Application of Random Walks for the Analysis of Power Grids in Modern VLSI Chips

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Baktash Boghrati

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Sachin S. Sapatnekar, Advisor

July, 2013

© Baktash Boghrati 2013
ALL RIGHTS RESERVED

Acknowledgements

Abstract

Power grid design and analysis is a critical part of modern VLSI chip design and demands tools for accurate modeling and efficient analysis. In this thesis, we develop new solutions for solving power grids, both incrementally and in total, based on an approach that uses random walks.

The process of power grid design is inherently iterative, during which numerous small changes are made to an initial design, either to enhance the design or to fix design constraint violations. Due to the large sizes of power grids in modern chips, updating the solution for these perturbations can be a computationally intensive task. The first issue addressed in the thesis relates to the problem of incremental analysis of power grids. We introduce two incremental solvers that utilize forward and backward random walks to identify the region of influence of the perturbation. The solution of the network is then updated for this significantly smaller region only. Our forward incremental random walk solver is capable of identifying the region of influence efficiently leveraging the record of the forward random walks that is obtained in a preprocessing step performed once for a series of consecutive perturbations. Experimental results show that our forward incremental random walk solver achieves speedups of up to 3× for 20 and up to 2× for 10 consecutive perturbations compared to a full system solution.

An alternative approach, backward incremental random walk solver, identifies the region of influence directly from perturbed system. It can handle consecutive perturbations without any degradation in the quality of the solution. Moreover, this approach is particularly well-suited to a new and more accurate modeling methodology for power grids, introduced in this work, that can result in asymmetrical systems of linear equations. Experimental results show that our backward incremental random walk solver achieves speedups of up to 13× as compared to a full re-solve of the power grid.

Another contribution of this thesis is to introduce a scaled random walk solver that is capable of finding the solution of individual nodes in the power grid much faster than the conventional random walks. This solver uses the notion of importance sampling to reduce the variance of the walks and therefore improve the runtime of the game. Experimental results show that our scaled solver is up to $2\times$ faster than the naïve random walks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The problem of verifying the integrity of a power grid is a critical step in the design of integrated circuits [1–11]. In the steady state (similar equations can be used for transient analysis with time-stepping), this verification corresponds to the solution of a set of equations to determine the voltages in the grid, and ensuring that they are within a certain permissible range. This system of equations can be written as:

$$G\mathbf{V} = \mathbf{E}, \tag{1.1}$$

where $G \in \Re^{N \times N}$ is the left hand-side (LHS) matrix, modeling the conductances, $\mathbf{V} \in \Re^N$ is the vector of unknown node voltages, and $\mathbf{E} \in \Re^N$ is the right hand side (RHS) vector, modeling the current loads. Matrix G is sparse and diagonally dominant ($\sum_{i \neq j} |g_{ij}| \leq g_{ii}, \forall i$), and all off-diagonals of G are less than or equal to zero. In fact, several problems in VLSI design and in other fields involve the solution of a system of similar linear equations where the left hand side has the form of a diagonally dominant matrix with positive diagonal and nonpositive off-diagonal entries, such as thermal analysis under the resistive-electrical duality [12–16], and quadratic placement [17–21]. Mainstream methods for solving such systems include direct methods such as LU/Cholesky factorization [22, 23] and iterative methods [24–26]. Lately, there has been an upsurge of interest in the use of random walk based solvers [27–31] for solving systems with diagonally dominant

LHS matrices, and these solvers are competitive with conventional solvers [32–35]. Random walk based solvers have been proven useful in various applications, instances are: potential theory [31,36–40], solution of Laplace equations [36,37,39], and solution of Poisson equations [13, 38].

This work first addresses a specific problem in the area of steady-state power grid analysis where an incremental change is made in an already-solved network and the new solution is to be determined. Power grid design is a highly iterative process in which the designer makes small perturbations to a complete initial design to fine-tune the design or fix violations in the noise specifications. Examples of perturbations to a power network include changes to the wire conductances (e.g., when the length or thickness of the wires change), power pad placement, or current loads. For a small perturbation the solution of the perturbed system is *close* to the initial solution, meaning that the change in the solution of most of the nodes of the network is insignificant, and the small subset of nodes that are physically or electrically close to the perturbation are most likely to be affected [41, 42]. Examples of small perturbations are: change in power grid structure locally, change in current sources, and change in pad positions.

The notion of closeness of the solution of the perturbed system to the initial solution suggests that for finding the perturbed solution, solving for the entire system from scratch is quite wasteful. To efficiently perform this task, one should leverage the property of "closeness" and utilize the unperturbed solution. However, this is not a trivial task since it is unclear *a priori* which nodes change significantly and which do not. One possible method would be to partition the network, create macromodels for each partition, and solve the problem hierarchically, as in [43]. However, this may require a large number of partitions, involving large amounts of computation. Moreover, the size of the optimal partition may depend on the magnitude of the perturbation. Other approaches employ iterative solvers [44] using the unperturbed solution as an initial guess, sensitivity methods [45], and the fictitious domain method [46]. These suffer from the fact that they require the full system to be solved again, for the entire chip, and do not

fully utilize the "closeness" properties above. An iterative solver is also described in [47], but operates on smaller, denser systems.

An alternative approach is to identify the set of nodes that are significantly affected by the perturbation, called the *region of influence* (RoI). The RoI is the set of all of the nodes in the network for which the voltage changes by more than a given threshold under the applied perturbation changes. Figure 1.1 illustrates a perturbation to power grid and the RoI associated with it. If the RoI could be found, one could, in principle, solve for a drastically smaller system of equations by keeping the solution of the nodes out of RoI at their initial values, and recomputing only the solutions within the RoI. Figure 1.2 schematically illustrates the dimension of LHS matrix corresponding to the full network and the much smaller subsystem corresponding to the RoI of a perturbation.

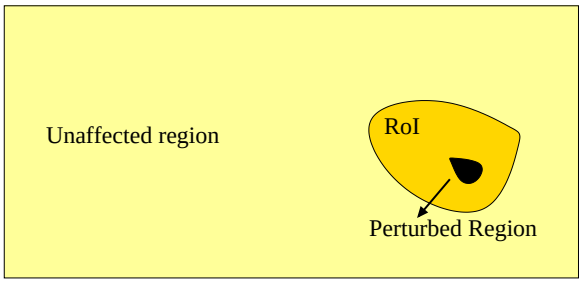


Figure 1.1: A schematic of a design perturbation and its RoI.

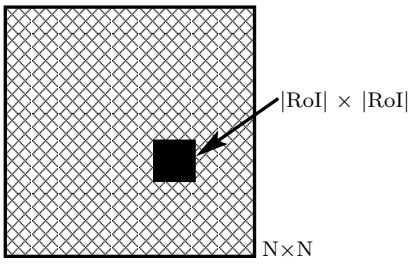


Figure 1.2: Reduction in system size using RoIs.

Identifying the RoI is nontrivial. In this work we introduce two different approaches to identify the RoI. First approach which leverages the *forward random*

walks, uses the bookkeeping information of the random walk solver of [32], obtained as part of a preprocessing phase, to identify the RoI for any perturbation. This approach is specially interesting for the case that the bookkeeping information of the system is previously known. The experimental results show that this approach can compute the RoI efficiently given the bookkeeping information. On the other hand this approach has two drawbacks: (1) it uses approximations that neglect second-order terms, leading to some errors, and (2) it neglects the fact that the bookkeeping information changes due to these perturbations, making this method increasingly inaccurate as more consecutive perturbations are made.

To overcome the shortcomings of the forward incremental random walk solver a second approach is developed that finds the RoI accurately and efficiently using the notion of *backward random walks*. This approach does not require preprocessing and finds the RoI directly.

The basic concept of backward random walks was outlined almost 50 years ago in [31], but with no regard to computational efficiency. The backward random walk method is also known as shooting method in computer graphics, and has been used in the context of radiosity and illumination problems to determine the reflections of a light source on the environment [48]. The shooting method in graphics shares the idea of our approach, finding the affected region due to a source by running random walks from a source, but the problem structure is sufficiently different that the resemblance stops at a very superficial level, and solutions from that domain cannot be adapted easily.

In this work, we develop the theory of backward random walks and show how a solution for the incremental analysis problem can be obtained by applying this method. We employ the backward random walk method to (1) make the approach computationally practical, (2) show a theoretical relation to LU factorization, and (3) apply it to incremental analysis. Our incremental solver can capture any number of consecutive perturbations on the LHS and the RHS of the network of fixed size, without degradation in the accuracy. Experimental results show speedups of up to 13× compared to the Hybrid Solver of [49] on IBM power grid

benchmarks of [50]. The key feature of the backward random walk approach is the ability to find some columns of G^{-1} efficiently, individually, and without incurring the large computational overhead of determining the entire G^{-1} matrix.

Further, we show the relationship between forward and backward random walks. In particular, forward random walks effectively construct G^{-1} by the row, while we show that backward walks construct G^{-1} by the column. Since G^{-1} for a symmetric G is symmetric, this implies that for a symmetric system, the forward and backward walks are equivalent, and our backward random walks can be substituted by the already-known theory of forward random walks. However, we also demonstrate that when a power grid modeled appropriately, the resulting system of equations results in voltage-controlled current sources that can result in an asymmetric G matrix. This is where the theory of backward walks developed in this work has particular applicability, since we will show that incremental analysis requires specific columns of G^{-1} to be reconstructed.

Finally in the last part of this work we consider solving for a single node of the network without solving for the whole system where there is no prior information about the solution of the network. The novelty of this work is in pointing out that even in applying random walks to solve a single node (depending on the analysis scenario, this analysis may be the problem to be solved, or may be a unit problem in the solution of the entire grid), further efficiencies are possible. The solution draws upon some key characteristics of power grids: first, that in any reasonably designed grid that is to be analyzed, the solutions at all nodes are very similar in value, and second, that for the most part, and for DC analysis in particular, the directionality of current loads from the functional blocks is similar, drawing current from the power network (or delivering current into the ground network). In other words, the entries of the right hand side vector, $\mathbf{E}$, are greater than or equal to zero: this is true in practice for DC analysis, and also for AC analysis, even in the presence of package inductances, which are handled as in [49].

In general, any random walk method in its naïve form can be slow, but its performance can be improved by guiding the randomness in a productive direction

[51, 52]. In this work, we develop a heuristic method inspired by Importance Sampling (IS) [53] to speedup the random walk solver, while maintaining the level of accuracy. Importance Sampling is a variance reduction technique that can potentially reduce the variance of the random walk solver to zero[1] , and represents a procedure that can reduce the runtime of the solver.

The idea behind the IS method, as discussed in more detail in Section 6.1.2, is to modify the distribution of the *naïve* random walks so that the metric provided by each walk, referred to as its gain, is approximately the same. The change in the distribution causes a bias in the solution which is compensated for, using a weighted average scheme, and is referred to as *scaling*.

The content of this thesis is organized as follows:

Chapter 2 discusses the procedure of extracting the power grid model and introduces a procedure for more accurately modeling power grids.

Chapter 3 first describes the forward game, how it is constructed and its relation to the rows of the inverse of the LHS matrix of Equation (1.1), G^{-1}. Next the motivation for computing the columns of G^{-1} is presented which is followed by a discussion on backward random walk game that provides an efficient way for computation of the columns of G^{-1} one at a time and without computing the entire G^{-1}. Finally in this chapter the relation of the backward random walks and the LU decomposition is presented.

Chapter 4 describes the incremental solver based on the forward random walks. The experimental results are presented that demonstrate the quality of the found RoI and compares its performance with the hybrid solver of [54].

Chapter 5 presents the incremental solver based on the backward random walks. The performance of the backward solver is demonstrated by comparison to the hybrid solver of [54] and the forward incremental solver of Chapter 4.

[1] For many practical problems, reducing the variance to zero may be unrealistic: it may require the solution of the system to be known, in which case there would be no point in solving the system!

Chapter 6 presents an scaled solver that leverages the notion of importance sampling in forward random walk game to speedup the computation of solution of individual nodes of power grids. Experimental results show significant speedups, as compared to naive random walks used by the state-of-the-art random walk solvers.

Chapter 2

Power Grid Modeling

2.1 Power Grid Abstraction

Figure 2.1 shows the layout of a portion of a 2D power grid layout where several cells are connected to a V_{dd} line. A *detailed extracted circuit* for the power grid of this figure is depicted in Figure 2.2. In this figure, the resistors model the wire resistances and the triangles symbolize various cells connected to the power grid. The capacitance and the inductance of the network are not shown since we are focusing on the steady-state analysis of the grid. Similar to [50], it is assumed that all cell connections are at the lowest metal level and that the tapping points only lie in these layers. At any intermediate metal level, the connections are only made at vias.

For a full chip power grid analysis, detailed extraction results in large systems of equations that are memory-intensive and computationally prohibitive, since every node with a current source must be retained. A simplifying assumption that is conventionally used in power grid extraction for steady-state analysis is to lump all the cells at the power line intersections, i.e., to assume the cells are connected to the power grid only at the power line intersections, which results in over all much smaller power grid to analyze, as illustrated in Figure 2.3. As pointed out in [50], this approximation has been though to be essential because it

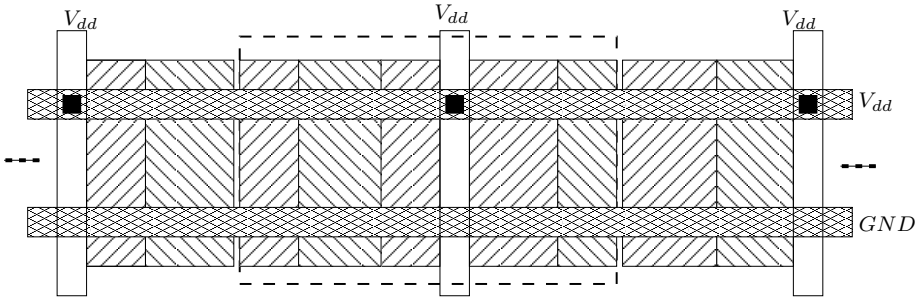


Figure 2.1: An example of the layout of a portion of a power grid.

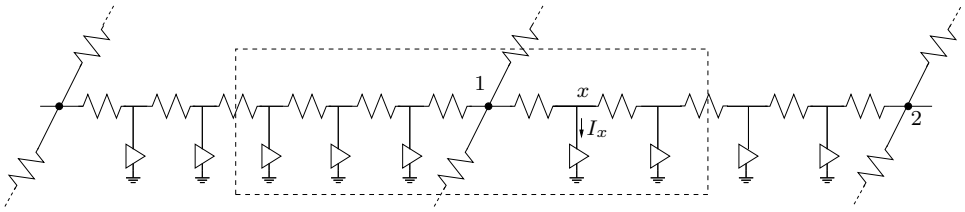


Figure 2.2: Schematic of the detailed extracted power grid of Figure 2.1.

is impractical to model individual connections between the gates and power grid. The lumped approach of Figure 2.3 has several drawbacks:

- It overlooks the dependence of the current drawn by each cell on the voltage of their power supply [50].
- It introduces a discretization error by placing the lumped current source potentially far away from the original locations of the cells.

To account for the first-order dependence of cell current to the voltage of the power supply, each cell can be modeled as a Voltage Controlled Current Source (VCCS) with $I_x = g_x V_x + I_x^0$ for a cell at intermediate node x. Note that the VCCS here is fairly trivial, and is easily modeled by placing a conductance g_x in parallel with each current source. However, the second issue conventionally requires either an exact analysis of a much larger system (containing numerous nodes, corresponding to all cell tapping points on the power grid as shown in

Figure 2.2), or an approximate analysis of a lumped system with discretization error.

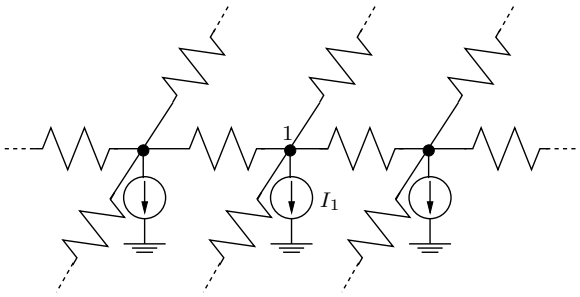


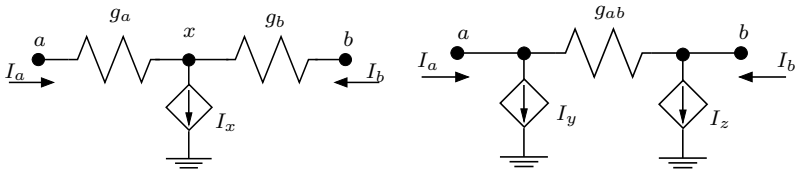
Figure 2.3: Schematic of the simplified extracted power grid of Figure 2.1.

In Section 2.2, we propose an approach for avoiding the discretization error in moving to a coarse extracted grid that eliminates all nodes but those at the intersections of grid wires, as in the simplified extracted grid. Using the VCCS model makes the task of going from detailed extracted power grid to a coarse extracted grid more difficult due to dependence of the current loads on intermediate nodes (i.e., nodes not on the power grid intersections) that are available only in the detailed extracted circuit. We employ circuit transformations to move the VCCS elements for each cell to VCCS elements at the intersections of the power grid. In effect, the circuit in Figure 2.2 is transformed to that in Figure 2.3, with the difference that the current sources are replaced by VCCS elements.

