) in a given sequence is important, but also its relative position in that sequence matters. More precisely, different interleavings among the vectors belonging to the same initial set \( (v_1, v_2, ..., v_n) \) (e.g. \( (v_1, ..., v_i, v_j, v_k, ..., v_n) \) or \( (v_1, ..., v_k, v_i, v_j, ..., v_n) \)) define completely different input sequences. Coming back to the model presented in Section 2, we observe that \( DMT_0 \) alone cannot capture this property; we say that \( DMT_0 \) has no memory and therefore the relative order of vectors in the initial sequence is irrelevant for \( DMT_0 \)'s construction. Obviously, \( DMT_0 \) is a poor structure for \( S \) because it does not preserve properly the order of events. In Fig.2b for instance, the value of \( \frac{3}{8} \) is the probability to see the particular string (state) ‘1001’ in the original sequence but this gives us no indication at all about the sequencing of this vector relative to another one, say ‘0001’.

To solve properly the compaction problem, we refine now the above structure by incorporating in it first-order memory effects. Specifically, we consider a more intricate structure, namely a tree called \( DMT_1 \) (Dynamic Markov Tree of order 1), where from the node representing any vector \( v \) there is an emergent arc to each value \( x \) connecting \( v \) to the successor node, associated with the string \( vx \).

**Example 2:** For the same sequence in Example 1, suppose we want to construct its corresponding tree \( DMT_1 \). We begin as in \( DMT_0 \) and for each leaf that represents a valid combination in the original sequence, we construct a new tree (having the same depth as \( DMT_0 \)) which is meant to preserve the context in which the next combination occurs. For instance, the vector \( v_2 = 0001 \) follows immediately after \( v_1 = 0000 \); consequently when we reach the node that corresponds to \( v_1 \) (the leftmost path in Fig.4a), instead of going back to the root (and therefore ‘forgetting’ the context), we start to build a new tree (rooted at the current leave of \( DMT_0 \)) as indicated in Fig.4a. Basically, we added a new path which corresponds to ‘0001’. The newly constructed tree will preserve the context in which \( v_2 = 0001 \) occurred that is, immediately after \( v_1 = 0000 \) (denoted by \( v_1 \rightarrow v_2 \)). After processing the pair \( (v_1, v_2) \), we come back to the root and continue with \( (v_2, v_3) \) as shown in Fig.4b; \( v_2 \) alone leads us to the second leftmost edge of \( DMT_0 \) from where, to construct \( DMT_1 \), we have to add the path ‘1001’ which corresponds to \( v_3 \). In this way, we indicate the sequencing between \( v_2 \) and \( v_3 \) that is, \( v_2 \rightarrow v_3 \).
What is important to note here, is that all vectors in the original sequence are processed that is, none of them is skipped during the construction of $DMT_1$. This is the theoretical basis for accurate modeling of the input sequences as first-order Markov sources of information. Similarly, continuing this process for all leaves in $DMT_0$ in Fig. 2b, we end up by building the whole tree $DMT_1$ as shown in Fig. 5.

In Fig. 5, we separated by a dashed line the two subtrees that constitute $DMT_1$. The upper subtree (levels 1 to 4) represents $DMT_0$, that is, it sets up the state probabilities for the sequence; the lower subtrees (levels 5 to 8), give the actual sequencing between any two successive vectors. To keep the counts in these subtrees consistent, while we traverse the lower subtrees and update the counts on their edges, we also accordingly increment the counts on the paths in the upper subtree (in fact, all vectors except the first and the last are processed exactly twice, once in the upper $DMT_0$ and next in the lower $DMT_0$). In practice the counts of these two subtrees may differ by one, due to the finite length of the sequences. This aspect can be seen in Fig. 5 at level 5 on the rightmost path at the border between
the upper and lower subtree. A practical solution to this issue is to consider the input sequence as being cyclic that is, to link the last combination in the sequence with the very first one or simply add two dummy states, one before and one after the initial sequence.

Obviously, $DMT_1$ provides more information than $DMT_0$. To give an example, string ‘1001’ can follow only after ‘0001’ or ‘1100’, information that cannot be gathered by analyzing $DMT_0$ alone.

