Levenberg-Marquardt Backpropagation model augmented with Prim’s algorithm approach (LMBP) to minimize power in FSM synthesis

Kaushik Khatua
Electronics and Electrical Communication Eng. Dept.
Indian Institute of Technology, Kharagpur
Kharagpur, India
kaushik.khatua31@gmail.com

Abstract—Recent advancements in intelligent learning enhance the logic synthesis domain, especially for low power minimization. This paper presents a low power synthesis by implementing AI with conventional greedy Prim’s algorithm. Initially, by applying Prim’s algorithm, an MST (Minimum Spanning Tree) has been formed from the undirected state transition graph, which has been obtained from a conventional FSM benchmark with the help of a probability method. After finding the MST, the minimum power is found from a weighted Hamming distance by state encoding from the SIS (synthesis for sequential circuit) tool, which is an NP-Hard problem and thus time-consuming. This motivates us to apply the AI integrated technique with the greedy algorithm. The database has been initially generated, and with that Levenberg-Marquardt backpropagation model is trained to replace the SIS tool to predict the power for multi-level realizations. The results are impressive, and the model is even more efficient for prediction. Experimental results can show that the augmented-based framework speed up the process by two times compared to the basic simulation-based process.

Index Terms—Power Consumption, Prim’s Algorithm, Levenberg-Marquardt, Backpropagation Model

I. INTRODUCTION

Low power drive circuit design is an important aspect with longer battery life and high performance in portable devices. For keeping a stringent budget in the cooling system, power dissipation should be handled in flexible ways. Several Artificial Intelligence (AI) algorithms have been augmented with other heuristics to get benefits for optimizing power, area, delay, and testability. AI has been proven a powerful and popular tool over the last decade and has a great influence on logic synthesis design, which helps to detect the fault and minimize time. For tabular data as an input to the training, the algorithm is sometimes not sufficient enough to produce accuracy, thus requiring extensive features. We are targeting to search the optimized solution in the low power testing domain, which took an interest in augmenting the heuristic with an AI-based framework. Several probabilistic and deterministic methods have been proposed to optimize such problems. All of them used several deterministic tools like NOVA, JEDI [1], but none of them considered the time constraint part. Here AI comes to the rescue of such things. We have proposed a framework that replaces the conventional tool to predict the power without affecting the tabular inputs.

In this paper, Prim’s algorithm applied on a weighted undirected transition graph, and a parallel database has been generated in order to train the NN model [2] to predict the optimized power within a small run-time. The data set is trained with the Lavenberg-Marquardt (LM) ([3], [4]) back propagation algorithm to minimize the desired power consumption. The motivation behind choosing this algorithm is the faster convergence rate and gives stable output. The computational complexity is a combination of Newton’s method and Gradient Descent which has a faster accuracy and lower error rate.

Some probabilistic methods like Tabu search (TS) and Probabilistic Tabu Search (PTS) [5] have been proposed, which is based on code swapping probability where lower cost is assigned to higher probabilities and higher cost to lower probabilities. This is partially based on hill-climbing behavior. Some evolutionary strategy like the majority-based algorithm (MBE) and multi-population evolutionary strategy (MPES) [6] has also been proposed, which operate on offspring populations. Some conventional method like a genetic algorithm (GA) [7] and particle swarm optimization (PSO) was also used, which is based on a partitioning strategy where the high transition from one state to another state is calculated by weighted hamming distance.

II. POWER COMPUTATION

It is very hard to estimate the exact power dissipation in a combinational circuit even after exact state encoding. A different form of technology was used to reduce power, but in gate-level implementation, the exact computation of dynamic power is not possible to find as glitches in the circuit are always not traceable from the input test pattern. The switching activity reduction approach is quite popular and
attracted the researcher for computing average dynamic power consumption, which can be described as:

\[ P = \frac{1}{2} \cdot C_L V_{DD}^2 E_{sw} f_{clk} \]  

(1)

Here \( C_L \) represents the node capacitance, \( V_{DD}^2 \) the supply voltage, \( f_{clk} \) the frequency of operation, and \( E_{sw} \) the switching activity of the node. Since from the equation the exact capacitive switching is not possible to calculate due to complex fan-out in the circuit so expected switching activity \( E_{sw} \) is possible to find by using weighted hamming distance formulae:

\[ \sum_{S_i, S_j \subseteq S} t_{p_{i,j}} H(S_i, S_j) \]  

(2)

The hamming distance from \( S_i \) to \( S_j \) and state transition probability is represented by \( H(S_i, S_j) \) and \( t_{p_{i,j}} \). The encoding strategy is applied by a Minimum Weighted Hamming Distance (MWHD) [8]. The state assignment with STG partitioning helps reduce switching activity \( E(sw) \). Using Chapman Kolmogorov equations, the state transition probability can be calculated from any state transition graph. The directed STG can be transformed into undirected graph for calculating transition probability. Assigning binary state codes to different states, which is represented as no. of nodes in the graph to find minimum weighted Hamming distance, thus minimum dynamic power consumption. Finding such exact state encoding is an NP-hard problem. The greedy algorithm is a popular optimization technique to find such NP-hard problems, and Prim is one of them to solve such.