As we will see, these new VCCS elements are more complex than the simple current source/resistor elements that model individual cells. These elements result in a set of power grid equations that are diagonally dominant with negative off-diagonals, but *asymmetric*. Intuitively, this asymmetry arises because the voltage of each intermediate node is closer in value to voltage of the intersection node it is physically closer to. Therefore, when the intermediate nodes are eliminated, the VCCS may be a stronger function of one adjacent grid intersection node rather than another one, which, in terms of the nodal analysis stamp of a VCCS [45], introduces asymmetry in the LHS of Equation (1.1).

2.2 Modeling the Power Grid using VCCS Elements

In this section, we first discuss the details of obtaining the coarser extracted power grid circuit from the detailed extracted power grid circuit, modeling each cell as a VCCS. Next, we discuss a method for obtaining such a model from a standard set of power grid benchmarks [50].



(a) T-model (b) Π-model

Figure 2.4: Converting a T-model to a Π-model.

Lemma 1 *Consider a circuit with two external ports at nodes a and b, represented either as a T-model shown in the circuit of Figure 2.4(a), or as a* Π*-model circuit of Figure 2.4(b), where the equations for the VCCS elements, I_x, I_y, and I_z, are*

$$\begin{aligned} I_x &= I_x^0 + g_{xa}v_a + g_{xb}v_b + g_x v_x \\ I_y &= I_y^0 + g_{ya}v_a + g_{yb}v_b \\ I_z &= I_z^0 + g_{za}v_a + g_{zb}v_b \\ g_{ab} &= \frac{g_a g_b}{g_a + g_b} \end{aligned}$$

The two models are electrically equivalent if the following relationships hold:

$$\begin{aligned}
g_{ya} &= g_a\left(1-\frac{g_a-g_{xa}}{\hat{g}_x}\right)-g_{ab}\\
g_{yb} &= g_{ab}-g_a\left(\frac{g_b-g_{xb}}{\hat{g}_x}\right)\\
I_y^0 &= \frac{g_a}{\hat{g}_x}I_x^0\\
g_{za} &= g_{ab}-g_b\left(\frac{g_a-g_{xa}}{\hat{g}_x}\right)\\
g_{zb} &= g_b\left(1-\frac{g_b-g_{xb}}{\hat{g}_x}\right)-g_{ab}\\
I_z^0 &= \frac{g_b}{\hat{g}_x}I_x^0
\end{aligned} \tag{2.1}$$

where $\hat{g}_x = g_a + g_b + g_x$.

Proof: See Appendix B □

To obtain the coarsened power grid from the detailed extracted grid, Lemma 1 is progressively applied on the intermediate nodes of the detailed extracted grid to replace the T-models with Π-models. In each wire segment In the original circuit, shown in Figure 2.2, each gate at node x is represented by a conductance g_x in parallel with a constant current source; therefore, $g_{xa} = g_{xb} = 0$. The transformation in the lemma is then applied to a T-model in Figure 2.5(a), marked with a dashed square around it, to create the corresponding Π-model shown in Figure 2.5(b). Next, the adjacent VCCS elements are summed up to create new T-models, shown in Figure 2.5(c) that are successively reduced. Finally, each wire segment between intersections is reduced to the Π-model shown in Figure 2.5(d), and the final circuit has the look of the lumped circuits used in [50] with all sources at the power grid intersections, except that the sources at each node are not independent current sources, but VCCS elements that introduce asymmetry into the G matrix. please check the paragraph above and the corresponding figures.

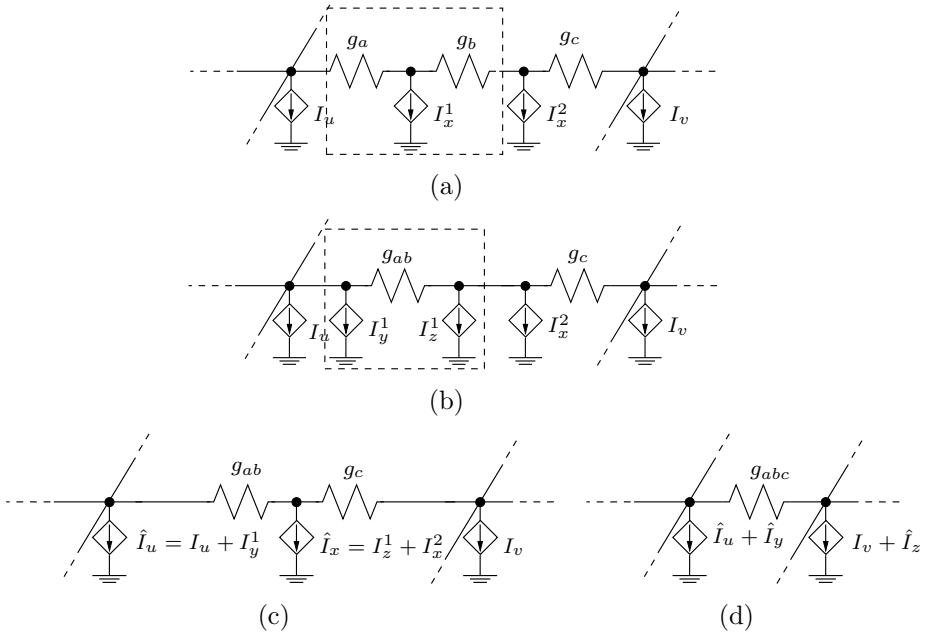


Figure 2.5: Steps for eliminating an intermediate nodes in the detailed extracted grid where $g_{abc} = g_{ab}g_c/(g_{ab} + g_c)$.

Note that the above process is exact and makes no approximations, implying that the impact of the voltage of the power line at each cell is accounted for accurately. Therefore, the dependence of the load of each cell on its supply voltage is correctly captured, and the discretization error mentioned in Section 2.1 is avoided.

Table 2.1 quantifies the inaccuracy that is inherent in the conventional lumped model of Figure 2.3. For the detailed extracted circuits corresponding to power grid benchmarks of [50], which are created using the procedure explained in Appendix A, we quantify the error introduced by the lumped approximation. The average and maximum modeling errors shown in the table are normalized to the value of V_{dd} and also normalized to value of maximum voltage drop in the power grid. The error as normalized to V_{dd} is large for the first two benchmarks and smaller for the others. This can be attributed to the fact that the worst-case drops on these circuits are larger. An alternative, and possibly better, metric for

Table 2.1: Quantifying the inaccuracy in voltage estimation using lumped current source models.

benchmark	Current source model error			
	Norm. to V_{dd}		Norm. to max voltage drop	
	Mean	Max	Mean	Max
ibmpg1	6.14%	28.38%	14.96%	69.11%
ibmpg2	5.75%	14.51%	32.26%	81.48%
ibmpg4	0.05%	0.14%	8.98%	25.43%
ibmpg5	0.12%	0.52%	3.49%	14.63%
ibmpg6	1.07%	3.72%	12.91%	44.85%
Average	**2.63%**	**9.45%**	**14.52%**	**47.10%**

measuring the error is to compare the error as a fraction of the maximum voltage drop over all nodes in the circuit[1] . Under this metric, it can be seen that the error is quite significant for all circuits, motivating the need for using our VCCS model over the existing lumped model.

The higher accuracy of the VCCS model comes at the expense of an *asymmetrical* LHS. The hybrid solver of [54] and forward solver of [55] are not capable of solving a system with asymmetrical LHS and it is harder for direct and iterative methods to solve. In this thesis, we propose a backward random walk solver that can solve asymmetrical equations as efficiently as symmetric equations.

[1] We choose not to show normalization of the error at a node with respect to the voltage drop at the node since it exaggerates small errors at nodes with small voltage drops, thus providing largely meaningless data.

Chapter 3

Forward and Backward Random Walks

In this chapter we first briefly explain the forward random walk game introduced by [32]. Next, we discuss in detail the motivation for backward random walks and how it is constructed to solve power grid equations. Finally we show the relation of backward random walk game with LU matrix decomposition. The relation of the forward random walk game and LU decomposition of matrix is shown in [54].

3.1 Forward Random Walks

The forward random walk game [32] is a construction for solving Equation (1.1) as $\mathbf{V} = G^{-1}\mathbf{E}$, that is based on the *rows* of LHS matrix, G. The method proceeds by solving the voltage at any specific node i as:

$$v_i = \begin{bmatrix} (G^{-1})_{i1} & (G^{-1})_{i2} & \dots & (G^{-1})_{iN} \end{bmatrix} \mathbf{E} \tag{3.1}$$

where $(G^{-1})_{ij}$ is the $(i,j)^{\text{th}}$ element of G^{-1}, and N is the dimension of G.

The forward random walk game is based on a network of roads with motels or homes at its intersections. To compute the voltage at node i, the walker starts with zero money in his pocket from the motel at i, pays for the cost of the motel

(on credit), and takes one of the roads at the intersection randomly, according to some known probability distribution p_{ij}, to get to an adjacent intersection, j, where $j \in \{1, \ldots, \text{degree}(i)\}$, and degree($i$) denotes the number of roads at intersection i. This process of paying for the motel at the current node and randomly picking the next node is repeated until the walker reaches one of the specially designated home nodes in the circuit, where a reward is collected.

Consider a node i in a conventional coarse extracted power grid model of Figure 2.3, connected by resistors to each neighboring node $j = 1, \cdots, degree(i)$ with conductance g_{ij}, and with a grounded current source at i, I_i (possibly of value 0). Let v_k be the voltage at any node k in the network. The application of Kirchhoff's current law, Kirchhoff's voltage law, and the device equations implies:

$$\sum_{j=1}^{degree(i)} g_{ij}(v_j - v_i) + I_i = 0 \tag{3.2}$$

The terms in this equation can be rearranged as:

$$v_i = \sum_{j=1}^{degree(i)} \frac{g_{ij}}{\Gamma_i} v_j + \frac{I_i}{\Gamma_i} \tag{3.3}$$

where $\Gamma_i = \sum_{j=1}^{degree(i)} g_{ij}$. Equation (3.4) implies that the voltage of node i is a weighted sum of the voltages of its neighbors. Due to diagonal dominance, all weights lie between 0 and 1, and the sum of the weights is less than 1. Moreover, if we add in conductances to ground at i the sum of the weights becomes equal to 1. And Equation (3.3) can be written as:

$$v_i = \sum_{j=1}^{degree(i)} \frac{g_{ij}}{g_{ii}} v_j + \frac{g_{i,GND}}{g_{ii}} \times 0 + \frac{I_i}{g_{ii}} \tag{3.4}$$

where $g_{i,GND}$ is the conductance of node i to ground and $g_{ii} = \Gamma_i + g_{i,GND}$ is the i^{th} diagonal element of LHS matrix G. For a power grid that has N non-V_{DD} nodes, we may write N equations of this form, one for each node. The solution of this system of N equations provides the exact solution to Equation (1.1).

The nodes in the random walk game have a one-to-one correspondence with those in the power grid. Nodes that are connected by resistors in the power grid are connected by roads in this network. At a given node, the probability of taking a specific road maps on to the corresponding weight, $p_{ij} = g_{ij}/g_{ii}$, for the edge (Equation (3.4)). Each node has a motel that charges a cost of $m_i = I_i/g_{ii}$ when visited. The nodes that correspond to fixed voltages v_i (e.g., V_{DD} or ground nodes) provide a reward of v_i, and are called *home nodes.*

A system of equations defined by Equation (3.4), for all i, can be modeled as a random walk game on this network of roads, where the walker goes from node i to its adjacent node j with probability p_{ij} and pays a motel cost, m_i, at each visited node. The walk terminates when the walker arrives at a home node.

For a node of interest, i, a random walker starts with zero money and a credit card that allows motel costs to be charged until a reward is obtained for reaching a home node. We define this amount for each random walk as the *gain* of that walk. Then the expected value of the gain, v_i, over all walks that begin at node i is given by

$$v_i = \sum_{j=1}^{degree(i)} p_{ij} v_j + m_i \tag{3.5}$$

It is easy to see that this equation and Equation (3.4) correspond exactly. The notation, v_i, is overloaded to denote both the average gain of the random walk and the voltage of the node, precisely because the two are identical, i.e., the expected value of the walk gain, given that the walk started from node i, maps directly to the voltage at node i.

The expected value of v_i from the random walk can be estimated by running a sufficiently large number of random walks from i and calculating the average result to get the estimated voltage of node i without solving for any other node within the network. If the number of walks from i is M_i, then:

$$v_i = \frac{1}{M_i} \sum_{k=1}^{M_i} V_i^k \tag{3.6}$$

where V_i^k is the walk gain in the k^{th} random walk. We refer to this approach as the *naïve simulation approach*. This approach is always unbiased [51] since $v_i = E_P[V_i]$ where V_i is the random variable with probability mass function (PMF) P_i^k equal to the probability of k^{th} random walk in Equation (3.6), which is equal to the product of the transition probabilities of the k^{th} random walk starting from node i.

In the random walk game above, the motel costs do not play a role in road probabilities and hence the likelihood that each road being taken by the walker. In other words, in this game for all the walks starting from node i, if we keep the record of the number visits, including the revisits, to each motel, j, the walk gain and therefore the node voltage, v_i, can be computed for any motel cost by simply applying the same walk record to the motel costs. In other words, the record of home visits is directly related to the elements of i^{th} row of the inverse of LHS matrix of Equation (1.1). The work of [32] shows that this relation for $i = 1, \cdots, N$ can be written as:

$$(G^{-1})_{ij} \;=\; \frac{\text{Number of visits to motel } j \text{ in } M \text{ walks from } i}{M \times g_{ii}} \; j \in [1, N] \quad (3.7)$$

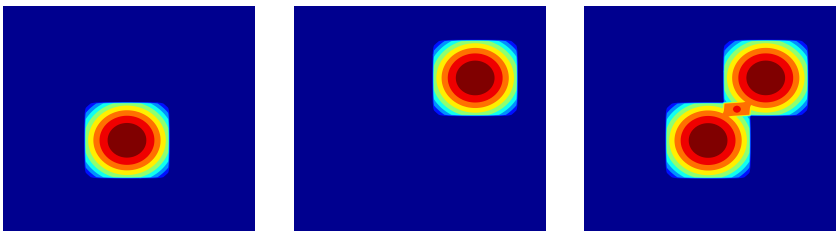
3.2 Motivation for Computing G^{-1} by the Column

In effect, as shown in [33, 54], the forward random walk method computes G^{-1} by the row for the row(s) of interest, corresponding to the nodes where the voltages are to be evaluated. Reconsidering Equation (1.1), its solution $\mathbf{V} = G^{-1}\mathbf{E}$, can also be written in another form as:

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_N \end{bmatrix} = \begin{bmatrix} (G^{-1})_{11} \\ (G^{-1})_{21} \\ \vdots \\ (G^{-1})_{N1} \end{bmatrix} e_1 + \cdots + \begin{bmatrix} (G^{-1})_{1N} \\ (G^{-1})_{2N} \\ \vdots \\ (G^{-1})_{NN} \end{bmatrix} e_N \quad (3.8)$$

where v_i and e_i are the i^{th} elements of $\mathbf{V}$ and $\mathbf{E}$, respectively.

Each element, $[(G^{-1})_{1i}, (G^{-1})_{2i}, \ldots, (G^{-1})_{Ni}]^T e_i$, of the summation is the contribution of the i^{th} element of the RHS, $\mathbf{E}$, on the solution vector $\mathbf{V}$. The full solution simply is the superposition of the contributions of all e_i. In the context of ground network analysis, this is the contribution of each current load on the node voltages of the network. In other words, each product within the sum represents the influence of e_i, and the nodes with near-zero values fall outside the RoI of e_i.



(a) (b) (c)

Figure 3.1: Contribution of individual current loads on the power network solution.

On-chip power grids with C4 pins distributed over the area of the chip are well known to satisfy the property of locality, whereby the voltage drop impact of an individual current source is predominantly felt at nodes that lie in the region near the source, and not in faraway areas. This is illustrated in the schematic contour plot in Figure 3.1(a), where a current load is applied at a certain node and its influence on the voltage drop (effectively, computing $\mathbf{V} = G^{-1}\mathbf{E}$ by the column corresponding to this single excitation) is seen to show the property of locality. If another excitation is applied due to another current load, as shown in Figure 3.1(b), another contour plot is obtained, and if both are applied together (Figure 3.1(c)), then the result is a superposition of the contours, consistent with Equation (3.8).

While this observation is of limited utility in the complete solution of $G\mathbf{V} = \mathbf{E}$ for a realistic power grid, where the number of current loads is extremely large, it has particular application to the problem of incremental analysis. As we will see in Section 5.1, the impact of small perturbations to the power grid can be modeled by

a small number of current loads. Therefore it is reasonable to attempt to compute the matrix inverse by the column, for the columns corresponding to these current loads, in order to generate a meaningful solution in a modest amount of time. For such a case, the nodes affected that constitute the RoI can be computed using a method similar to that illustrated in Figure 3.1. Therefore, there is a close relation between finding the RoI and Equation (3.8). As we will show soon, the use of backward random walks permits this computation in an easy manner.

3.3 The Backward Random Walk Game

In this section, we qualitatively explain the backward random walk method in contrast with the forward random walk approach. The key difference is that the forward game is constructed based on the *rows* of the LHS matrix, G, inverting one row at a time, and captures the effect of all of the RHS elements on the solution of a single node; while the backward game is constructed based on the *columns* of G, inverting one column at a time, and captures the effect of a single source on the solution of the entire system. Therefore, the forward game finds the matrix inverse, G^{-1}, by the row, and the backward game finds this matrix inverse by the column.

By Equation (3.8), for the case where there are few nonzero elements on the RHS (e.g., during incremental analysis), only a few corresponding columns of G^{-1} must be computed, and backward random walks are extremely appealing. The forward game, on the other hand, is suitable for the case where the voltages of a small subset of nodes in the network are desired, as seen from Equation (3.1).