**Proposition 1** [19]. We write the probability of a vector string $v = v_1v_2...v_n$ as follows:

$$P(v) = P(v_1) \cdot P(v_2 | v_1) \cdot ... \cdot P(v_n | v_1v_2...v_{n-1})$$  \hspace{1cm} (4)

where the conditional probabilities are uniquely defined by: $P(x|y) = P(xy)/P(y)$.

This property, used in connection with the counts on the edges, allows a quick calculation of the transitions probabilities that characterize a particular sequence. For example, if we want to calculate the transition probability ‘1001’ $\rightarrow$ ‘1100’ we have from Proposition 1 $P(v) = P(v_1v_2) = P(v_1) \cdot P(v_2 | v_1) = 3/8$ which is exactly the count on the path ‘10011100’ in the tree $DMT_1$ divided by the sequence length.

**Theorem 2.** Any sequence in $S$ can be modeled as a first-order Markov source using the structure $DMT_1$ and parameters $P$. We call this process Dynamic Markov Chain (DMC) modeling.

**Sketch of proof:** If $v = v_1v_2$ is a string in the structure $DMT_1$ such that $v_1$ is in the upper tree and $v_2$ is in the lower tree, then $P(v_2 | v_1) = P(v) / P(v_1)$. Thus, the parameters stored on the edges of $DMT_1$ structure provide the conditional probabilities that characterize the lag-one Markov chain for the sequence in $S$. \[\blacksquare\]

**Theorem 3.** The structure $DMT_1$ and parameters $P$ are equivalent to a stochastic sequential machine.

**Sketch of proof:** $DMT_1$ defines a Markov source (based on Theorem 2). Any Markov source is characterized by a stochastic matrix $A$. According to the decomposition Theorem 1 given in [14], this matrix is uniquely associated to a stochastic machine (a finite-state machine with randomly generated inputs). Thus, $DMT_1$ is equivalent to a SSM. \[\blacksquare\]

Generally speaking, the theory of stochastic sequential machines is far more developed than the theory of DMC modeling. However, the DMC modeling technique based on $DMT_1$ seems to be more effective as it offers a much more compact structure and generally outperforms the compaction techniques based on stochastic machines. Specifically:

- Sequence compaction based on $DMT_1$ avoids the time consuming decomposition process necessary in stochastic machines’ synthesis.
- Using $DMT_1$ one can avoid the need for partitioning the input vectors into groups of bits. Therefore, one can expect to improve the accuracy, especially in those cases when the input patterns are highly correlated.
- $DMT_1$ is constructed dynamically (new nodes are added only ‘on demand’) therefore it offers a much more compact data structure than matrix $A$ does.

The DMC modeling technique can be successfully used to model such complex spatiotemporal correlations. The structure $DMT_1$ just introduced is general enough to capture completely the correlations among all bits of the same
input vector and also between successive input patterns. Indeed, the recursive construction of $DMT_1$ by considering successive bits in the upper and lower subtrees completely captures the word-level (spatial) correlations for each individual input vector in the original sequence. Furthermore, cascading lower subtrees for each path in the upper subtree, gives the actual sequencing (temporal correlation) between successive input patterns. This model captures completely spatial correlations and first-order temporal correlations. However, it has conceptually no inherent limitation to be further extended to capture temporal dependencies of higher orders. For instance, if we continue to define recursively $DMT_2$ (as a function of $DMT_1$), we can basically capture second-order temporal correlations (Fig.6).

![Diagram](image)

**Fig.6**

**Theorem 4.** The general structure $DMT_n$ and parameters $P$ can model spatiotemporal correlations of order $n$.

**Sketch of proof:** Let $v = v_1 v_2 ... v_n$ be a string in $DMT_n$ (the substring $v_i$ belongs to the $i$-level tree). Using Proposition 1, we have $P(v_n | v_1 v_2 ... v_{n-1}) = P(v_1 v_2 ... v_n) / P(v_1 v_2 ... v_{n-1})$ and thus the lag-$n$ Markov chain characterizing the input can be fully modeled by the $DMT_n$ structure. □