Next, in Fig. 1(b), the STG is converted weighted directed graph by eliminating the self-loop and calculating the state transition probability and weighted transition matrix using the Chapman-Kolmogrov equation. The probability matrix is obtained from the Markov-chain probabilistic model. In Fig. 1(c), the directed STG has been converted into undirected graph for applying Prim’s algorithm. From this undirected graph Minimum Spanning Tree (MST) will be formed where all the vertices will be covered with minimum weight cost. As it’s a greedy algorithm so, for larger benchmarks circuit, it will not give a guarantee to visit all the nodes and branches. The algorithm forms the closest neighbor node with respect to the undirected state transition graph. Prim’s algorithm depends on the starting node. Here we run the Prim’s algorithm for all the nodes. So basically it will cover the whole graph and find the all MST with possible WHD. Therefore, we can find a total of N different solutions for N different nodes. For developing the algorithm with N nodes, it is needed to estimate \((N^2 - N)/2\) edge weight. The Prim’s algorithm should run N times to estimate the exact solution. For N number of runs, the power estimator should be run for \((N^2 - N)/2\) times. This forces a higher run-time overhead.

A. Initial framework

In order to reduce the run-time, neural network comes to the rescue where the predicted model will estimate the power without the simulator. The database, which is extracted from Prim’s algorithm, is prepared where the weighted transition matrix (WTM), state encoding, fanin, and fanout is considered as one of the input feature columns and power dissipation is considered as an output column. For bypassing repeating estimation in the feature matrix, we have included the switching transition matrix \((NXN)\) where diagonal elements are filled with zeros. For different state encoding and switching transitions, the power dissipation caused by equation 1 will be different and time-consuming. NN is capable of predicting the power from the trained database. Figure 2 shows the process flow of the overall framework.

IV. LEVENBERG-MARQUARDT ALGORITHM

It is one of the popular algorithms which tries to minimize the mean squared error (MSE). This is one of the fastest methods but requires high memory. The derivation of the algorithm consists of Steepest Descent and Newton’s algorithm. The Steepest Descent algorithm calculates the first-order derivative to find the local minima in error space where the training process is asymptotic convergence. The second-order derivatives of the error function are calculated in Newton’s method. A scalar quantity \( \mu \) is introduced with a lower value with the gradient decent method, and it gets a higher value to make Newton’s algorithm faster in calculating the low error. So for the neural network implementation, the above problems are to be solved. The training process stops when the maximum no. of Epochs is reached, and \( \mu \) exceeds a preset value.

III. PRIM’S ALGORITHM

Here we have considered the initial framework by applying Prim’s algorithm [9] to an example benchmark shown in Figure 1 (a). Any state transition graph can be represented as a number of nodes (N) and edges (W). The directed graph can be converted to undirected graph probabilistic model Markov Process [10] where the transition probability can be calculated for switching transition. We have considered an STG in Fig. 1(a) with four states where input and output transition is given.

Fig. 1. (a) An Example State Transition Graph (STG), (b) Directed graph with state transition probability, (c) Undirected graph with edge weight
A. Back propagation Model

It is essential to generate the database to train the NN for replacing the SIS tool. The generated database can be obtained by running the Prim’s algorithm-driven by SIS. There is a trade-off between CPU time and database generation. If there is an incomplete database, then the ANN will be unable to train the predicted solution. In contrast, the CPU-time is needed to be higher to generate a larger database. The back propagation model is used to calculate the Jacobian matrix of the performance function to adjust the weight and bias, which is being calibrated by the Levenberg-Marquardt Algorithm. The ANN is well trained by the database for the well-trained model.

The overall input is given through some activation function. For training the back propagation model, we have used a feed forward network with one input layer, one or more hidden layers and one output layer. The FFNN with one hidden layer is built in this study because a neural network with one hidden layer can handle the majority of difficult functions. The FFNN with one hidden layer is built in this study because a neural network with one hidden layer can handle the majority of difficult functions. The back propagation model is used to minimize the mean square error by adjusting different parameters by the steepest descent algorithm and learning rate. In order to train the back propagation network to process the mean square error less than 5% which is having hidden layers.

B. NN augmented Prim’s Algorithm

After training the back propagation model with the existing database obtained from Prim’s algorithm, we have used a feed forward network with one input layer, one or more hidden layers and one output layer. The FFNN with one hidden layer is built in this study because a neural network with one hidden layer can handle the majority of difficult functions. The back propagation model is used to minimize the mean square error by adjusting different parameters by the steepest descent algorithm and learning rate. In order to train the back propagation network to process the mean square error less than 5% which is having hidden layers.