As described later in Section 3.4, the construction of the forward and backward games is the same except in the fact that the forward walk road probabilities are constructed based on the *rows* of G, while the road probabilities of the backward game are based on the *columns* of this matrix. In both games multiple walks are started from a *motel* of interest, ended when a *home* is reached, and the motel visits are recorded. The difference then is how this information is interpreted.

In a forward game, as mentioned in Section 3.1, the analogy is that the walker has to pay for each motel he visits and gets an award as he reaches a home. In this process the average number of visits to motel nodes are translated into the rows of G^{-1} and the sum of money into node voltages, in other words, sum of the contribution of each motel (RHS element) on the walker's total money.

In contrast, the analogy for the backward game is that the walker has a sum of money that he has to distribute among the motels he visits based on how frequently he visits them. Then the average number of motel visits is translated to the columns of G^{-1} and the sum of money (the contribution of one RHS element on entire system) distributed on all the nodes in the game, into the contribution of one RHS entry on the voltage of all nodes.

For a symmetrical G, the forward and backward games turn out to be identical and the columns of G^{-1} can be obtained by transposing the corresponding rows of G^{-1} computed by forward random walks and vise versa. However, as mentioned in Section 2, the LHS of the power grid equations can be *asymmetrical* where the forward and backward games turn out to be different. Like the forward solver, the backward solver is also directly related to the LU decomposition of the LHS matrix, as discussed in Section 3.8). As in [33, 54], this can be used as a preconditioner in combination with an iterative solver, but this is not explored in detail within this thesis.

3.4 Constructing the Backward Walk Game

We now describe how the backward random walk game is constructed from Equation (1.1), and how the columns of G^{-1} are computed using this approach. As in the forward walk game, this game is based on a network of roads with motels or homes at its intersections. To compute the j^{th} column of G^{-1}, the walker starts from the motel at j, pays for the motel, and takes one of the roads at the intersection randomly, according to some known probability distribution p_{ji}, to get to the adjacent intersection, i, where $i \in 1, \ldots, \text{degree}(j)$, and degree$(j)$ denotes the

number of roads at intersection j. The walker continues until a home is reached. This completes one full *walk*. The number of motels visited, including the revisits, on the path is called *walk length*. The goal of this game is to enumerate the visits to each motel during a walk.

Theorem 1 *The LHS matrix G of the power grid with asymmetric VCCS models satisfies the following properties:*

$$\sum_{i=1,i\neq j}^{N} |g_{ij}| \leq g_{jj},\ \forall j \tag{3.9}$$

$$\begin{cases} g_{ij} & \leq & 0 & j \neq i \\ g_{ij} & > & 0 & j = i \end{cases} \tag{3.10}$$

Proof: See Appendix C. □

For power grid LHS matrix G with these properties, Equation (3.9) can be modified as:

$$\begin{aligned} \sum_{i=1,i\neq j}^{N+1} |g_{ij}| &= g_{jj},\ \forall j \\ g_{j(N+1)} &= -\left(g_{jj} - \sum_{i=1,i\neq j}^{N} |g_{ij}|\right)\ \forall j \end{aligned} \tag{3.11}$$

The added conductances are accounted for in Equation (1.1) by adding dummy node $N+1$ with known solution of 0. The insertion of the dummy node is essential to obtain valid road probability distributions discussed in Theorem 6. In the rest of this chapter, we assume that G notation is overloaded to include this added dummy node.

Theorem 2 *For each column of a system of linear equations defined by Equation* (1.1)*, a valid probability mass distribution can be defined as: Does not include the column corresponding to the dummy node $N+1$.*

$$p_{ij} = \begin{cases} -g_{ji}/g_{jj} & j \in [1,N], j \neq i \\ 0 & j = i \end{cases} \tag{3.12}$$

Proof: For a probability mass function to be valid, all probabilities should lie between 0 and 1, and the sum of all probabilities should be 1. For the conductance matrix of a power grid, G, using Equations (3.10) and (3.11), for $\forall i, j$, we have:

$$0 \leq p_{ij} \leq 1$$
$$\sum_{i=1}^{N} p_{ij} = 1$$

□

Note that in Equation (6.8), $p_{ij} = 0$ for many of the j's due to the sparsity of G. Writing the p_{ij}'s in matrix form, P, where subscript i and j denote the row and column index of this matrix, we get:

$$G = (I - P)^T D \tag{3.13}$$

where D is the matrix of the diagonals of G, and I is the identity matrix. Notice that as mentioned in Section 3.3, the road probabilities on the rows of matrix P are computed using the columns of the LHS matrix, G, which distinguishes the backward game from the forward game.

A random walk game can be constructed from Equation (1.1) by modeling each element of the vector $\mathbf{V}$ as an intersection, nonzero elements of the LHS as roads, and road probabilities as in Equation (6.8). Note that the vector $\mathbf{V}$ consists of both unknowns as well as known variables. The unknown and known variables of vector $\mathbf{V}$, are mapped to motels and homes, respectively.

Note that in construction of the game, connectivities and transition probabilities, and therefore number of node visits, are determined by the LHS only.

3.5 An Example Illustrating the Forward and Backward Games

Consider the following diagonally dominant matrix:

$$G = \begin{bmatrix} 1.5 & 0 & -1 & 0 \\ 0 & 2 & -1 & -0.5 \\ -0.75 & -1.25 & 2.25 & -0.25 \\ 0 & 0 & -0.25 & 1.25 \end{bmatrix} \tag{3.14}$$

Since we are interested in illustrating the setup for the backward [forward] random walks required to create G^{-1} by the column [row] (rather than the solution vector $\mathbf{V}$), we do not show the RHS vector $\mathbf{E}$ in this example.

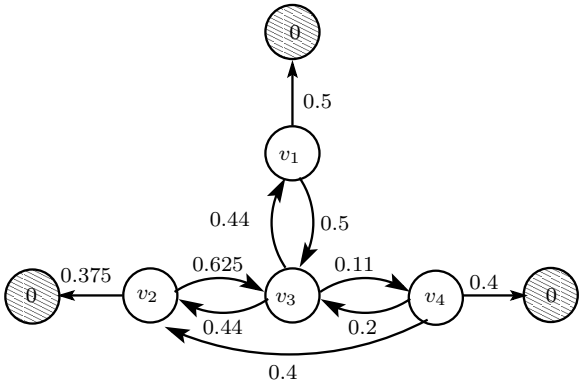


Figure 3.2: An example of a backward random walk game modeling Equation (3.14).

To construct the backward game corresponding to this matrix, a dummy node is added to this equation such that the columns of G sum up to zero as described in Section 3.4. Next the *columns* of G is normalized to its diagonals, such that it can be written in the form of Equation (3.13), a row-wise probability matrix (corresponding to the transpose of the normalized G). Figure 3.2 shows the random

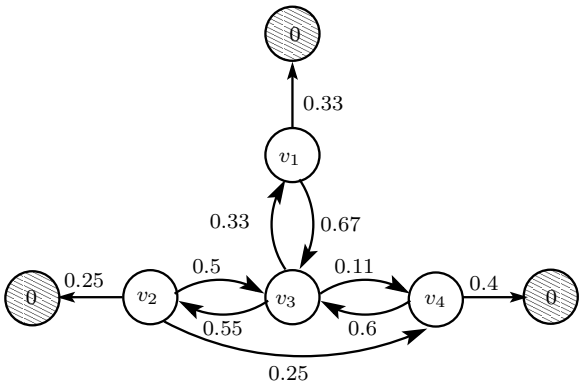


Figure 3.3: An example of a forward random walk game modeling Equation (3.14).

walk game, with the probability matrix:

$$P_{backward} = \left[\begin{array}{cccc|c} 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0.625 & 0 & 0.375 \\ 0.44 & 0.44 & 0 & 0.11 & 0 \\ 0 & 0.4 & 0.2 & 0 & 0.2 \end{array}\right] \tag{3.15}$$

Note that in this equation the last column corresponds to the ground node, to generate all probabilities as in Equation (3.11).

Similarly the forward random walk game is constructed based on probability matrix of Equation (3.16) which is computed by adding a dummy node to G such that its rows sum up to zero, and normalizing the *rows* of matrix G to its diagonals. Figure 3.3 shows the corresponding game.

$$P_{forward} = \left[\begin{array}{cccc|c} 0 & 0 & 0.67 & 0 & 0.33 \\ 0 & 0 & 0.5 & 0.25 & 0.5 \\ 0.33 & 0.55 & 0 & 0.11 & 0 \\ 0 & 0 & 0.6 & 0 & 0.4 \end{array}\right] \tag{3.16}$$

Having the random walk game constructed, the following section shows how it is used to compute the columns of the G^{-1} individually without computation of the entire G^{-1}.

3.6 Computing the Individual Columns of G^{-1}

In this section we show how the individual columns of G^{-1} can be computed using backward random walks.

Carrying out M walks from intersection j, one can find the conditional expected number of visits to each motel by:

$$\begin{aligned} z_{ij} &= \frac{\text{Number of visits to motel } i \text{ in } M \text{ walks from } j}{M} \\ Z &= [z_{ij}]_{N\times N} \end{aligned} \tag{3.17}$$

where z_{ij} is the expected number of visits to motel i when total of M walks initiated from motel j. This expected value becomes more exact as $M \to \infty$.

It is shown in [31] that the backward random walk game constructed based on the of probability matrix P, as described in Section 3.2, gives the solution to:

$$\begin{aligned} (I-P)^T\mathbf{X} &= \mathbf{U} \\ \mathbf{X} &= Z\mathbf{U} \end{aligned} \tag{3.18}$$

where

$$Z = (I-P)^{-T} \tag{3.19}$$

Translating the notation of [31] to the notation in our problem,

$$\begin{aligned} \mathbf{X} &= D\mathbf{V} \\ \mathbf{U} &= \mathbf{E} \end{aligned} \tag{3.20}$$

Substituting Equations (3.19) and (3.20) into Equation (3.18) we can see that the backward random walk approach, as formulated in [31], provides the solution to Equation (1.1). Specifically,

$$\begin{aligned} D\mathbf{V} &= (I-P)^{-T}\mathbf{E} \\ \text{i.e., } \mathbf{V} &= D^{-1}(I-P)^{-T}\mathbf{E} = G^{-1}\mathbf{E} \end{aligned}$$

where the last equality follows from Equation (3.13).

Finally, from Equation (3.13) and (3.19), we can see that $G^{-1} = D^{-1}Z$. Therefore, the elements of G^{-1} can be written as:

$$(G^{-1})_{ij} = \frac{z_{ij}}{g_{jj}},\ i = 1, \ldots, N \tag{3.21}$$

For the example of Figure 3.2, let us consider the use of backward random walks to compute the second column of the exact G^{-1}. Table 3.1 shows the calculated value of this column, denoted by $(G^{-1})_{*2}$, and the average relative error, in percent, of the estimation. In order to avoid bias from a specific run of the random walk method, we show the average is over 100 runs of the random walk method.

This table suggests that as M increases, the random walk results become more accurate. As we will see later in Section 3.7 this table also suggests that the error of the random walk results halves for 4 times more walks. In other words the error of the random walk method is proportional to $1/\sqrt{M}$. Note that for even a small number of walks, the results are fairly accurate. Hence, if a rough but fast estimation of G^{-1} is desired, backward random walks is a suitable candidate.

Table 3.1: Average relative error, in percent, for estimated G_2^{-1} for different walk numbers, M.

$(G^{-1})_{*2}$	$M = 25$	$M = 100$	$M = 400$	$M = 1600$
0.4115	26.7%	12.0%	5.8%	2.9%
0.8395	10.1%	4.9%	2.5%	1.2%
0.6173	21.3%	8.9%	4.6%	2.4%
0.1235	34.3%	18.9%	10.8%	5.1%

3.7 Stopping Criteria for the Random Walks

In this section we show how the number of walks is determined dynamically as the random walks are proceeding to get a desired accuracy for the forward random walk game and backward random walk game.

3.7.1 Stopping Criteria for Forward Game

For the forward random walk game similar to work of [32], the number of walks is determined dynamically based on the estimated variance of the walk gains (i.e. the money in the walker's pocket at the end of each walk) at each point in the random walk game. Let us say random variable V_i represents the walk gain from node i. Given K random walks are carried out from node i, one can get an estimate of the variance of V_i as:

$$\sigma_{V_i}^2 = \frac{1}{K-1}\sum_{k=1}^{K}(V_i^k - v_i^*)^2 \tag{3.22}$$

where v_i^* is the estimate of the solution v_i. This estimate then is used to make sure that the absolute error of the estimated v_i is smaller than a given threshold Δ with confidence of θ such that:

$$P\left[-\Delta < v_i - v_i^* < \Delta\right] > \theta \tag{3.23}$$

where $P[.]$ denotes the probability. A typical value for θ is 99%.

Equation (3.6) is the average value of M_i independent identically distributed (IID) random variables since each walk is carried out without any knowledge from the others. Hence the running average of the walks can be modeled as a normal random variable with mean v_i and variance $\sigma_{V_i}^2/M_i$ (by the Central Limit Theorem [56], this is an exact model as $M_i \to \infty$). Hence to satisfy Equation (3.23) we should have:

$$\begin{aligned} Q\left(\frac{\Delta\sqrt{M_i}}{\sigma_{V_i}}\right) &< \frac{1-\theta}{2} \\ M_i &> \frac{\sigma_{V_i}^2}{\Delta^2}\left(Q^{-1}\left(\frac{1-\theta}{2}\right)\right)^2 \end{aligned} \tag{3.24}$$

where Q is the tail probability of normal distribution. Note that in practice, several walks, e.g., 10 walks, should be performed to get a reasonably good estimate of the v_i^* and σ_{V_i} before Equation (3.24) can be used.

3.7.2 Stopping Criteria for Backward Game

For the backward random walks the accuracy is defined as the relative error of the absolute sum of the solution, $\mathbf{V}$, from Equation (1.1) as defined by:

$$err = \frac{\sum_{i=1}^{N} (|v_i| - |v_i^*|)}{\sum_{i=1}^{N} |v_i^*|} \tag{3.25}$$

where v_i and v_i^* are the i^{th} element of the random walk solution and the exact solution respectively, and $|.|$ denotes the absolute value.

For M full walks starting from motel j, we take advantage of the fact that the total walk length is equal to the total number of visits to all of the motels. Mathematically, this is stated as:

$$\sum_{i=1}^{N} z_{ij} = \frac{1}{M} \sum_{s=1}^{M} w_{sj} \tag{3.26}$$

where z_{ij} is defined by Equation (3.17) and w_{sj} represents the length of the s^{th} walk starting from j.

Moreover, if W_j represents the random variable of the length of the walks starting from node j, we have:

$$\frac{1}{M} \sum_{s=1}^{M} w_{sj} \approx E[W_j] \tag{3.27}$$

where $E[W_j]$ is the expected value of W_j. This estimation becomes exact as the number of walks tends to infinity.

Theorem 3 shows relation of the relative error to the average walk length, which is calculated as the random walk game proceeds, and is used to estimate the relative error of the solution on the fly.

Theorem 3 *The relative error of Equation* (3.25)*, corresponding to* j^{th} *motel, is given by:*

$$err = \frac{\frac{1}{M} \sum_{s=1}^{M} w_{sj} - E[W_j]}{E[W_j]} \tag{3.28}$$

Proof: Consider a backward walk starting from node j, designed to determine the contribution v_i^j of the j^{th} element of $\mathbf{E}$, denoted by e_j, on the voltage of some node i. We obtain:

$$v_i^j = (G^{-1})_{ij} e_j = \left(\frac{z_{ij}}{g_{jj}}\right) e_j \tag{3.29}$$

where the second equality follows from Equation (3.21). The sum of the absolute values of this error, over all nodes i, is then given by:

$$\sum_{i=1}^{N} |v_i^j| \;=\; \frac{|e_j|}{|g_{jj}|} \sum_{i=1}^{N} z_{ij} \tag{3.30}$$

Substituting $\sum_{i=1}^{N} z_{ij}$ from Equation (3.26) we have:

$$\sum_{i=1}^{N} |v_i^j| \;=\; \frac{|e_j|}{M|g_{jj}|} \sum_{s=1}^{M} w_{sj} \tag{3.31}$$

As $M \to \infty$, the solution of random walk method converges to the exact solution v_i^{j*}, and we have:

$$\sum_{i=1}^{N} |v_i^{j*}| \;=\; \frac{|e_j|}{|g_{jj}|} E[W_j] \tag{3.32}$$

where v_i^{j*} is the i^{th} element of the exact solution corresponding to j^{th} column. Substituting Equation (3.31) and (3.32) into Equation (3.25) completes the proof. □

Incremental solution of a system for a given perturbation, involves computation of several columns of G^{-1} corresponding to nonzeros of RHS. Theorem 3 is useful to compute each of these columns individually with a desired relative error corresponding to one nonzero element of RHS. It is easy to verify that the result of Theorem 3 holds for general case that there are multiple nonzeros in the RHS. Therefore the relative error of the updated solution corresponding to all the nonzero elements of the RHS is the same as the relative error of the individual columns. Note that the sum of several variables with the same relative error will have the same relative error.