**4. A DMC-based vector compaction procedure**

A practical procedure to construct $DMT_1$ and generate the compacted sequence is given in Fig.7a. During a one-pass traversal of the original sequence (when we extract the bit-level statistics of each individual vector $v_1, v_2, ..., v_n$ and also those statistics that correspond to pairs of consecutive vectors $(v_1, v_2), (v_2, v_3), ..., (v_{n-2}, v_{n-1}), (v_{n-1}, v_n)$) we grow simultaneously the tree $DMT_1$. We continue to grow $DMT_1$ as long as the Markov model is smaller than a user-specified threshold (model_size), otherwise we just generate the new sequence up to that point and discard (flush) the model. A new Markov model is started again and the process is continued up to the end of the original sequence. The generate_seq procedure called by the DMC program is detailed in Fig.7b. Each generation phase is driven by the user-specified compaction parameter ratio that is, in order to generate a total of $m = n/ratio$ vectors, we have to keep
the same compaction ratio for every dynamically grown Markov model.

\begin{minipage}{0.48\textwidth}
\begin{verbatim}
procedure DMC (input_file, ratio, model_size) {
    initial_state = new_state ();
    symbol = read_input (input_file);
    update_tree (symbol, upper_tree, initial_state);
    crt_state = last_state (symbol, upper_tree);
    while (EOF (input_file)) {
        symbol = read_input (input_file);
        if (number_of_states < model_size) {
            update_tree (symbol, lower_trees, crt_state);
            update_tree (symbol, upper_tree, initial_state);
            crt_state = last_state (symbol, upper_tree);
        } else {
            generate_seq (upper_tree, lower_trees, ratio);
            flush_model (upper_tree, lower_trees);
            initial_state = new_state ();
            symbol = read_input (input_file);
            update_tree (symbol, upper_tree, initial_state);
            crt_state = last_state (symbol, upper_tree);
        }
    }
    generate_seq (upper_tree, lower_trees, ratio);
}
\end{verbatim}
\end{minipage}\hspace{1em}
\begin{minipage}{0.48\textwidth}
\begin{verbatim}
procedure generate_seq (upper_tree, lower_trees, ratio) {
    crt_symbol = generate_random ();
    lower_tree_node = last_node (upper_tree, crt_symbol);
    upper_tree_node = root (upper_tree);
    do {
        for each bit in the current vector {
            generate '0' or '1' to maximize the decrease in absolute error;
            if ('0' is generated) {
                lower_tree_node = left (lower_tree_node);
                upper_tree_node = left (upper_tree_node);
            } else {
                lower_tree_node = right (lower_tree_node);
                upper_tree_node = right (upper_tree_node);
            }
        }
        lower_tree_node = upper_tree_node;
        upper_tree_node = root (upper_tree);
    } while there are still vectors to be generated;
}
\end{verbatim}
\end{minipage}

Fig.7

In all our experiments we used the DMC modeling technique based on the structure $DMT_1$. We also note that this strategy does not allow 'forbidden' vectors that is, those combinations that did not occur in the original sequence, will not appear in the final compacted sequence either. This is an essential capability needed to avoid 'hang-up' ('forbidden') states of the circuit during simulation process for power estimation.

Example 3: Assume that we are given the following 3-bit sequence consisting of 17 non-distinct vectors: $(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{11}, v_{12}, v_{13}, v_{14}, v_{15}, v_{16}, v_{17}) = (001, 100, 001, 110, 111, 111, 101, 110, 011, 000, 101, 001, 100, 000, 110, 110, 011)$; our objective is to compact this sequence with a compaction ratio of 2.