V. RESULT

The tool has been coded and executed in python language and runs on a computer with an Intel Core i5-4570 @3.2GHz, DDR3-4GB system configuration. We have used 5-7% sample data from the possible inputs of \(2^n\) for training purposes. So the time taken to generate a database is considered a trivial factor. So from the total database (75%) is used for training purpose, (15%) used for validation and (10%) for testing purpose. Generally, the validation process can change the network parameters till the network is not generalized. The tendency behind this division is to provide as much data for training and validation purposes which is purely experimental. Using the maximum number of data for test and validation purposes is to define the model’s efficiency.

In Table 1, the result has been presented for five complex benchmark circuits. The \(3^{rd}\) and \(4^{th}\) column represents neural network model details as to which circuit needed training samples and layers. We have compared our method with one of the recently published state assignment techniques, Probabilistic Tabu Search [5], and MPES [6]. Out of five benchmarks, two produce the best results where PTS [5], and MPES [6] single results each. The predicted model we trained from the database has produced the best result for \(s1\) benchmark. Column \(9^{th}\) and \(10^{th}\) represent the database generation and neural network training time. It can be seen the training time is short, and the time percentage share is also very less when compared to the conventional Prim’s algorithm-driven with SIS for measuring multi-level power realization. The total NN time has been calculated in \(12^{th}\) column, where the database generation time (t1), training time (t2), and NN augmented Prim’s (t3) are added. For every benchmark, it can be seen that run-time is quite faster compared to all other cases. For planet benchmark the
TABLE I
TEST RESULT OF MULTI-LEVEL POWER OF THE PROPOSED TECHNIQUE ON MCNC BENCHMARK CIRCUITS

<table>
<thead>
<tr>
<th>FSM</th>
<th>I/O/S</th>
<th>Neural Network Details</th>
<th>Power Comparison of Multilevel Analysis in micro Watt (μWatt)</th>
<th>LM Backpropagation Model Augmented With Prim’s (sec)</th>
<th>Run-time Comparison (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Database Generation Time t1 (s)</td>
<td>Training time t2 (s)</td>
<td>Prim’s algo. with NN t3 (s)</td>
</tr>
<tr>
<td>keyb</td>
<td>7/2/19</td>
<td>2500 10</td>
<td>258.4 279.4 262.4 345.2</td>
<td>147.6 0.25 41.65 189.5</td>
<td>245.6 22.26 3144</td>
</tr>
<tr>
<td>planet</td>
<td>7/19/48</td>
<td>6000 30</td>
<td>352.6 348.9 335.2 1459.2</td>
<td>214.3 0.63 112.7 327.63</td>
<td>8.47 6116.9 3941</td>
</tr>
<tr>
<td>s1</td>
<td>8/6/20</td>
<td>3000 10</td>
<td>356.2 345.8 371.3 510.1</td>
<td>125.7 0.24 125.7 245.6</td>
<td>52.4 20332 3138</td>
</tr>
<tr>
<td>s832</td>
<td>18/19/25</td>
<td>3000 20</td>
<td>438.7 445.1 459.4 654.1</td>
<td>189.7 0.37 189.7 302.87</td>
<td>742.6 23821 8050</td>
</tr>
<tr>
<td>sk</td>
<td>6/3/32</td>
<td>6000 20</td>
<td>625.7 614.7 608.3 606.8</td>
<td>250.8 0.24 250.8 375.8</td>
<td>613.5 68538 8972.3</td>
</tr>
</tbody>
</table>

Fig. 3. (a) validation performance (b) Training and validation

run-time is 2.51, 186.53 and 28.51 times faster compared with Prim’s, PTS [5] and MPES [6].

The effect of prediction is tested under a different number of neurons is tested under the range from 8 to 50, so the training time (t2) varies from 0.2 to 0.4 sec, which is shown in table 1. The best prediction of power that has been performed for s1 benchmark is 345.8 μWatt. Fig. 3(a) investigated the mean squared error vs. 20 epochs for the validation performance of planet benchmark. In 14 epochs, it reaches 0.0416 for the train and test data. The regression performance for the planet benchmark is shown in Fig. 3(b). For training, validation, and test data, regression is 0.988, 0.984, and 0.985.

VI. CONCLUSIONS

This experiment’s analysis shows the efficiency and accuracy of the LMBP model. Overall the trained model produces 29.05% lesser power. The overall run-time for the model is 54.81 times faster than the other technique reported in the literature. The NN based framework can achieve the above result with 98.75% accuracy for prediction and 4.025% for maximum error on average.

REFERENCES