An immediate result of Theorem 3 is the number of walks required to achieve a desired relative error of δ with a confidence of α. In other words to have:

$$prob\left[\left|\frac{1}{M}\sum_{s=1}^{M} w_{sj} - \mu\right| < \mu\delta\right] > \alpha \tag{3.33}$$

where $prob[.]$ is the probability function and μ is the expected value of the random variable of the average walk length, $\frac{1}{M}\sum_{s=1}^{M} w_{sj}$. The walk length average is the sample average of a series of identical and independent (I.I.D.) random variables, W_j, the random variable of the length of walks starting form j. Therefore according to the central limit theorem we have:

$$\frac{1}{M}\sum_{s=1}^{M} w_{sj} \sim \mathcal{N}(\mu, \sigma) \tag{3.34}$$

where $\mathcal{N}(\mu, \sigma)$ denotes the normal distribution with mean of $\mu = \mu_{W_j}/M$ and standard deviation of $\sigma = \sigma_{W_j}/\sqrt{M}$, where μ_{W_j} and σ_{W_j} are the mean and standard deviation of W_j.

Taking advantage of the normal distribution of Equation (3.34), Equation (3.33) will be satisfied if the following holds:

$$\frac{(\sigma/\mu)^2}{M} < \left(\frac{\delta}{Q^{-1}\left(\frac{1-\alpha}{2}\right)}\right)^2 \tag{3.35}$$

where Q^{-1} is the inverse of the Q-function, defined as $Q(x) = 1/\sqrt{2\pi}\int_x^\infty e^{-u^2/2}du$.

In Equation (3.35), note that $M \propto 1/\delta^2$. Therefore, in order to get two times more accurate solution, four times more walks is required. As a result, random walks are efficient for moderate accuracies but for higher accuracies this method could be expensive.

3.8 Backward Walks and LU Decomposition

Having developed the idea of backward random walks as an alternative to forward random walks, and having applied backward walks to efficiently solve the

incremental analysis problem, we now explore another novel theoretical result. We study an interesting correspondence between the backward random walks of Section 3.3 with LU decomposition of the LHS matrix of Equation (1.1), G. This relation can be used to find a quick and moderately accurate LU factorization of G which can be used for a variety of applications, e.g., as a preconditioner for an iterative method similar to the work of [54]. The work of [54] showed the relation between the UL factors of the LHS and the forward random walks, and this is its counterpart for backward walks. This relationship is not specifically used by the incremental solver but is pointed out in this section for completeness.

The basic idea behind this relation is the notion of *reusing* the walks. Taking a second look at the backward random walk game described in Section 3.3, it is easy to see that during a walk, when the walker arrives at an intersection that has been frequently visited in the past, further walks are not necessary: the walker already has all the necessary information. Therefore, for the purposes of the game, he can simply stop the walk and use the previous information, i.e., reuse the previous walks.

In practice, this notion is implemented simply by marking a processed motel as a home and keeping two separate visit records, one for motel visits, and one for the stops at the home nodes which are the previously processed motels. Formally, we define

$$\begin{aligned} q_{ij} &= \frac{\text{Number of visits to motel } i \text{ in } M \text{ walks from } j}{M} \\ h_{ij} &= \frac{\text{Number of stops at home } i \text{ in } M \text{ walks from } j}{M} \end{aligned}$$

Writing these in matrix format, we have:

$$Q = [q_{ij}]_{N\times N}\,,\ H = [h_{ij}]_{N\times N} \tag{3.36}$$

where Q and H contain the motel and home visit records, respectively. Note that the matrices Q and Z have similar definitions, but the difference is that the Q matrix incorporates the effect of stopping at home nodes that are defined during the process of solving the system. As discussed shortly, Q is a lower triangular

matrix while Z is a full matrix. The G matrix can be constructed from either Z and D or from Q, H, and D.

Without loss of generality, assume that the motels are processed in natural order. Hence, when processing motel l, all of the motels $j < l$ are previously processed and marked as home. As a result, the indices of all motels that the walker visits on his way are greater than or equal to the starting motel index. Similarly, the indices of all of the home nodes that the walker stops at are strictly less than the starting motel index. Therefore, Q and H will be lower-triangular and strictly upper-triangular matrices, respectively.

As discussed in Section 3.6 there is a direct relation between the backward random walks and G^{-1}, through Equation (3.17). Similarly, there is a direct relation between H, Q and G^{-1}, stated in Theorem 4 below. The idea behind this relation is that the full visit records of Equation (3.17) can be reconstructed from columns of H and Q.

Theorem 4 *Given the visit records, $\{H, Q\}$, G^{-1} can be found by:*

$$\begin{cases} Z_1 = Q_1 \\ Z_j = Q_j + \sum_{i=1}^{j-1} h_{ij} Z_i, \ j = 2, \ldots, N \end{cases} \tag{3.37}$$

$$G^{-1} = D^{-1} Z \tag{3.38}$$

where Z_j and Q_j denote the j^{th} column of Z and Q, respectively, and D is the matrix of the diagonals of G.

Proof: The case of $j = 1$ is trivial since no processed motel exists for the walker to stop at.

For the case of $j > 1$, by definition, out of M walks started from node j, $M \times h_{ij}$ walks stop at node $i < j$. Playing by the rules of the original game, i.e., not stopping at node i, is equivalent to starting $M \times h_{ij}$ new walks from node i after stopping in the modified game. Doing so, walker visits node $k \in [1, N]$, $M \times h_{ij} \times z_{ki}$ times. Accounting for all the nodes that the walker stopped at during

the M walks from node j, number of visits to node k missed due to stopping, $M \times \hat{z}_{ki}$, is:

$$M \times \hat{z}_{ki} = M \sum_{i=1}^{j-1} h_{ij} z_{ki} \tag{3.39}$$

Adding the number of visits to node $k \geq j$, the total number of visits to node k in M walks from node j, $M \times z_{kj}$, we have:

$$M q_{kj} + M \sum_{i=1}^{j-1} h_{ij} z_{ki} = M z_{kj} \tag{3.40}$$

substituting in Equation (3.21) and writing in vector format completes the proof. □

Equation (3.37) can be rewritten in matrix format as:

$$\begin{aligned} G^{-1} &= D^{-1}(I-H)^{-1}Q \\ G &= Q^{-1}(I-H)D = L_G U_G \end{aligned} \tag{3.41}$$

where $L_G = Q^{-1}$ and $U_G = (I-H)D$ are the lower-triangular and upper-triangular factors of G. For a symmetrical G, LDL and Cholesky factorization can be computed from D, H, and diagonals of Q, without actually computing Q^{-1}, similar to [54].

Chapter 4

Forward Incremental Random Walk Solver

4.1 Book-keeping for the Forward Random Walk Method

As discussed in Section 3.1, the expected value of v_i from the random walk can be estimated by running a sufficiently large number of random walks from i and calculating the average result as in Equation (3.6). If the number of walks from i is M_i, and if the number of visits to each motel node is J_{ij}, and to each home node is H_{ik}, then:

$$v_i = \frac{1}{M_i}\left(\sum_{k\in\text{homes}} H_{ik}v_k - \sum_{j\in\text{motels}} J_{ij}\frac{I_j}{g_{jj}}\right) \tag{4.1}$$

To reuse computations, the walks can be made shorter by converting a motel node to a home node once it is solved.

Without loss of generality, assume that the nodes are solved in natural order, $i = 1, 2, \cdots, N$, so that when solving node i, nodes $1, 2, \cdots, i-1$ are previously

solved and marked as home nodes. By rewriting Equation (4.1), we have:

$$v_i = \sum_{j=1}^{i-1} \frac{H_{ij}}{M_i} v_j - \sum_{j=i}^{N} \frac{J_{ij}}{M_i} \frac{I_j}{g_{jj}} \tag{4.2}$$

Rewriting Equation (4.2) in matrix form, we have:

$$\mathbf{V} = Y\mathbf{V} + Z\mathbf{b} \tag{4.3}$$

$$\text{i.e., } \mathbf{V} = (I - Y)^{-1} Z\mathbf{b} \tag{4.4}$$

Here, $Y, Z \in \Re^{n \times n}$ and $\mathbf{b} \in \Re^n$. The elements of Y and Z are defined as follows: $z_{ij} = J_{ij}/M_i$ is the average number of visits to motel node j, and $y_{ij} = H_{ij}/M_i$ is the average number of visits to home node j over all the walks started from node i. The elements of $\mathbf{b} \in \Re^n$ are given by $b_i = -I_i/g_{jj}$, and represent the cumulative motel costs at node i over the walks. Some observations about these matrices are:

- Matrix Z is an upper-triangular, and matrix Y is a lower-triangular matrix with zeroes on its diagonal.
- All entries of the Y and Z matrices only depend on the LHS matrix and are independent of RHS.

These two matrices were introduced in [33] and constitute the *bookkeeping information* for the forward random walks. In particular, [33] showed that the matrices Y and Z are directly related to the LU factors of matrix G. The Y and Z matrices play a key role in our incremental solver.

It can be shown that solving Equation (4.4) has a complexity of $O(n^2)$. We present a more computationally efficient method.

4.2 The Incremental Solver Method

4.2.1 The Perturbed System

Our incremental solver proceeds under the reasonable assumption that the original system has already been solved, so that the initial solution to the system of

equations, $G\mathbf{V} = \mathbf{E}$, is known. When the design is perturbed, it may result in a change in either G or $\mathbf{E}$, resulting in the new relationship:

$$(G + \Delta G)(\mathbf{V} + \Delta\mathbf{V}) = \mathbf{E} + \Delta\mathbf{E} \tag{4.5}$$

where ΔG models the change in the left hand side (LHS), $\Delta\mathbf{E}$ models the change in the right hand side (RHS), and $\Delta\mathbf{V}$ is the change in the solution caused by the perturbation. Assuming that the perturbation is small, the product of the incremental terms can be ignored, giving us:

$$\begin{aligned} G\Delta\mathbf{V} + \Delta G\mathbf{V} &\approx \Delta\mathbf{E} \\ \text{i.e., } G\Delta\mathbf{V} &= \Delta\hat{\mathbf{E}}, \end{aligned} \tag{4.6}$$

where $\Delta\hat{\mathbf{E}} \triangleq \Delta\mathbf{E} - \Delta G\mathbf{V}$.

The significance of Equation (4.6) is that any perturbation in the design can be modeled as an equation with the same LHS and a new RHS, corresponding to new excitations. The circuit interpretation of this equation is that the network of conductances remains unchanged, but the excitations now correspond to the new right hand side. Specifically, the i^{th} element of the vector $\Delta\hat{\mathbf{E}}$, $\Delta\hat{E}_i$, is the value of the current injected into node i by the grounded current source at i; similar interpretations can be found for voltage sources.

Regardless of which perturbation is being examined, if the bookkeeping information for the random walks is available from an initial analysis, it can be used to perform incremental analysis. We define a new right hand side, $\Delta\hat{\mathbf{b}}$:

$$(\Delta\hat{\mathbf{b}})_i = \Delta\hat{b}_i = \Delta\hat{E}_i/g_{ii}, \tag{4.7}$$

The result of incremental analysis can be obtained by substituting the new right hand side into Equation (4.3).

In case the original system was solved using random walks, the bookkeeping information is readily available; if not, by running a small set of random walks, a fast and less accurate approximation of the bookkeeping information, appropriate for the purposes of identifying the RoI, may be found. As mentioned earlier, the

perturbation can result in a change in the solution of some nodes, and if this change exceeds a user-specified threshold, the nodes are said to lie in the RoI. The change may be estimated using the bookkeeping information: the level of accuracy of this information determines the accuracy of the RoI. If approximate bookkeeping information is used, a safety margin may be used to obtain a pessimistic RoI.

Note that the RoI is fed to a solver that works with a smaller system of equations, assuming that all nodes outside the RoI are at their unperturbed values. Therefore, there is a trade-off between the computation time required to find the exact RoI, and that required to evaluate the reduced system. Since the runtime of random walks varies quadratically with the accuracy requirement [32], it is preferable to find a more approximate RoI, at the expense of spending slightly more CPU time to evaluate the disturbance in the RoI.

4.2.2 The Concept of $\hat{Z}$

Since the perturbed system has the same left hand side matrix as the original system, it may use the same bookkeeping matrices, Y and Z. From Equations (4.3), (4.6), and (4.7):

$$\Delta \mathbf{V} = Y \Delta \mathbf{V} + Z \Delta \hat{\mathbf{b}} \tag{4.8}$$

Let us consider the case where we use the random walk method to solve for the first node in the power grid. Since the bookkeeping record maintains Y, Z and $\hat{\mathbf{b}}$, in case of an incremental change to the network, it is easy to infer that

$$\Delta v_1 = \sum_{j=1}^{N} z_{1j} \Delta \hat{b}_j \tag{4.9}$$

Therefore, if there is a single perturbation in the circuit, its effect at any node can be computed in $O(1)$ time.

In the solution for node $i, i \geq 2$, the previously solved nodes $j < i$ are treated as home nodes. The advantage of this is that it allows information from previous walks to be reused, so that a walk that encounters node j automatically leverages

all the walks that were run from j in earlier steps. Performing a similar analysis, we find that:

$$\Delta v_i \;=\; \sum_{j=1}^{i-1} y_{ij}\Delta v_j + \sum_{j=i}^{N} z_{ij}\Delta \hat{b}_j, \tag{4.10}$$

The above procedure implies the need to find $\Delta v_j \;\forall\; j < i$. This could be a very wasteful computation, especially for high values of i, since many of these nodes may lie outside the RoI, and may not need any computation. Note that the ordering of nodes during bookkeeping is fixed, and since potentially any part of the power grid may need incremental analysis, any fixed ordering may incur such wasteful computations.

We can trace the origins of this problem to the fact that Y is a lower triangular matrix where walks terminate on previously calculated vertices; if the walks could be made to continue until they reach the original home nodes of the circuit, i.e., the V_{DD} nodes in a V_{DD} network, the scenario would be similar to Equation (4.9). In such a case, Equation (4.3) would become:

$$\Delta \mathbf{V} = \hat{Z}\Delta \hat{\mathbf{b}}, \tag{4.11}$$

where $\hat{Z}$ is no longer a triangular matrix, and $\Delta v_i = \hat{Z}_i \Delta \hat{\mathbf{b}} \;\forall\; i$ may be written for each node independently, where $\hat{Z}_i$ is the i^{th} row of $\hat{Z}$.

This is essentially the classical random walk solution, without bookkeeping [27]. A random walk solver that works with this $\hat{Z}$ matrix can be very inefficient, and it has been shown in [32] that the reuse of prior computations, by terminating walks at previously computed nodes, provides a substantial speedup. Therefore, from a practical standpoint, for full analysis, it is important to stay with the Y and Z matrices, but for incremental analysis, it is useful to "expand" this representation to reconstruct the rows of the $\hat{Z}$ matrix from the Y and Z matrices *on demand*.

4.2.3 The Incremental Solution

Before going into details of reconstructing $\hat{Z}$, let us interprete the information in the Y, Z, and $\hat{Z}$ matrices:

- Entry y_{ij} in the i^{th} row of Y represents the number of walks started from node i that ended at node $j < i$, divided by the total number of walks started from node i. Entry $y_{ij} \leq 1$ and can be interpreted as the probability that a walk starting at i ends at j.

- Entry z_{ij} in the i^{th} row of Z represents the fraction of walks started from i that pass through $j \geq i$.

- Entry $\hat{z}_{ij}$ in the i^{th} row of $\hat{Z}$ represents the fraction of walks started from i that pass through *any* node j.

From the above description, it is clear that the upper triangle and diagonal of $\hat{Z}$ and Z are identical. The lower triangle of $\hat{Z}$ is, in effect, an "expanded" version of the lower triangle of Y, where the walks that terminate at solved nodes are allowed to continue to the home nodes (this can also be proved formally using Equation (4.8)). Therefore, significant effort is required only to reconstruct the lower triangle of $\hat{Z}$.

Given that Y is a probability matrix, it has a spectral radius of less than 1. Therefore, Equation (4.8) yields:

$$\Delta\mathbf{V} = (I - Y)^{-1} Z \Delta\hat{\mathbf{b}} = (I + Y + Y^2 + Y^3 + \cdots) Z \Delta\hat{\mathbf{b}}, \tag{4.12}$$

and a classical result shows that this infinite sum converges. This can also be inferred from the above interpretations of the entries of Y as probabilities that lie between 0 and 1, so that for large k, $Y^k \to 0$.

Taking Equation (4.11) and comparing it with Equations (4.4) and (4.12), it is easily seen that:

$$\hat{Z} = (I - Y)^{-1} Z = Z + YZ + Y^2 Z + \cdots \tag{4.13}$$

Truncating this series at various points, each row, Z_i, of Z is defined by the following recurrence relation:

$$\begin{aligned}
\hat{Z}_i^{(0)} &= Z_i && (4.14)\\
\hat{Z}_i^{(1)} &= (Z + YZ)_i \\
&= Z_i + \sum_{j=1}^{i-1} y_{ij} Z_j && (4.15)\\
\hat{Z}_i^{(2)} &= (Z + YZ + Y^2 Z)_i \\
&= Z_i + \sum_{j=1}^{i-1} y_{ij} Z_j + \sum_{j=1}^{i-1} \sum_{k=1}^{j-1} y_{ij} y_{jk} Z_k && (4.16)\\
&\vdots
\end{aligned}$$

As the number of levels of substitution increases, the truncation error becomes smaller. This is consistent with the idea that increasing the levels of substitution involves nodes that are farther away from the perturbation: intuitively, as we go away from the perturbed region, its effect grows smaller. Hence, if the number of levels of substitution is sufficiently large, the last term can be truncated.

At each level of substitution q, we define the incremental change in Δv_k at that level to be $\delta v_k^{(q)}$, i.e.,

$$\delta v_k^{(q)} = (\Delta v_k^{(q)} - \Delta v_k^{(q-1)}), \tag{4.17}$$

where $\Delta v_k^{(q)} \triangleq 0 \; \forall \; q < 0$.