We start building the Markov model that characterizes the initial sequence. For clarity, the construction of the tree $DMT_1$ is shown in Figs.8-9 for two different scenarios. First, in Fig.8, we assume that the parameter model_size is set by the user to the value 35; this means that the model can be grown dynamically (without any need for flushing) until this limit is reached.
Once we built the Markov tree in Fig.8, we start the procedure \textit{generate\_seq} with parameter \textit{ratio} = 2 and generate a subset of 8 vectors which best approximate the original sequence. To this effect, we use a modified version of the \textit{dynamic weighted selection algorithm} [20]. In that approach, a similar structure with DMT\textsubscript{0} is built; more precisely, a full tree having on the leaves the symbols that need to be generated. The counts on the edges are dynamically updated and the symbols are generated according to their probability distribution. For this, a single random number generator is required in order to divide the interval [0, 1] into subintervals that correspond to symbols’ probabilities. At each level, the random number is compared to the left probability: if lower, a zero value is generated; if greater, a one value is generated and the number is decreased by the left probability. In our case, this strategy is used only to generate the first vector. After that, to ensure a minimal level of error, we use an \textit{error controlling mechanism} in a greedy fashion. More precisely, at each level in the lower Markov tree, in order to decide whether a zero or one has to be generated, we compute the transition probabilities for both alternatives and choose the one that minimizes the absolute error accumulated up to that point. Simultaneously, the upper tree is parsed from the root to the leaves, according to the bits generated in the lower subtree. The procedure is then resumed until the needed number of vectors is generated.

In our example, if we assume that \(x = 0.23\) is the first randomly generated number, based on the tree in Fig.8, since \(0.23 < 7/17\) we take the left edge, generate a value 0 and \(x\) remains unchanged. At the second level, \(x = 0.23 < 5/17\) so again we generate a ‘0’ and leave \(x\) unchanged. Now \(x = 0.23 > 2/17\) so a ‘1’ is generated and \(x\) becomes \(x = 0.23 - 2/17 = 0.11\). For the lower subtree rooted at the node denoted by the vector ‘001’ (that is, we parse the upper subtree according to the already generated bits 0, 0, 1), to produce the second vector, we use the error controlling mechanism. Specifically, at node \(N_1\) in Fig.8, the only choice is to take the right edge, generating a ‘1’. Next, at node \(N_2\), the absolute error made for the transition probabilities becomes \(|2/17 - 1/8| + 11/17 - 01| = 0.066\) if we take the left edge, and \(|2/17 - 01 + 11/17 - 1/8| = 0.183\) if we take the right edge (8 is the length of the sequence to be obtained). The first choice is preferred and therefore a ‘0’ is generated. At the last level, at node \(N_3\), the decision is quite simple as we have only one descendent. Thus, after the first vector ‘001’, we generate ‘101’ as the second vector. The generation procedure continues for the lower subtree rooted at the node denoted by the vector ‘101’ until the desired length \(m = \text{...}\)
\( n/r \) is achieved. Despite its locality, this decision strategy performs very well in practice; as we'll see in the experimental part, the overall level of error is very small in all practical cases.

In the second scenario, illustrated in Fig. 9, the \textit{model size} parameter is set by the user as being 30 therefore the tree in Scenario 1 cannot be grown as such because the limit of 30 nodes is reached before the whole sequence is scanned. As a consequence, once we reach this limit (this actually happens immediately after processing the subsequence \( v_1, v_2, ..., v_9 \)), we stop growing the tree and call \textit{generate_seq} procedure with parameter \( ratio = 2 \) (Fig. 9a). This will produce a subsequence of 4 vectors which best approximate the first 'segment' \( (v_1, v_2, ..., v_9) \) of the original sequence. After that we flush the model (keeping only the very last processed vector \( v_9 \)) and start a new Markov tree as shown in Fig. 9b. When the whole sequence is exhausted, based on this new Markov tree, we generate a new subset of 4 vectors which best approximate the second 'segment' \( (v_{10}, v_{11}, ..., v_{17}) \) of the original sequence.

![Diagram](image)

Fig. 9

In general, by alternating the generation and flush phases in the DMC procedure, the complexity of the model can be effectively handled. The issue of accuracy in the context of these repeated flushes is discussed in the subsequent section.

5. Practical considerations

5.1 Complexity related issues

The DMC modeling approach offers the significant advantage of being a \textit{one-pass adaptive technique}. As a one-pass technique, there is no requirement to save the whole sequence in the on-line computer memory. Starting with an
initial empty tree $DMT_1$, while the input sequence is scanned incrementally, both the set of states and the transition probabilities change dynamically making this technique highly adaptive.