A key result that enables the recursive computation of the circuit response is shown next.

Theorem 5 If $v_i^{(l)}$ is the voltage response at node i computed using $\hat{Z}_i^{(l)}$, the l^{th}-level-truncation of $\hat{Z}_i$, then

$$\delta v_i^{(l+1)} = \sum_{j=1}^{i-1} y_{ij} \delta v_j^{(l)}, \tag{4.18}$$

Proof: Truncating Equation (4.12) at the k^{th} power of Y, we have:

$$\Delta \mathbf{V}^{(k)} = (I - Y)^{-1} Z \Delta \hat{\mathbf{b}} = (Z + YZ + \cdots + Y^{(k)} Z) \Delta \hat{\mathbf{b}} \tag{4.19}$$

This implies that

$$\delta \mathbf{V}^{(l)} = (\Delta \mathbf{V}^{(l)} - \Delta \mathbf{V}^{(l-1)}) = Y^{(l)} Z \Delta \hat{\mathbf{b}} \tag{4.20}$$

Therefore, $\delta \mathbf{V}^{(l+1)} = Y \delta \mathbf{V}^{(l)}$, and the result follows from this. □

This implies a simple approach for finding the incremental solution. Rewriting Equation (4.18) as:

$$\Delta v_i^{(l+1)} = \Delta v_i^{(l)} + \sum_{j=1}^{i-1} y_{ij} \delta v_j^{(l)}, \tag{4.21}$$

we can see that the right hand side depends entirely on $\Delta v_i^{(q)}, q \leq l$, implying that $\delta v_i^{(l+1)}$ can be computed recursively. It can easily be shown that the complexity of each level of substitution is $O(n \cdot \mid \text{RoI} \mid)$, where $\mid \text{RoI} \mid$ is the size of the RoI.

4.2.4 Efficient Computation Techniques

In the above recursive computation, the voltage value for some nodes may converge easily, using a small number of substitution levels, while for other nodes, it may be necessary to employ a much higher number of substitution levels. The above approach is thus constrained by the weakest link, i.e., the substitution will continue until all nodes are computed sufficiently accurately. Instead, we *adaptively* terminate this propagation when an error tolerance, ϵ, is met,

$$e_{v_i^{(l)}} = \left| \delta v_i^{(l)} \right| < \epsilon \tag{4.22}$$

then we stop further substitutions into v_i. In other words, we set $\Delta v_i^{(m+1)} = \Delta v_i^{(m)} \; \forall \; m \geq l$. As a result, in Equation (4.18), $\delta v_j^{(l)} = 0$ for $l > l_j$, and the amount of computation is reduced since our data structure ignores any multiplications where one operand is zero.

We can rewrite Equation (4.22) as $\left|\sum_{j=1}^{i-1} y_{ij}\delta v_j^{(l-1)}\right| < \epsilon$, i.e.,

$$\left|\sum_{j\in S} y_{ij}\delta v_j^{(l-1)} + \sum_{j\in \bar{S}} y_{ij}\delta v_j^{(l-1)}\right| < \epsilon \tag{4.23}$$

where S is the set of all the neighbors of node i that have previously converged, and $\bar{S}$ is the set of all other neighbors of node i. The first term evaluates to zero, and convergence criterion effectively becomes $\left|\sum_{j\in \bar{S}} y_{ij}\delta v_j^{(l-1)}\right| < \epsilon$. The error of each step of this approximation is upper-bounded by the accumulated errors in S, and can be written as

$$\left|\sum_{j\in S} y_{ij}\delta v_j^{(l-1)}\right| < \epsilon$$

The above inequality follows from the facts that (a) by construction, $\delta v_j^{(l-1)} < \epsilon$ for $j \in S$, (b) $\sum_{j\in S} y_{ij} \leq 1$, and (c) $y_{ij} \geq 0 \; \forall \; i, j$.

4.2.5 Incremental Solver Algorithm

Algorithm 1 shows the pseudo-code for recursively finding the solution of a single node, going up to a given number levels of substitution. In this algorithm, which implements the recursions of Equation (4.18), once the solution to a node of some level of substitution is found, it is stored for consequent calculations.

In practice, the random walk solver provides a reasonable trade-off between accuracy and speed [32], but may not be accurate enough to capture small changes due to an incremental analysis exactly. We assume here that the initial solution that is provided does *not* come from a random walk solver. Therefore, we initially use the random walk solver with moderate accuracy to build the bookkeeping information; the typical accuracy used is empirically chosen to be one-third of V_{DD}. Note that while this accuracy limit seems high, this is only an upper bound on the accuracy; the average error is much smaller. This is found to be sufficient for the purposes of identifying the RoI. This operation needs to be performed

Algorithm 1 Single Node Solution Using Bookkeeping Info

Input: Bookkeeping Info (Y and Z matrices), RHS (b), and tolerance
Output: Node solution of the given level of indirection
function find-x(node, level)
if node.$isDone$ **OR** node.$level \geq$ level **then**
  return
end if
if level = 1 **then**
  node.Δv[level] $\leftarrow Z$[node.$index$]$^T b$
else
  home-neighbors $\leftarrow$ {non-zero items in Y on the same column as node}
  for all home-neighbors **do**
    neighbors-$\Delta v \leftarrow$ find $- x$(neighbor, level-1)
    Δv[level] $\leftarrow Y$[node.$index$][home-neighbors]^{T}neighbors $- \Delta v$
    node.$level \leftarrow$ level
    if abs(node.Δv[level] $<$ tolerance) **then**
      node.$isDone \leftarrow$ TRUE
    end if
  end for
end if

only once for a system, and any number of incremental changes can be made to the system using this bookkeeping information. *The high accuracy bound implies that the RoI is effective even under large approximations, e.g., after successive incremental changes.*

The algorithm uses a node queue, Q, to keep track of all the potentially affected nodes. It first adds all the nodes in the perturbed region to Q and starts processing them using the first level of substitution in Equation (4.18). Next, it successively increases the levels of substitution, i.e., goes to successive levels of recursion, until the stopping criterion described above is met. As stated earlier, we account for approximate bookkeeping by allowing a pessimistic RoI, found by multiplying the tolerance in the stopping criterion by a safety factor $s < 1$.

The last step of *refinement* involves refining the solution using an exact solver. All nodes outside the RoI are kept at their previous values, and a smaller |RoI| $\times$ |RoI| system of equations, as illustrated in Figure 1.2, is solved to obtain an accurate solution. Since this is a small system, any direct or iterative solver can be used; in this work we use LAPACK [57] for this purpose. A pessimistic RoI contains all the nodes with substantial change and the computational expense is passed on to the exact solver that operates on the small RoI region, whose size is $\ll n$.

4.3 Experimental Results

Our algorithm is tested and compared on a UNIX machine with 2GHz processor and 2GB RAM, and applied to 12 benchmarks summarized in Table 6.1. This table represents the statistics of the LHS matrix of the DC analysis; the role of the RHS is much less significant for perturbation analysis. For each matrix, we list the size, i.e., the number of unknowns, the number of non-zeros in G, and the average number of non-zeros per row, as a sparsity metric. The value of V_{DD} is 1.2V.

To demonstrate the accuracy of the RoI found by the proposed algorithm, we

Table 4.1: Benchmark Details: Statistics of the LHS Matrix

Name	Size	Number of Nonzeros	Average Nonzeros/Row
c1	16194	98030	6.1
c2	26300	165810	6.3
c3	29551	178345	6.0
c4	34784	203322	5.8
c5	37403	251535	6.7
c6	42409	255747	6.0
c7	58866	352310	6.0
c8	65938	412448	6.3
c9	69351	438985	6.3
c10	93194	593908	6.4
c11	92327	553245	6.0
c12	95303	635727	6.7

insert various perturbations to the benchmarks and find the corresponding RoI using the proposed algorithm as well as an exact direct solver. The *exact RoI* is the set of nodes for which the exact solution is perturbed by more than a specified tolerance. The first measure of the quality of the RoI found by our approach is the number of *undetected nodes*: these are nodes that belong to the exact RoI, but are not listed in the RoI found by our method. The second measure is the exact amount of the change in the solution of the undetected nodes, as computed by the exact solver.

The circuits are perturbed in two ways: for a randomly selected node and a group of its neighbors, (i) the RHS vector, $\mathbf{b}$, is multiplied by the given perturbation value to obtain $\Delta\hat{\mathbf{b}}$ in Equation (4.7), and (ii) the conductances to its neighbors are multiplied by a perturbation: as shown earlier, this is used to obtain a perturbation to the RHS as $\Delta\hat{\mathbf{b}}$ in Equation (4.7). In either of these, the amount of perturbation is chosen randomly, with a uniform distribution between 0 and 0.1.

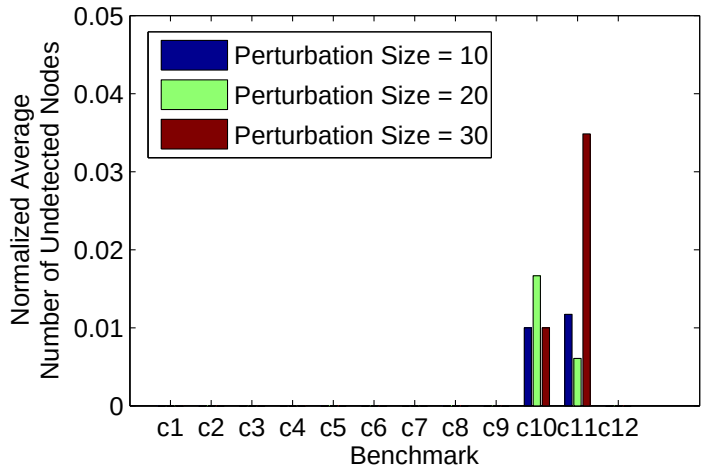
For the range of perturbation in our experiments, the size of the RoI is no

more than about 2% of the total circuit size, and depending on the change, and is often less. The empirically-chosen safety factor parameter s in the algorithm is set to 0.1, to compensate for approximation error and to generate a pessimistic RoI. Approximation errors could arise from the approximate nature of the initial random walk solver used to obtain the bookkeeping information, or from the stopping criterion used to terminate the substitutions.

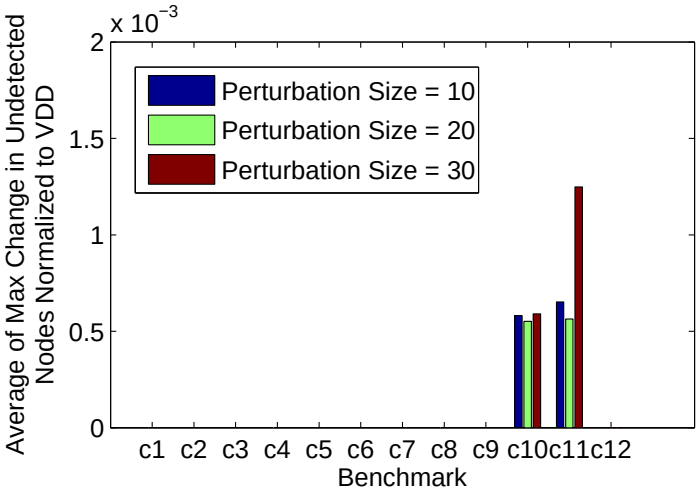
Figure 4.1(a) shows the number of undetected nodes, normalized to the exact RoI size, versus the size of the perturbed region. The numbers shown in all plots are averaged over 20 different sets of perturbations. This figure indicates that the number of undetected nodes is equal to zero for benchmarks $c1$ to $c9$, and $c12$, and it goes up as the size of the perturbation region is increased, which is consistent with the idea that this method is intended for small changes. The next natural issue to study is the error for these undetected nodes, and this is plotted in Figure 4.1(b). It is found that this error is below 1% of V_{DD}, indicating that even the nodes that lie within the RoI, but are not detected, see an insignificant degradation in accuracy.

Next, we examine the accuracy of the solution within the RoI. We see this in two steps: Figure 4.2(a) shows the normalized error of the nodes within the RoI for different perturbation region sizes, when the perturbation is applied to the RHS. These figures suggest that although the accuracy of the bookkeeping information used to find the estimated solution of the nodes, is not high (i.e. $V_{DD}/3$), the amount of error in the estimated change in the solution is less than 3% of V_{DD}. If we feed this solution to the refinement stage, where we solve a much smaller system of size $\mid \text{RoI} \mid \times \mid \text{RoI} \mid$, the solution becomes more accurate; this can be seen by comparing the results in Figures 4.2(a) and 4.2(b). Similarly, Figures 4.3(a) and 4.3(b) demonstrate the effectiveness of the algorithm for the case that the LHS is perturbed, representing perturbation to the conductances of the power network.

To demonstrate the computational efficiency of the proposed algorithm, we compare it in Table 6.2 against a very efficient public-domain iterative solver that

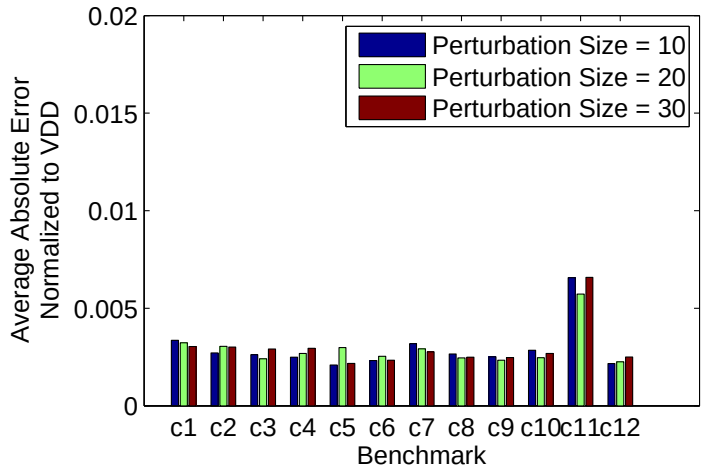


(a) Normalized number of undetected nodes

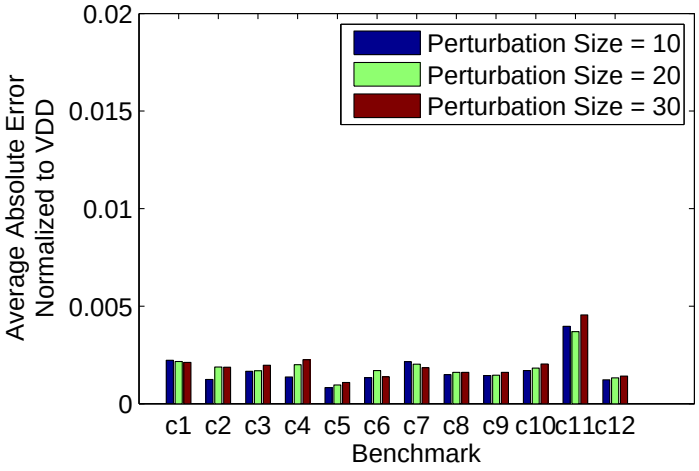


(b) Maximum change in the voltage of undetected nodes

Figure 4.1: Number of undetected nodes, normalized to the exact RoI size, and the maximum change in their voltage, for various perturbation region sizes (tolerance $= 1\%V_{DD}$, perturbation value uniformly distributed in $(0, 0.1)$, averaged over 20 perturbations).

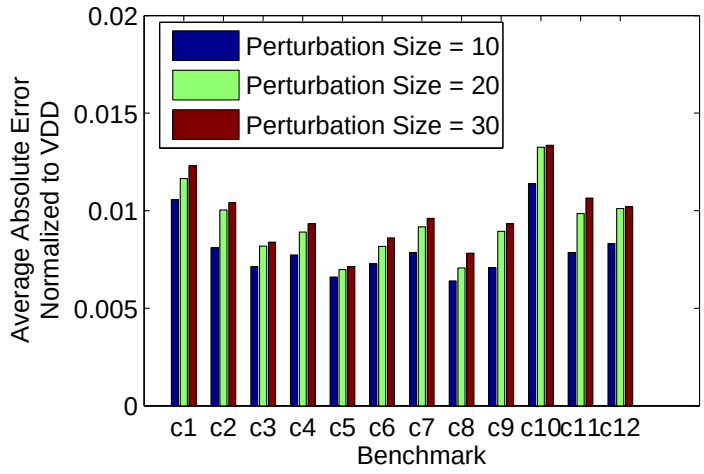


(a) Before refinement

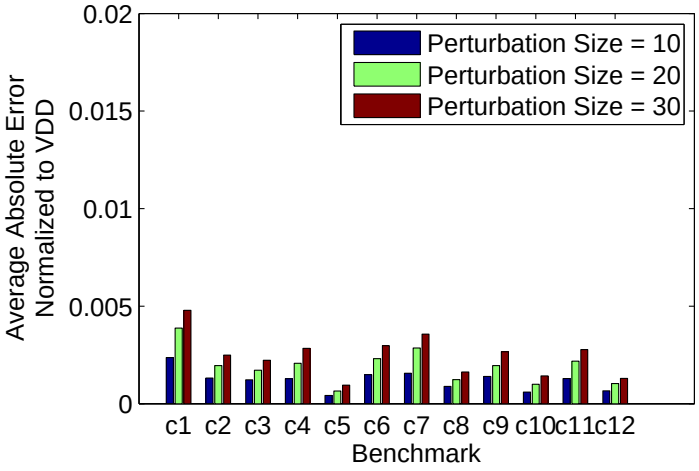


(b) After refinement

Figure 4.2: Absolute error of the solution, normalized to V_{DD} and averaged over 20 perturbations, for nodes within the RoI before and after the refinement phase. The perturbation applied to the RHS (tolerance = 1% V_{DD}) is uniformly distributed in $(0, 0.1)$).



(a) Before refinement



(b) After refinement

Figure 4.3: Average absolute error of the solution, normalized to V_{DD}, for nodes within the RoI before and after the refinement phase, perturbation applied to the LHS (tolerance = 1% V_{DD}, uniformly distributed perturbation in $(0, 0.1)$, averaged over 20 perturbations)

Table 4.2: Runtime comparisons (tolerance = 1% V_{DD}, perturbation region size = 20, 30% perturbation)

	[33] (sec)	Book-keeping (sec)	RoI (sec)	Refine-ment (sec)	10 runs (sec)	20 runs (sec)
c1	0.29	0.99	0.08	0.01	1.87 (1.6×)	2.75 (2.1×)
c2	0.57	2.04	0.17	0.01	3.85 (1.5×)	5.67 (2.0×)
c3	0.59	2.23	0.19	0.01	4.28 (1.4×)	6.32 (1.9×)
c4	0.70	2.98	0.21	0.01	5.19 (1.4×)	7.41 (1.9×)
c5	0.84	2.86	0.23	0.02	5.27 (1.6×)	7.69 (2.2×)
c6	1.23	4.68	0.31	0.02	7.98 (1.5×)	11.28 (2.2×)
c7	1.88	7.02	0.40	0.02	11.25 (1.7×)	15.48 (2.4×)
c8	2.72	8.29	0.48	0.03	13.31 (2.0×)	18.33 (3.0×)
c9	2.92	9.16	0.44	0.03	13.77 (2.1×)	18.39 (3.2×)
c10	3.83	11.90	0.59	0.04	18.14 (2.1×)	24.38 (3.1×)
c11	3.79	12.97	0.72	0.04	20.53 (1.8×)	28.09 (2.7×)
c12	4.16	13.16	0.68	0.04	20.35 (2.0×)	27.53 (3.0×)
				Average:	1.7×	2.5×

uses an approximate conditioner based on random walks [33] that has been shown to be up to nearly an order of magnitude faster than other comparable solvers, using identical solver tolerances. The first column shows the runtime of the solver in [33], and the remaining columns are related to our incremental solver. The bookkeeping time is the time required for the initial approximate random walks used to find the bookkeeping information, and is a fixed expense that must be computed once for the unperturbed circuit, and can be used by all perturbations henceforth. The runtime of the substitution procedure used to find the RoI is listed next (RoI), followed by the runtime of the refinement phase, when the RoI is solved using an accurate solver. We expect this solver to be used to solve what-if scenarios; therefore, it is reasonable to report the runtime for multiple incremental analyses; note that *all* incremental analyses, anywhere in the circuit, use the same bookkeeping information. The last two columns report the run times for 10 and 20 incremental analyses, with the speedups in parentheses. For example, for 20

runs, the speedup over [33] for c1 is $20 \times 0.29/2.75 = 2.1\times$ (Note that [33] is reported to be $5 - 10\times$ faster than a state of the art solver.). The runtime for k incremental analyses includes the bookkeeping time, plus k times the sum of times required to find the RoI and the time for the refinement phase. As the number of incremental analyses increases, the cost of bookkeeping is amortized. Alternatively, if the bookkeeping information is already available since the original solution used random walks (e.g., for the preconditioner in [33]), this expense disappears; the speedups are correspondingly greater: e.g., they approach $20\times$ for c12.

Table 6.2 also shows that as the size of the benchmark increases, the amount of speedup increases, which is due to the fact that size of RoI is independent of the size of the circuit and only depends on its topology. Hence, as the size of benchmark increases the time for the solver in [33] increases accordingly while the time for the incremental solver remains the same.

Chapter 5

Backward Incremental Random Walk Solver

5.1 Incremental Solver Method

We now propose an efficient incremental solver based on backward random walks in the framework of power network analysis. Our incremental solver proceeds under the reasonable assumption that the solution to the unperturbed system of equations, $G\mathbf{V} = \mathbf{E}$, is known. When the design is perturbed, it may result in a change in either G or $\mathbf{E}$, resulting in the new relationship:

$$(G + \Delta G)(\mathbf{V} + \Delta \mathbf{V}) = \mathbf{E} + \Delta \mathbf{E} \tag{5.1}$$

where ΔG models the change in the LHS, $\Delta \mathbf{E}$ models the change in the RHS, and $\Delta \mathbf{V}$ is the change in the solution caused by the perturbation. This equation can be rewritten as:

$$\begin{aligned} (G + \Delta G)\Delta \mathbf{V} &= \Delta \mathbf{E} - \Delta G \mathbf{V} \\ G_{\text{eff}} \Delta \mathbf{V} &= \Delta \mathbf{E}_{\text{eff}} \end{aligned} \tag{5.2}$$

where $\Delta \mathbf{E}_{\text{eff}} = \Delta \mathbf{E} - \Delta G \mathbf{V}$ is the effective change in the RHS and $G_{\text{eff}} = G + \Delta G$ is the total perturbed LHS.

Note that in contrast with [55], where the second order term, $\Delta G \Delta \mathbf{V}$, is ignored, Equation (5.2) captures any perturbation to the system without any approximation. Moreover, this method does not make any assumptions regarding the nature of the perturbation as long as the number of nodes of the network is fixed. Further, since the perturbed system models a power grid, the LHS is diagonally dominant, and all of the off-diagonals are less than or equal zero, therefore it can be solved using random walks.

The steps followed by the proposed incremental solver are:

Step 1: Solve Equation (5.2) using backward random walks.

Step 2: Find the RoI using the computed solution.

Step 3: Refine the solution for the nodes within RoI.

Step 1: The first step of the incremental solution involves finding the columns of G_{eff}^{-1} corresponding to nonzeros of $\Delta \mathbf{E}_{\text{eff}}$. For a small perturbation, most of the LHS matrix, G, and the RHS vector, $\mathbf{E}$, will be unchanged and therefore most of the entries of $\Delta \mathbf{E}_{\text{eff}}$ are zero. And if there are η nonzeros in $\Delta \mathbf{E}_{\text{eff}}$, we have:

$$\eta \ll |\text{RoI}| \ll N \tag{5.3}$$

where $|\text{RoI}|$ and N are the size of RoI and Equation (5.2), respectively. As a result, *only a few columns* of G_{eff}^{-1} must be computed, corresponding to the η nonzeros of $\Delta \mathbf{E}_{\text{eff}}$. Then, $\Delta \mathbf{V}$ is given by:

$$\Delta \mathbf{V} = \left[G_{\text{eff}}^{-1}\right]_{N \times \eta} \left[\Delta \mathbf{E}_{\text{eff}}\right]_{\eta \times 1} \tag{5.4}$$

where $\left[\Delta \mathbf{E}_{\text{eff}}\right]_{\eta \times 1}$ denotes η nonzeros of $\Delta \mathbf{E}_{\text{eff}}$, and $\left[G_{\text{eff}}^{-1}\right]_{N \times \eta}$ denotes the columns of G_{eff}^{-1} corresponding to these nonzeros.

This step of the algorithm relies on the fact that random walk solver is capable of finding an estimate of the solution efficiently with moderate accuracy, just

enough to identify the RoI that corresponds to the nodes that are significantly affected.

As discussed in Section 3.7, the accuracy of the random walk solution is proportional to the square root of the number of walks, and it is not efficient for very high accuracies. Therefore, this step merely feeds Step 2, which finds the RoI based on this solution, and a precise solution is found in Step 3.

In this work a relative tolerance of 30% is used for the random walk solver. This means that the relative error of $G_{\text{eff}}^{-1} \times (\Delta \mathbf{E}_{\text{eff}})_j$ is less than or equal to 30% with a confidence of $\alpha = 99\%$, where $(\Delta \mathbf{E}_{\text{eff}})_j$ denotes the vector of $\Delta \mathbf{E}_{\text{eff}}$ where all of its elements are set to zero except for the j^{th} one.

Step 2: The second step of the incremental solver compares the computed estimate of $\Delta \mathbf{V}$ given by Equation (5.4), to a user-defined tolerance, tol, to determine the RoI. Specifically, node j is said to lie within the RoI if $\Delta V_j > tol$.

In order to account for the potential errors in the estimated solution, a safety margin (i.e., $s < 1$) is used to get a pessimistic RoI where the criterion for putting node j in this pessimistic RoI is $\Delta V_j > s \times tol$. A pessimistic RoI contains all of the nodes with potentially substantial change and the computational expense is passed on to the exact solver that operates on the small RoI region, whose size is $\ll N$. In this work, the safety margin is chosen empirically to be $s = 1/3$.

Step 3: The last step of *refinement* involves the application of an exact solver. In this step, we leverage the initial solution of the network, $\mathbf{V}$, the estimated changes computed using random walks, $\Delta \mathbf{V}$, and the RoI. Reordering Equation (5.2) according to RoI we have:

$$\begin{bmatrix} G_{\text{eff}}^{(in,in)} & G_{\text{eff}}^{(in,out)} \\ G_{\text{eff}}^{(out,in)} & G_{\text{eff}}^{(out,out)} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{V}^{(in)} \\ \Delta \mathbf{V}^{(out)} \end{bmatrix} = \begin{bmatrix} \Delta \mathbf{E}_{\text{eff}}^{(in)} \\ \Delta \mathbf{E}_{\text{eff}}^{(out)} \end{bmatrix} \tag{5.5}$$

where the superscripts in and out denote the nodes inside and outside of the RoI, respectively.

Although $\Delta \mathbf{V}^{(out)}$ should be set to 0 from the definition of the RoI, in practice, we find that it is more appropriate to use the constant (but small) value of $\Delta \mathbf{V}^{(out)}$ from Step 1, which enables us to be less conservative with the RoI. Therefore, we

solve the above equation for $\Delta\mathbf{V}^{(in)}$, keeping the $\Delta\mathbf{V}^{(out)}$ unchanged from Step 1, by solving:

$$G_{\text{eff}}^{(in,in)}\Delta\mathbf{V}_r^{(in)} = \Delta\mathbf{E}_{\text{eff}}^{(in)} - G_{\text{eff}}^{(in,out)}\Delta\mathbf{V}^{(out)} \tag{5.6}$$

where $\Delta\mathbf{V}_r^{(in)}$ is the refined solution of the nodes within RoI. The size of this equation, |RoI|, is significantly smaller than N as illustrated in Figure 1.2. For this small system, any direct or iterative solver can be used; here, we use LAPACK [57]. The total solution is then computed by adding $\Delta\mathbf{V}_r^{(in)}$ for the nodes inside the RoI, and $\Delta\mathbf{V}^{(out)}$ for nodes out of RoI, to the initial solution $\mathbf{V}$.

5.2 Experimental Results

To demonstrate the performance of the proposed backward random walk incremental solver, this solver is implemented in C++ and is compared against the Hybrid solver of [33] and the forward random walk solver of [55] on the original IBM Power Grid (IBMPG) benchmarks [50] Next, the effectiveness of our solver on asymmetrical equations is examined using VCCS model of IBMPG benchmarks described in Appendix A. The reason for showing results on the original benchmarks is because this enables a comparison with the Hybrid solver and the forward random walk solver, which cannot handle asymmetry. Moreover, as we will see, the runtime of the backward walk solver on the original and modified PG benchmarks is very similar. The experiments are conducted on a UNIX machine with 8GB RAM and 2.5GHz processor, where V_{dd} is set to 1.8V, as in the benchmarks.

5.2.1 Performance of the Backward Random Walk Solver on Symmetric LHS Matrices

The performance of the proposed incremental solver is demonstrated based on several metrics indicating the quality of the found RoI compared to *exact RoI* and precision of the solution of the nodes within the RoI. The *exact RoI*, found

using a direct solver, is the set of nodes for which the exact solution is perturbed by more than a specified tolerance, 1% of the V_{dd} here. The metrics used for evaluating the quality of the computed RoI are:

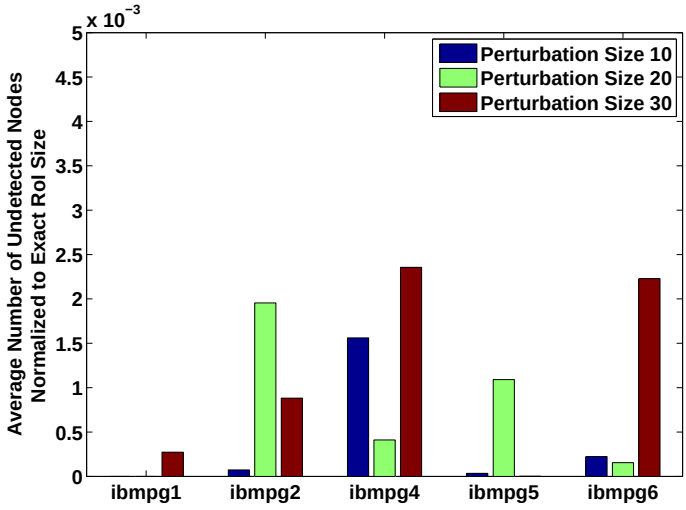
- The number of *undetected nodes*, i.e., the nodes that belong to the exact RoI, but are not listed in the RoI found by our method.
- *Error magnitude* in the solution of the undetected nodes.

A comparison is performed over several random perturbations of various *perturbation amounts*, i.e., the magnitudes of change relative to the solution of the node, and various *perturbation sizes*, i.e., the number of nodes perturbed. To generate a perturbed benchmark, one node is chosen randomly from the original benchmark and the conductances, g_{ij}'s, and the current loads, e_i's, of that node and a number of its neighboring nodes are modified. This approach models the locality property of a real perturbation.

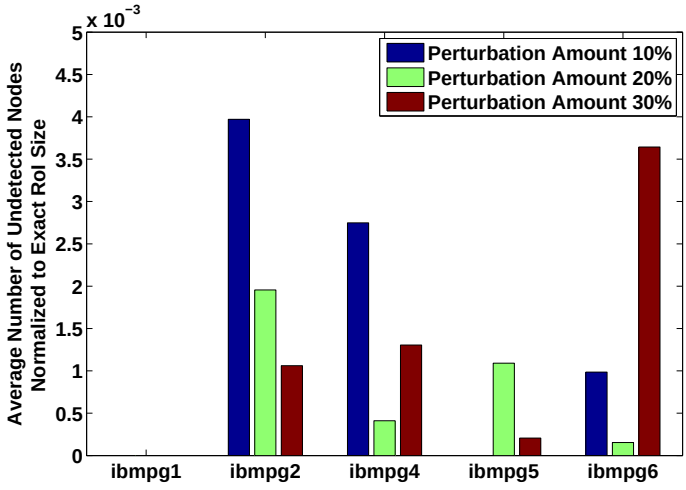
For the range of perturbations in our experiments, the maximum and the average size of the RoI is 17.1% and 2.9% of the total circuit size, respectively, which is significantly larger perturbations than the ones used in prior work [55]. The empirically-chosen safety factor parameter in the algorithm is set to 1/3 to compensate for approximation error and to generate a pessimistic RoI.

Figure 5.1(a) shows the number of undetected nodes, normalized to the exact RoI size, versus the size of the perturbed region, for perturbation amount of 20%, for various perturbation sizes. Similarly, the normalized number of undetected nodes versus the amount of perturbation, for perturbation size of 20, for various perturbation amounts, is shown in Figure 5.1(b). The numbers shown in all plots are averaged over 10 different sets of perturbations. These figures indicate that the number of undetected nodes is less than 0.5% the size of the corresponding exact RoI. Note that the backward solver has been able to identify all of the nodes within RoI for some of the perturbations.

The next issue is the significance of the undetected nodes. The errors caused by the undetected nodes are the errors of the estimated solution from random walks.

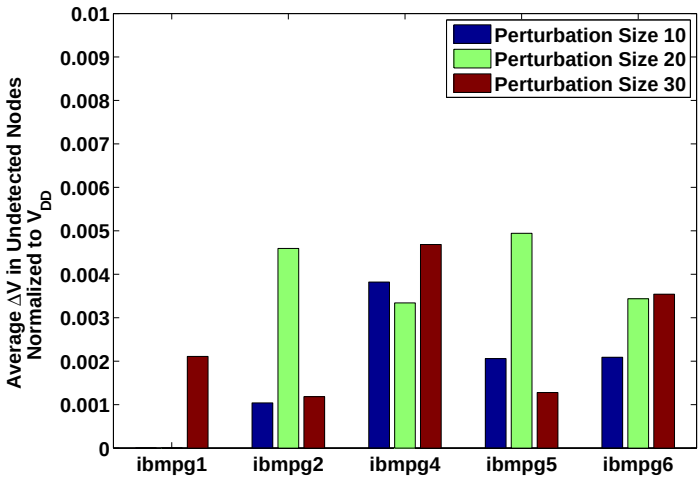


(a) Perturbation amount = 20%

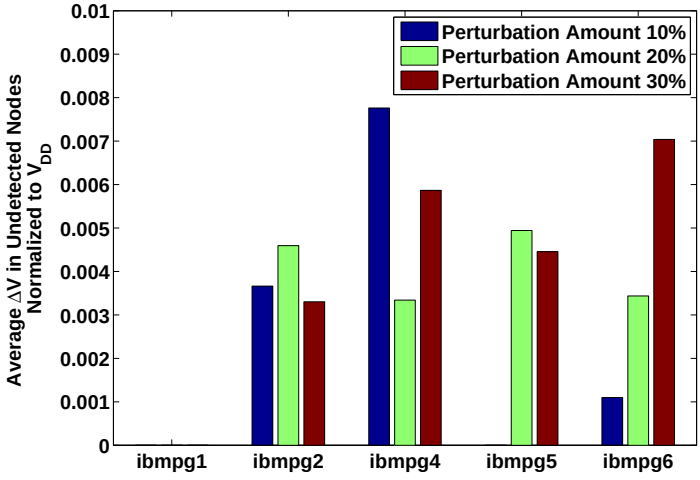


(b) Perturbation size = 20

Figure 5.1: Number of undetected nodes, normalized to the exact RoI size (tolerance = $1\%V_{dd}$, averaged over 10 perturbations).