Input sequences having a large number of bits $k$ are very common in practice; the success of DMC models for sequence compaction when $k$ is large is based on two key observations:

- The larger the value of $k$ is, the sparser the structure of $DMT_1$ will be.

To motivate this, assume a finite input sequence of length $n$ ($n \ll 2^k$). Intuitively, in a worst-case scenario when $DMT_1$ is completely skewed (that is, all vectors are distinct), $DMT_1$ will have a number of nodes proportional to $2nk$ (in all other cases, due to the sharing of paths among nondistinct vectors, the number of nodes will be smaller). On the other hand, the corresponding full tree (statically constructed) with the same depth, will have a number of nodes proportional with $2^2k$. Therefore the sparsity of the tree $DMT_1$ (compared to the corresponding full tree) will increase with $k$ as: $\text{Sparsity} \approx \frac{2nk}{2^{2k}}$. Assuming for instance an input sequence on 60 bits having a length of 100,000 vectors, then the sparsity of $DMT_1$ is about $10^{-29}$. The DMC modeling technique exploits this observation by starting with an initially empty model and dynamically growing the Markov tree that characterizes the input sequence. By doing so, one can expect to build much smaller trees than the ones otherwise obtained by using a static model based on an initial full tree. Indeed, in practice the dynamic growing of the Markov model performs very well and the experimental results presented in the next section will support this claim.

- Biased sequences which usually occurs in practice as candidates for power estimation, contain a relatively small number of distinct patterns which arise in many different contexts in the whole sequence therefore a probabilistic model is ideally suited for modeling them.

We point out that both these observations can be efficiently exploited only by a probabilistic technique such as DMC modeling; a deterministic technique (e.g. [13]) has no such inherent capability and therefore cannot avoid all the difficulties that arise from this type of complexity.

However, a natural question still remains: when should this growing process be halted? If it is not halted, there is no bound on the amount of memory needed. On the other side, if it is completely halted we lose the ability to adapt if some characteristics of the source message change. A practical solution is to set a limit on the number of states in the DMC [17] as we actually did in Example 3. When this limit is reached, the Markov model is flushed and a new model is started. Although this solution may appear as too drastic, in practice it performs very well. The intuition behind this property is the capability of DMC model to adapt very fast to changes that occur while the input is scanned. A less extreme solution to limit model growing is also possible; we can keep a backup buffer that retains the last $p$ vectors emitted by the source and whenever the model should be discarded, we may reuse this information to avoid starting the new model from the scratch.
5.2 Accuracy related issues

To see how the flushing technique affects the accuracy, let’s assume that an input sequence of length \( n \) is modeled by the DMC approach. Suppose that during the building of the Markov model, flushing occurs after the first \( n_1 \) vectors, then after the next \( n_2 \) vectors, and so on. If the number of flushes is \( f \), then \( n_1 + n_2 + \ldots + n_f = n \). Let \( v_i (u_i) \) be a vector from the initial (compacted) \( i \)-th subsequence (obtained due to successive flushes) and \( v \) \(( u \) \) a vector from the initial (compacted) sequence. We note that:

\[
P(v^- = X \land v^+ = Y) = \frac{\sum_{i=1}^{f} n_i \cdot P(v_i^- = X \land v_i^+ = Y)}{n} \text{ and}
\]

\[
P(u^- = X \land u^+ = Y) = \frac{\sum_{i=1}^{f} \frac{n_i}{r} \cdot P(u_i^- = X \land u_i^+ = Y)}{\frac{n}{r}} \text{ where } r \text{ is the compaction ratio.}
\]

If the \( i \)-th subsequence is approximated with an error less than \( \varepsilon_i \), then the accuracy for the whole sequence is

\[
\varepsilon = \sum_{i=1}^{f} \frac{n_i \cdot \varepsilon_i}{n} \leq \max(\varepsilon_i).
\]

Therefore, as long as the partial DMC models accurately the transition probabilities for the initial subsequences, the transition probabilities for the entire sequence are preserved up to some \( \varepsilon \). Differently stated, we do not have to worry about the number of flushes needed to manage complexity if the individual Markov models capture accurately the characteristics of the subsequences.