(a) Perturbation amount = 20%



(b) Perturbation size = 20

Figure 5.2: Average change in the voltage of undetected nodes (tolerance = $1\%V_{dd}$, averaged over 10 perturbations).

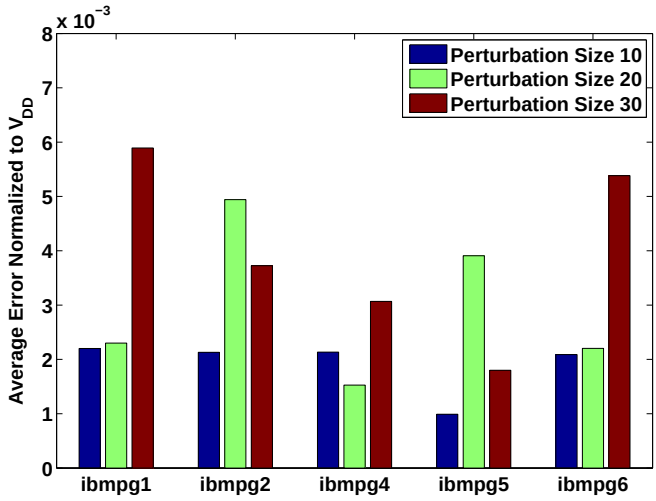
The average of this error versus the perturbation size for a perturbation amount of 20% is plotted in Figure 5.2(a) and this error versus perturbation amount for a perturbation size of 20 is shown in Figure 5.2(b).

It is found that the average of this error is less than 1% of V_{dd}, indicating that even the nodes that lie within the RoI, but remain undetected, see an insignificant degradation in accuracy, and this is due to the fact that the estimated solution given by random walks is sufficiently accurate for the purposes of detecting the RoI.

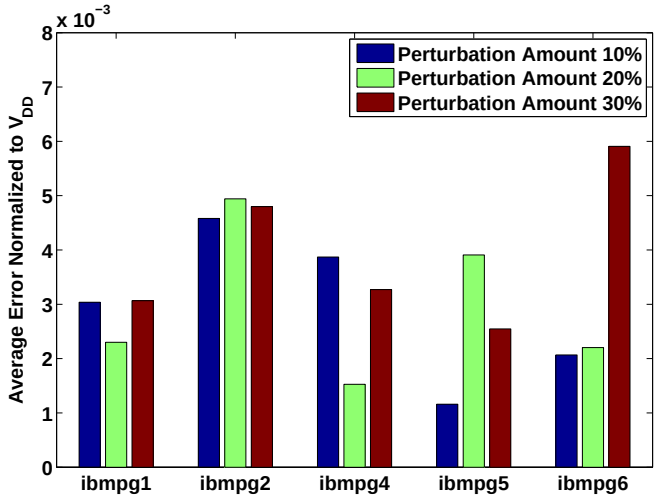
Due to the stochastic nature of random walks, different runs result in slightly different estimated solutions and hence different computed RoIs. As discussed in Section 3.7, the differences remain less than the given tolerance (i.e., $1\%V_{dd}$), with a confidence of α. Due to this effect, in Figure 5.2(a) for instance, the error for perturbation size of 20 was more than that for a perturbation size of 30 for benchmark *ibmpg*2. For the similar reason, in Figure 5.2(b) perturbation amount of 20% resulted in less error than perturbation amount of 10% for *ibmpg*4. In Figure 5.1 for the same reason some counterintuitive behavior is observed.

In this chapter we exclude the results of *ibmpg*3 because of the specific structure of the LHS of this benchmark (similar observations have been made by other authors). This benchmark models the power network of a circuit with very few external V_{dd} and ground connections, less than 0.1% of the nodes. As a result, the corresponding random walk game has very few home nodes, resulting in very long walks and hence large runtimes. For such circuits, conventional solvers should be used instead of random walk solvers.

Next, we examine the accuracy of the solution within the RoI. Figure 5.3 shows the average error of the nodes within the RoI normalized to V_{dd}, for different perturbation sizes and perturbation amounts. This figure suggests that although the relative tolerance of the random walk solver is set to 30%, the average error of the estimated solution is less than 0.7% of V_{dd}. Feeding the solution to the refinement stage, where we solve a much smaller system of size $|\text{RoI}| \times |\text{RoI}|$, the solution becomes more accurate; this can be seen by comparing the results in

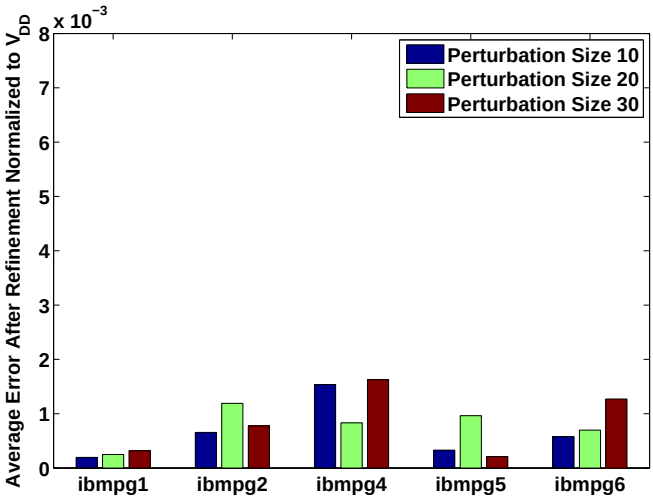


(a) Perturbation amount = 20%

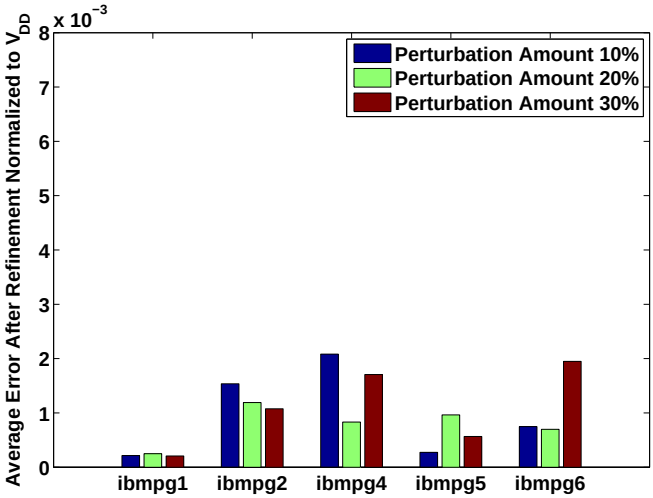


(b) Perturbation size = 20

Figure 5.3: Absolute error of nodes within the RoI before the refinement phase, normalized to V_{dd} and averaged over 10 perturbations. (tolerance = 1% of V_{dd}).



(a) Perturbation amount = 20%



(b) Perturbation size = 20

Figure 5.4: Absolute error of nodes within the RoI after the refinement phase, normalized to V_{dd} and averaged over 10 perturbations. (tolerance = 1% of V_{dd}).

Figures 5.3 and 5.4.

Table 5.1: Runtimes (tolerance = 1% of V_{dd}, perturbation region size = 30, perturbation amount = 20%).

	Equation Size (N)	Hybrid (sec)	Random Walk (sec)	Refinement (sec)	Incremental Total (sec) [Speedup]
ibmpg1	30,638	0.17	0.111	0.001	0.112 [1.5×]
ibmpg2	127,238	2.14	0.743	0.058	0.801 [2.7×]
ibmpg4	953,583	26.37	1.673	0.274	1.947 [13.5 ×]
ibmpg5	1,079,310	18.74	2.108	0.122	2.230 [8.4×]
ibmpg6	1,670,494	37.23	2.898	0.200	3.098 [12.0 ×]
				Average Speedup:	7.6×

Table 6.2 compares the runtime of our proposed incremental solver with the Hybrid Solver of [33], which is an efficient public-domain iterative solver that uses a preconditioner based on random walks that has been shown to be faster than other comparable solvers, using identical solver tolerances. The first column shows N, the matrix dimension for each benchmark. The second column shows the runtime of the Hybrid Solver and the remaining columns are related to our incremental solver. It can be seen that the refinement phase is extremely fast and takes only a small fraction of the total runtime, and that the total speed up of the proposed incremental solver for perturbation size of 30 and perturbation amount of 20% is significant: an average of 7.6× and a maximum of 13.5×.

Moreover, broadly speaking, as the system size increases the benefit of using the incremental solver becomes more significant. The intuitive reason for this is that for benchmarks of similar topology, a perturbation of the same size and amount results in a RoI of almost the same size, which requires almost the same amount of effort for the random walk solver to find the RoI and the exact solver to refine the solution. In fact, the speedup depends on the structure of the equation as well, i.e., its density, condition number, and the number of home nodes in its corresponding random walk game.

5.2.2 Backward Solver on Asymmetric LHS Matrices

As discussed in Section 2, the use of VCCS in modeling power grids makes it possible to account for the effect of supply voltage drop on the current load which results in a more accurate power grid model but equations with *asymmetrical* LHS. In this section we demonstrate that the backward solver is capable of handling these asymmetrical equations as efficient as the symmetrical equations while the forward solver of [55] is capable of solving the symmetrical equations only.

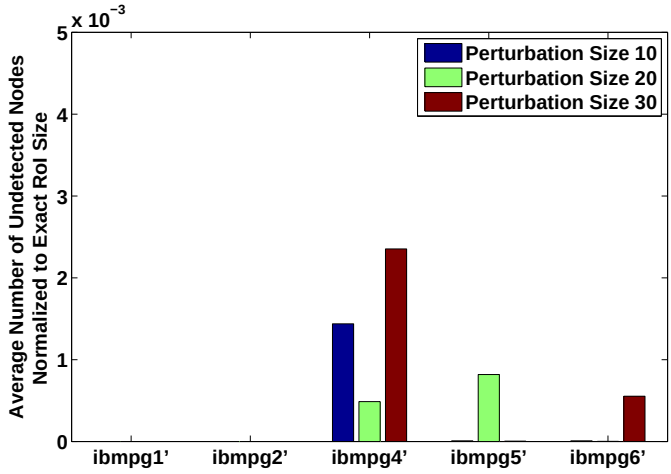
We have created variations of the ibmpg benchmarks [50] using the procedure of Section 2.2 where the detailed extracted circuit of the benchmarks are obtained as described in Appendix A. These benchmarks are referred to as the **ibmpg'** and the circuit with each number represent the corresponding circuit with the same number in the original benchmark suite.

Figure 5.5 is analogous to Figure 5.1 and shows the average number of undetected nodes, normalized to the exact RoI size, versus the size of the perturbed region, and amount of perturbation. Comparing these figures, it can be seen that the number of undetected nodes is less than 0.5% the size of the corresponding exact RoI for both symmetrical and asymmetrical power grid models.

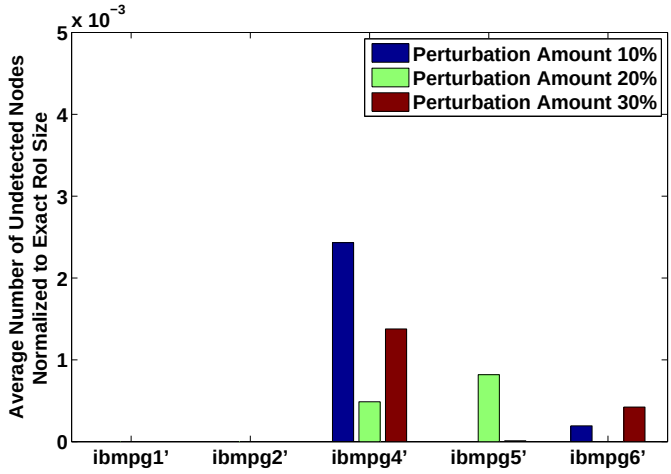
The performance of the backward solver in terms of error and runtime is shown in Table 5.2 for symmetrical and asymmetrical power grid models. This table shows the absolute error of the solver after the refinement phase normalized to V_{dd} averaged over 10 random perturbations of region size of 30 and amount of 20% and their corresponding runtime. It is clear from this table that the backward solver performs equally well for both models.

5.2.3 Comparing Forward/Backward Solver Based Incremental Analysis

Finally in this section we compare the forward incremental solver of [55] with the proposed backward solver of this work over the *ibmpg* benchmarks. The work of [58] compares these solvers using the benchmarks of [55].



(a) Perturbation amount = 20%



(b) Perturbation size = 20

Figure 5.5: Number of undetected nodes, normalized to the exact RoI size (tolerance = $1\%V_{dd}$, averaged over 10 perturbations).

Table 5.2: Comparison of the error of the nodes within RoI after refinement phase and total runtime of backward solver on symmetrical and asymmetrical equations (Averaged over 10 random perturbations. perturbation region size = 30, perturbation amount = 20%, error is normalized to V_{dd}, tolerance = 1% of V_{dd}).

	Symmetrical model			Asymmetrical model	
	Error (Norm to V_{dd})	Runtime (sec)		Error (Norm to V_{dd})	Runtime (sec)
ibmpg1	3.18E-04	0.112	**ibmpg1'**	1.11E-04	0.111
ibmpg2	7.77E-04	0.801	**ibmpg2'**	8.62E-05	0.740
ibmpg4	1.63E-03	1.947	**ibmpg4'**	1.56E-03	1.673
ibmpg5	2.11E-04	2.230	**ibmpg5'**	1.50E-04	2.111
ibmpg6	1.27E-03	3.098	**ibmpg6'**	5.22E-04	2.897
Average	8.41E-04	1.638	**Average**	4.86E-04	1.506

Table 5.3 compares the forward and backward solver accuracy in terms of the quality of the computed RoI and error in the solution of the nodes within the RoI before and after refinement phase. The number of undetected nodes is normalized to the RoI size, and ΔV is normalized to V_{dd}. As this table suggests, on average, the number of undetected nodes of the backward solver is about 50× less than that of the forward solver and also the error in the undetected nodes is about 70× less. Moreover this tables shows that the error in the solution of the nodes within RoI, computed by backward solver, before the refinement phase is up to 12× and on average 9×, and after refinement phase up to 21× and on average 14× less than the forward solver.

In this section, comparisons are shown only for the *ibmpg1* and *ibmpg2* benchmarks since the forward solver implementation is unable to handle large benchmarks. Moreover, smaller perturbations are used here: due to recursive nature of the forward solver method, the forward solver cannot handle the large perturbations used in Section 5.2.1. The key difference between the backward and forward solver that makes the backward solver significantly more efficient is that

Table 5.3: Comparison of the accuracy of the Forward solver and Backward solver (Averaged over 10 random perturbations. perturbation amount = 5%, tolerance = 1% of V_{dd}).

Bench-mark/ Pert. size	Forward Incremental Solver				Backward Incremental Solver			
	Norm. Undet. Nodes	Norm. ΔV in Undet.	Error Norm. to V_{dd} Before Refined	Error Norm. to V_{dd} After Refined	Norm. Undet. Nodes	Norm. ΔV in Undet.	Error Norm. to V_{dd} Before Refined [×Less]	Error Norm. to V_{dd} After Refined [×Less]
ibmpg1/3	0.04	5.4E-03	1.1E-03	2.8E-04	0	0	2.1E-04[5.03×]	1.5E-05[19.40×]
ibmpg2/3	0.02	2.4E-03	1.5E-03	5.6E-04	0	0	2.2E-04[6.53×]	1.4E-04[3.92×]
ibmpg1/5	0.02	2.9E-03	1.3E-03	7.7E-05	0	0	1.0E-04[12.76×]	9.1E-06[8.49×]
ibmpg2/5	0.04	6.0E-03	5.4E-03	6.08E-03	0.001	2.25E-04	5.3E-04[10.17×]	3.3E-04[18.33×]
ibmpg1/7	0.06	1.2E-02	7.2E-03	2.5E-03	0	0	9.6E-04[7.52×]	1.7E-04[14.19×]
ibmpg2/7	0.10	1.5E-02	1.3E-02	6.7E-03	0.004	3.78E-04	1.0E-03[12.42×]	3.1E-04[21.41×]
Average	**0.05**	**7.4E-03**	**4.9E-03**	**2.7E-03**	**0.001**	**1.01E-04**	**5.1E-04[9.07×]**	**1.6E-04[14.29×]**

for obtaining the RoI, the amount of effort of backward solver is proportional to the perturbation size, while this amount for forward solver is proportional to RoI size, which is significantly larger than perturbation size (see Equation (5.3)).

Table 5.4 shows the runtime breakdown for forward solver and the backward solver. The forward solver uses three steps in creating an incremental solution, *Bookkeeping*, *RoI computation*, and *Refinement*. The bookkeeping step is the initialization step and is performed only once for a series of consecutive perturbations to a circuit. Therefore, the overhead of this initialization step is amortized for many consecutive perturbations. Hence, in this table the runtime of the forward and backward solver is shown for different number of consecutive perturbations.