An extreme case may occur when all patterns in the original sequence are distinct that is, there are no two identical vectors in the whole sequence. In such a case all terminal edges in the upper subtree of \( DMT_1 \) will have only one successor in the lower subtree, therefore the probabilistic choices during the generation phase will degenerate into a pure deterministic generation of a subsequence from the initial sequence. To avoid this, we propose to use a random pairwise generation phase that is, to select randomly the first vector in the upper subtree of \( DMT_1 \) and generate deterministically the second vector from the lower subtree of \( DMT_1 \). An alternative solution would be to partition the initial sequence into disjoint groups of bits and apply the DMC modeling technique to each individual group separately. As partitioning criteria we may use either functional considerations (e.g. separation between control bits and data-path bits) or the level of correlations among individual bits as was originally proposed in [14]. By partitioning, we increase the chances for repeated patterns in each group of bits therefore making probabilistic arguments more reasonable. The disadvantage however, is that this solution will ignore correlations across group boundaries.
6. Experimental results

The overall strategy is depicted in Fig.10.

![Diagram showing the process of experimental results]

Fig.10

Basically, we verified our ability to compact large input sequences which may also be used as power benchmarks in the design process. We assume that the input data is given in the form of a sequence of binary vectors. While this is a valid assumption at the logic level, it requires some justification at the architectural level. An instruction stream at the architectural level can be converted to a binary stream using the information about opcodes and dynamic instruction traces that resolve memory references and ambiguities. The obtained binary stream can be then subjected to any compaction technique developed for bit-level specifications.

Starting with an \( k \)-bit input sequence of length \( n \), we perform a one-pass traversal of the original sequence and simultaneously build the basic tree \( DMT_1 \); during this process, the frequency counts on \( DMT_1 \)'s edges are dynamically updated.

The next step in Fig.10 does the actual generation of the output sequence (of length \( m \)). As explained in Section 4, to generate the new sequence we use a modified version of the dynamic weighted selection algorithm presented in [20]. If the initial sequence has the length \( n \) and the new generated sequence has the length \( m < n \) then the outcome of this process is a compacted sequence, equivalent to the initial one as far as total power consumption is concerned; we say that a compaction ratio of \( r = n/m \) was achieved.

Finally, a validation step is included in the strategy; for short sequences we used the commercial tool PowerMill [2] whilst for long sequences we resorted to an in-house gate-level logic simulator developed under SIS. The total power consumption of some ISCAS’85 and ISCAS’89 benchmarks has been measured for the initial and the compacted sequences, making it possible to assess the effectiveness of the compaction procedure (under both zero- and real-delay models).

In Tables 1-2, we provide only the real-delay results for two types of initial sequences. Sequences of type 1 are large input streams having the same initial length \( n = 100,000 \) and being then prime candidates for compaction; type 1 refers to biased sequences obtained by doing bit-level logical operations on ordinary pseudorandom sequences. The sequences of type 2 (having the length 4,000) are highly biased sequences obtained from real industry applications. As shown in Table 1, sequences of type 1 were compacted with two different compaction
ratios (namely \( r = 50 \) and 100); we give in this table the total power dissipation measured for the initial sequence (column 3) and for the compacted sequence (columns 4, 5). In the last column, we give the time is seconds (on a Sparc 20 workstation with 64 Mbytes of memory) necessary to read and compress data with DMC modeling. Since the compaction with DMC modeling is linear in the number of nodes in the structure \( DMT_j \), the values reported in the last column are far less than the actual time needed to simulate the whole sequence. During these experiments, the number of states allowed in the Markov model was 20,000 on average.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Number of inputs</th>
<th>Power for initial seq.</th>
<th>Power for ( r = 50 )</th>
<th>Power for ( r = 100 )</th>
<th>Time for DMC (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>36</td>
<td>1816.32</td>
<td>1838.89</td>
<td>1779.60</td>
<td>42</td>
</tr>
<tr>
<td>C499</td>
<td>41</td>
<td>3697.84</td>
<td>3546.65</td>
<td>3622.26</td>
<td>48</td>
</tr>
<tr>
<td>C880</td>
<td>60</td>
<td>3314.07</td>
<td>3229.85</td>
<td>3329.31</td>
<td>75</td>
</tr>
<tr>
<td>C1355</td>
<td>41</td>
<td>3205.27</td>
<td>3044.20</td>
<td>3109.18</td>
<td>48</td>
</tr>
<tr>
<td>C3540</td>
<td>50</td>
<td>10876.22</td>
<td>9910.08</td>
<td>10687.32</td>
<td>61</td>
</tr>
<tr>
<td>C6288</td>
<td>32</td>
<td>110038.69</td>
<td>114199.30</td>
<td>109077.42</td>
<td>37</td>
</tr>
<tr>
<td>s344</td>
<td>9</td>
<td>751.58</td>
<td>748.54</td>
<td>719.53</td>
<td>10</td>
</tr>
<tr>
<td>s386</td>
<td>7</td>
<td>818.11</td>
<td>844.58</td>
<td>848.80</td>
<td>8</td>
</tr>
<tr>
<td>s838</td>
<td>34</td>
<td>1052.05</td>
<td>1061.73</td>
<td>1091.14</td>
<td>41</td>
</tr>
<tr>
<td>s1196</td>
<td>14</td>
<td>3687.47</td>
<td>3702.32</td>
<td>3580.63</td>
<td>16</td>
</tr>
<tr>
<td>s9234</td>
<td>36</td>
<td>9192.75</td>
<td>9157.31</td>
<td>9209.75</td>
<td>43</td>
</tr>
<tr>
<td>Average relative error (%)</td>
<td>2.80</td>
<td>2.93</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As we can see, the quality of results is very good even when the length of the initial sequence is reduced by 2 orders of magnitude. Thus, for C432 in Table 1, instead of simulating 100,000 vectors with an exact power of 1816.32 uW, one can use only 2000 vectors with an estimate of 1838.89 uW or just 1000 vectors with a power consumption estimated as 1779.60 uW. This reduction in the sequence length has a significant impact on speed-up the simulative approaches where the running time is proportional to the length of the sequence which must be simulated.

The sequences of type 2 were compacted for two compaction ratios (\( r = 5 \) and \( r = 10 \)) using PowerMill [2]; to assess the potential of efficiency of the approach, for both original and compacted sequences, we report also the actual running time required by PowerMill to provide power estimates. The number of states allowed for the Markov model construction, was 5,000 on average; the CPU time for DMC modeling was below 3 seconds in all cases.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Number of inputs</th>
<th>Current (mA)</th>
<th>Time to simulate (sec)</th>
<th>Current (mA) ( r = 5 )</th>
<th>Current (mA) ( r = 10 )</th>
<th>Time to simulate (sec) ( r = 10 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>36</td>
<td>0.4135</td>
<td>1186</td>
<td>0.4352</td>
<td>0.4066</td>
<td>120</td>
</tr>
<tr>
<td>C499</td>
<td>41</td>
<td>0.8188</td>
<td>2675</td>
<td>0.8337</td>
<td>0.8290</td>
<td>235</td>
</tr>
<tr>
<td>C880</td>
<td>60</td>
<td>0.7907</td>
<td>2289</td>
<td>0.7995</td>
<td>0.8023</td>
<td>274</td>
</tr>
<tr>
<td>C1355</td>
<td>41</td>
<td>1.1375</td>
<td>2993</td>
<td>1.1549</td>
<td>1.1461</td>
<td>284</td>
</tr>
<tr>
<td>C1908</td>
<td>33</td>
<td>1.2976</td>
<td>4034</td>
<td>1.2821</td>
<td>1.2833</td>
<td>367</td>
</tr>
<tr>
<td>C3540</td>
<td>50</td>
<td>3.4490</td>
<td>9467</td>
<td>3.3582</td>
<td>3.5719</td>
<td>1082</td>
</tr>
<tr>
<td>C6288</td>
<td>32</td>
<td>14.5749</td>
<td>88032</td>
<td>14.8020</td>
<td>13.3095</td>
<td>5005</td>
</tr>
<tr>
<td>Average relative error (%)</td>
<td>2.15</td>
<td>2.63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
As it can be seen in Table 2, the average relative error is below 3% while the speed-up in power estimation is about one order of magnitude on average. For example, using the original sequence of 4000 vectors, PowerMill took for C432 about 1186 seconds to estimate a total current of 0.4135 mA. On the other side, using the sequence generated with DMC of only 400 vectors (r = 10), PowerMill estimated a total current of 0.4066 mA in only 120 seconds. This speed-up in power estimation becomes more significant for larger modules (e.g. C6288). We note also, that the results presented both tables 1 and 2, are significantly better than those reported in [14] in terms of accuracy, running time and memory requirements.

Finally, we compare our results with simple random sampling of vector pairs from the original sequences [21]. In simple random sampling, we performed 1,000 simulation runs with 0.99 confidence level and 5% error level on each circuit. We report in Table 3 the maximum and average number of vector pairs needed for total power values to converge [11]. We also indicate the percentage of error violations for total power values, using as thresholds 5%, 6% and 10%. Using different seeds for the random number generator (and therefore choosing different initial states in the sequence generation phase), we run a set of 1,000 experiments for the DMC technique. In Table 4, we give the DMC results for the same thresholds as those used in simple random sampling.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Max.</th>
<th>Avg.</th>
<th>&gt; 5%</th>
<th>&gt; 6%</th>
<th>&gt; 10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>3300</td>
<td>2176</td>
<td>1.1</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>C499</td>
<td>1500</td>
<td>852</td>
<td>1.4</td>
<td>1.3</td>
<td>0.4</td>
</tr>
<tr>
<td>C880</td>
<td>3990</td>
<td>2795</td>
<td>1.8</td>
<td>0.4</td>
<td>0.7</td>
</tr>
<tr>
<td>C1355</td>
<td>1380</td>
<td>814</td>
<td>1.7</td>
<td>1.0</td>
<td>0.2</td>
</tr>
<tr>
<td>C1908</td>
<td>1620</td>
<td>846</td>
<td>1.9</td>
<td>1.3</td>
<td>0.2</td>
</tr>
<tr>
<td>C6288</td>
<td>7470</td>
<td>5422</td>
<td>1.4</td>
<td>1.4</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Number of vectors</th>
<th>&gt; 5%</th>
<th>&gt; 6%</th>
<th>&gt; 10%</th>
</tr>
</thead>
<tbody>
<tr>
<td>C432</td>
<td>2000</td>
<td>6.7</td>
<td>1.9</td>
<td>0.0</td>
</tr>
<tr>
<td>C499</td>
<td>800</td>
<td>0.3</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>C880</td>
<td>2000</td>
<td>1.4</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>C1355</td>
<td>800</td>
<td>0.2</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>C1908</td>
<td>800</td>
<td>1.9</td>
<td>1.2</td>
<td>0.0</td>
</tr>
<tr>
<td>C6288</td>
<td>2000</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Once again, the results obtained with DMC modeling technique score very well and prove the robustness of the present approach. As we can see, using fewer vectors, the accuracy of DMC is higher than the one of simple random sampling in most of the cases. Noteworthy examples are benchmarks C499, C880, C6288 where less than 60% of the maximal number of vector pairs needed in random sampling for convergence are sufficient for DMC to achieve higher accuracy and confidence levels.

7. Conclusion

In this paper, we addressed the vector compaction problem from a probabilistic point of view. Based on dynamic Markov Chain modeling, we proposed an original approach to compact an original sequence into a much shorter equivalent one, which can be used after that with any available simulator to derive power estimates in the target circuit.

The mathematical foundation of this approach relies in Markov models; within this framework a family of

1. This means that the probability of having a relative error larger than 5% is only 1%.
dynamic Markov trees is introduced and characterized as an effective and flexible way to model complex spatiotemporal correlations which occur during power estimation. The results obtained both on combinational and sequential benchmarks show that large compaction ratios of 1-2 orders of magnitude can be obtained without much loss in accuracy in total power estimates.

The issues brought into attention on this paper represent an important step to reduce the gap between simulative and nonsimulative techniques which are currently the norm.

References