Table 5.4 shows that the backward solver is up to 24×, and on average, 16× faster for a single run. For 10 runs (i.e., 10 consecutive perturbations) the backward solver is up to 3×, and on average, 2× faster. For 30 runs the runtime of the backward solver and forward solver are about the same. In general as the number of runs increase, the runtime benefit of backward solver compared to [55] decreases. However, the solution from [55] solver loses its accuracy as errors from consecutive perturbations accumulate (the bookkeeping information collected becomes less accurate); on the other hand, the backward incremental solver retains its accuracy regardless of the number of perturbations. In addition, as pointed

Table 5.4: Comparison of the runtime of the Forward solver and Backward solver (Averaged over 10 random perturbations. perturbation amount = 5%, tolerance = 1% of V_{dd}).

Bench-mark/ Pert. size	Forward Incremental Solver Runtime (sec) Book-keeping	RoI	Refine	1 run	10 runs	30 runs	Backward Incremental Solver Runtime (sec) RoI	Refine	1 run [Speedup]	10 runs [Speedup]	30 runs [Speedup]
ibmpg1/3	0.39	0.01	1.7E-04	0.40	0.47	0.62	0.02	2.6E-04	0.02[18.7×]	0.21[2.2×]	0.64[1.0×]
ibmpg2/3	2.48	0.06	1.2E-03	2.54	3.09	4.31	0.10	1.3E-03	0.10[24.6×]	1.03[3.0×]	3.10[1.4×]
ibmpg1/5	0.39	0.01	1.6E-04	0.40	0.49	0.70	0.03	1.9E-04	0.03[12.1×]	0.33[1.5×]	1.00[0.7×]
ibmpg2/5	2.48	0.07	1.1E-03	2.55	3.14	4.46	0.15	3.6E-03	0.15[16.9×]	1.51[2.1×]	4.52[1.0×]
ibmpg1/7	0.39	0.01	1.5E-04	0.40	0.49	0.70	0.03	3.4E-04	0.03[13.2×]	0.30[1.6×]	0.91[0.8×]
ibmpg2/7	2.48	0.07	1.1E-03	2.55	3.14	4.46	0.21	4.7E-03	0.21[12.1×]	2.11[1.5×]	6.32[0.7×]
								Average Speedup	**16.28**×	**1.98**×	**0.92**×

out earlier, the backward solver is scalable to larger problems.

Chapter 6

Scaled Solver

6.1 Background

6.1.1 The Notion of Importance Sampling

The idea behind the IS method, as discussed in more detail in Section 6.1.2, is to modify the distribution of the *naïve* random walks so that the gain of each walk is approximately the same. The change in the distribution causes a bias in the solution which is compensated for, using a weighted average scheme, and is referred to as *scaling*.

Figure 6.1 shows a portion of a graph on which random walks are run. For both the naïve and the optimally scaled game, each vertex in the graph represents a node in the power network. The edges are annotated with a set of transition probabilities (only two of which are actually shown in the figure), and the dashed edges represent connections to the rest of the network.

In the random walk game, the walker must to thoroughly inspect vertices on either side of v, i.e., the walks that start with transitions to v_1 and those that first go to v_2. For the naïve game, the walks through v_1 will have an average gain of $v_1 = 2$ (equal to its solution), while those through v_2 will have an average gain of $v_2 = 10$. This indicates that the variance of the data samples that are averaged

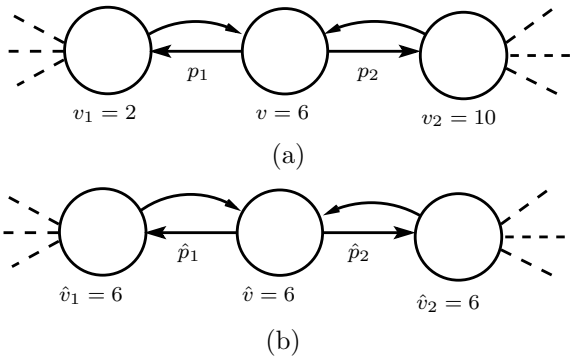


Figure 6.1: Example of the (a) naïve and (b) optimally scaled random walk games.

to compute v may be large.

In the ideal optimally-scaled game shown in Figure 6.1(b), the left hand side matrix is modified such that the solution of all the nodes are equal to (or practically, close to) 6. Regardless of the direction chosen, the average walk gain will be the same and equal to 6. If this is true at each node, the walk gain will have zero variance. Hence, the stopping criterion will kick in much more quickly since the samples have lower variance. In order to permit this change, as we will soon see, the probabilities of the corresponding edges must be changed from the naïve game and scaling factors must be employed.

6.1.2 The Theory of Importance Sampling

Importance sampling is based on the observation that:

$$\begin{aligned} v_i &= E_{Q_i}[V_i] = \sum_k V_i^k Q_i^k \\ &= \sum_k \left(V_i^k \frac{Q_i^k}{\hat{Q}_i{}^k} \right) \hat{Q}_i{}^k = E_{\hat{Q}_i}[V_i L_i] \end{aligned} \tag{6.1}$$

where $\hat{Q}_i{}^k Q_i^k > 0, \forall k$. The terms Q_i and $\hat{Q}_i$ are PMFs of the random variable V_i in the naïve and scaled problem, respectively, and V_i^k denotes the k_{th} sample of

V_i with probability Q_i^k and $\hat{Q}_i^k$, in the naïve and scaled problem, respectively.

The term $L_i^k = Q_i^k/\hat{Q}_i^k$ is called the likelihood ratio [51]. For the random walk solver for linear equations, the condition $\hat{Q}_i{}^k Q_i^k > 0, \forall k$ ensures that the structure (roads connecting the motels) of the random walk game is preserved.

For a suitable $\hat{Q}_i{}^k$, we can modify Equation (3.6) using Equation (6.1) to form a new unbiased estimator for v_i as [51]:

$$\hat{v}_i = \frac{1}{M_i} \sum_{k=1}^{M_i} V_i^k L_i^k \tag{6.2}$$

where the probability of the random walks is guided by $\hat{Q}_i^k$, and hence the notation $\hat{v}_i$. The variance of this estimator is [51]:

$$\sigma_{\hat{v}_i}^2 = \frac{E_{\hat{Q}}[(V_i L_i)^2] - v_i^2}{M_i} = \frac{\mu_{\hat{Q}}^2 - v_i^2}{M_i} \tag{6.3}$$

where $\mu_{\hat{Q}}^2$ is the second moment of the estimator $V_i L_i$ and L_i is a function of k as defined above. If $\hat{Q}_i$ is chosen appropriately, $\hat{v}_i$ will have significantly lower variance than v_i. In the optimal case, IS can achieve exactly zero variance, where the optimal choice of the PMF $\hat{Q}_i$ is given by:

$$\hat{Q}_i^k = V_i^k Q_i^k / v_i \tag{6.4}$$

If $V_i^k L_i^k = v_i, \forall k$, i.e., all walk gain samples are the same, from Equation (6.3), we have zero variance, which leads to Equation (6.4). The name "importance sampling" refers to the notion that $\hat{Q}_i^k$ is proportional to the amount of contribution of the k^{th} walk on the average walk gain, i.e., its importance.

It is clear that since this optimal choice requires some knowledge of v_i, and since v_i is the unknown that we are trying to compute, zero variance is impractical. However, this intuition is the founding idea of our proposed heuristic.

6.2 Fast Random Walk Solver

6.2.1 Scaled Random Walks

Our fast random walk solver is a heuristic technique, inspired by IS, to speedup the random walk solver for power network analysis. The essential idea of the approach is to modify the transition probabilities, p_{ij}, in Equation (3.5) such that the solution of all the nodes in the network is equal to α and use *scaling factors*, s_{ij}, to find the solution, v_i, to node i. The scaled form of Equation (3.5) can be written as:

$$v_i = \sum_{j=1}^{degree(i)} \hat{p}_{ij} s_{ij} v_j + m_i \tag{6.5}$$

where $\hat{p}_{ij}$ (analogous to $\hat{Q}_i$ in Section 6.1.2) denotes the new modified transition probabilities, s_{ij} (analogous to L_i) denotes the scaling factors corresponding to roads from node i to its neighbor j, and m_i is the motel cost at node i as in Equation (3.5). In the language of IS, the random walks that are constructed using Equation (3.5) provide an estimate of v_i while those corresponding to Equation (6.5) (we will shortly define them) are denoted by $\hat{v}_i$; both represent the voltage of node i.

Note that that for Equation (6.5) to model the same equations as (3.4) and (3.5), we must have:

$$s_{ij} = \begin{cases} 1, & i = j \\ p_{ij}/\hat{p}_{ij}, & i \neq j \end{cases}$$

Clearly, $\hat{p}_{ij}$ and p_{ij} both must be valid probabilities that lie between 0 and 1, and therefore $s_{ij} \geq 0$. Moreover, for the structure of the random walks to be intact, required for the IS to be valid as discussed in Section 6.1.2, we must have $s_{ij} > 0$ and $p_{ij} \times \hat{p}_{ij} > 0$.

Figure 6.2 shows an example of a graph on which the random walk game may be executed. The home nodes are indicated by the shaded circles and the rest of

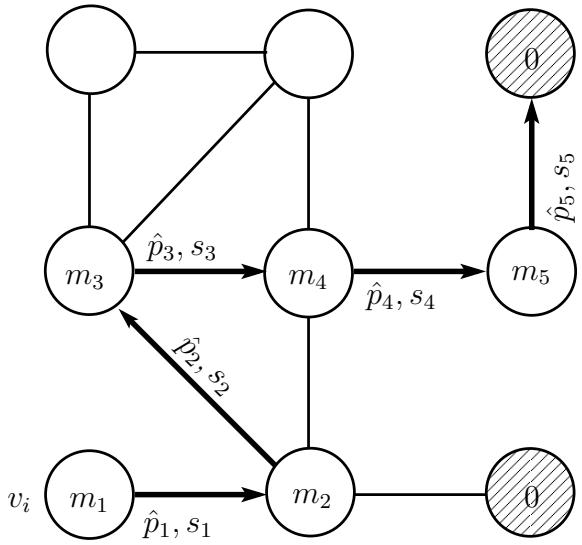


Figure 6.2: An example of a scaled random walk game, with a highlighted path showing a walk of length $N = 6$.

the vertices are the motel nodes. The highlighted path shows a walk of length $N = 6$ starting from vertex v_i, as the walker takes to reach a home node. The cost of the r^{th} motel node is reflected by m_r and the home rewards are all zero. The modified probabilities and the scales in the graph are shown by $\hat{p}_r$ and s_r, respectively.

The scaled random walk from node v_i, similar to the naïve game, starts with zero money and a credit card on which debt may be racked up. In the scaled walk, the walker accumulates a *multiplier* on the way, which models the scaling scheme. On the first transition the walker pays the price of the motel, m_1, and a road is picked with probability $\hat{p}_1$; the value of the multiplier is set to s_1. Next, the second transition is chosen and the motel costs are paid – but due to the accumulated multiplier, the amount to be paid is not m_2 but $s_1 \times m_2$. The multiplier changes to $s_1 \times s_2$, so that the motel cost at the third transition is $s_1 \times s_2 \times m_3$, the multiplier is $s_1 \times s_2 \times s_3$, and so on. The walker keeps paying and accumulating

new terms in the multiplier until a home node is reached, where a reward (also scaled by the multiplier) is received. Therefore, the total gain of a walk of length N can be written as:

$$\hat{V}_i^k = m_1 + s_1 m_2 + s_1 s_2 m_3 + \cdots + \prod_{r=1}^{N-1} s_r m_N \tag{6.6}$$

where r denotes the motels visited along the way from node i, and k denotes that this is the k^{th} walk among the M_i walks performed to estimate v_i. The result of this walk is included in an average that is used to estimate v_i:

$$\hat{v}_i = \frac{1}{M_i} \sum_{k=1}^{M_i} \hat{V}_i^k \tag{6.7}$$

As in [32], the number of walks, M_i, required for getting the solution with the desired tolerance is determined dynamically based on the estimated solution and its variance at each point in the random walk game such that the error is smaller than a user defined threshold with a confidence of 99%.

6.2.2 Computing the Scaling Factors

The scaling factors in Section 6.2.1 give us the freedom to modify the road probabilities, $\hat{p}_{ij}$, and yet find the solution to the original equation using Equation (6.6). A key unanswered issue is the choice of scaling factors. The IS approach suggests that we choose the $\hat{p}_{ij}$ such that every walk has a gain of α, where α is to be selected to get a feasible random walk and the best possible speedup. Based on this, we will now present a set of results that show how these factors may be computed.

Theorem 6 *For a system of linear equations defined by Equation* (6.5)*, if $s_{ij} v_j$ is at a constant value of α for all neighbors j of node i, and the value of v_i is also α then:*

$$1 - \sum_{j=1}^{degree(i)} p_{ij}/s_{ij} = m_i/\alpha \tag{6.8}$$

Proof: Based on the conditions shown above, we have:

$$\alpha = \sum_{j=1}^{degree(i)} \hat{p}_{ij}\alpha + m_i \tag{6.9}$$

Substituting p_{ij}/s_{ij} for $\hat{p}_{ij}$, we have the result. □

The conditions of Theorem 6 state that:

$$s_{ij} = \frac{\alpha}{v_j} \tag{6.10}$$

For a V_{dd} grid, the values of all node voltages are roughly similar in magnitude for a reasonable candidate power grid: the variations of a well-designed grid are within 10% of V_{dd}. In a use case where such an analyzer is being applied to the inner loop of an optimizer (e.g., one that sets the optimal wire widths in the grid), it is likely that for any candidate configuration, the voltage values may differ by more than 10%, but not by orders of magnitude. Therefore, since the v_j values are similar in magnitude, it is a reasonable approximation to assume that $s_{ij} = s_i, \forall j$. Note that we will use this approximation only to reorient the probabilities in the original random walk game for importance sampling. Therefore, by the definition of importance sampling, *this approximation does not affect the correctness of the random walks*, but only the variance of the samples, and hence the convergence speed.

The quality of this approximation is directly related to the amount by which the variance may be reduced. If a power grid has a catastrophic fault that causes its voltage to be very far from V_{dd}, this may not be a good solution, but it is an excellent choice within an optimizer where realistically, most of the candidate solutions have voltages close to V_{dd}.

Corollary 1 Under the assumptions that $s_{ij} = s_i \ \forall j$, the scaling factor s_i can be computed as:

$$s_i = \frac{1}{1 - m_i/\alpha} \sum_{j=1}^{degree(i)} p_{ij} \tag{6.11}$$

The proof is trivial and follows directly from Theorem 6.

Corollary 1 provides a recipe for determining the scaling factors in Equation (6.5). However, the parameter α has not been precisely defined. We will now consider constraints on the feasible values of α that are required to maintain the physical constraints associated with a random walk game.

Theorem 7 *For the case where $m_i \geq 0 \ \forall i$, α satisfies the feasibility and physicality of the random walk game if:*

$$\alpha > \max_i(m_i) \tag{6.12}$$

Proof: For all i and j, we must have:

$$0 < s_{ij} \tag{6.13}$$

$$0 < \hat{p}_{ij} \leq 1 \tag{6.14}$$

$$\sum_{j=1}^{degree(i)} \hat{p}_{ij} \leq 1 \tag{6.15}$$

The first condition is required to keep the structure of the random walk game intact, as required by the IS method in Section 6.1.2: in other words, the graph for the original game is isomorphic to that of the modified game. In the next two constraints, $\hat{p}_{ij} > 0$ implies that s_{ij} is finite (i.e., it avoids a divide-by-zero operation in Equation (6.6)), and the remaining constraints are basic requirements on a PMF.

Equations (6.11) and (6.13) together imply that $\alpha > m_i \ \forall i$, which immediately leads to Equation (6.12).

Equations (6.11) and (6.15) together imply that $m_i/\alpha > 0$. This is self-consistent with the result of this theorem. □

Note that for the case where $m_i \leq 0 \ \forall i$ (for the ground net), a result similar to Theorem 7 may be derived: that $\alpha < \min_i(m_i)$.

6.2.3 Choosing the Value of α

Based on Theorem 7, the value of α may be chosen as

$$\alpha = \max_i(m_i)\beta,\ \beta > 1 \tag{6.16}$$

where β is chosen empirically for the best speedup.

We now present an intuitive feel for the considerations for choosing β. Qualitatively, β determines the probability of transition to a home node at each node of the network. The choice of α alters probabilities p_{ij} to $\hat{p}_{ij}$, but it can be verified from Equation (6.8) that in general, at any node i, $\sum_i p_{ij} \neq \sum_i \hat{p}_{ij}$. In the original circuit, typically $\sum_i p_{ij} = 1$ at many nodes, but this is not the case in the modified circuit. This introduces a new home transition probability at node i for the fast solver:

$$\hat{h}_i = 1 - \sum_{j=1}^{degree(i)} \hat{p}_{ij} = \frac{m_i}{\max_i(m_i)\beta} \tag{6.17}$$

As β increases, the home probabilities decrease and therefore it will be less likely for the random walker to reach a home node. As a result, individual walk lengths increase as β increases, and the walker spends more time exploring "far away" parts of the network which have small contributions to the solution due to the locality property of power grids.

On the other hand as β decreases, the home probabilities become larger, making it more likely for random walks to often terminate at nearby home nodes, even within the radius of locality. To achieve accuracy, this implies the need for a larger number of walks, implying a larger total number of steps in the random walk, and hence, larger runtime for the solver. Another factor is that in order to keep m_i unchanged, required by Equation (6.5), the edges connected to these new home nodes must have a reward of zero, rather than V_{dd}. Therefore, the conditions that led to $s_{ij} = s_i \forall i$ are somewhat violated since one neighbor contributes a value of zero.

Experimental results empirically indicate that values of $5 \leq \beta \leq 50$ give the best results. This value could change if the topology of the benchmarks (e.g.,

average degree of each node) changes, but companies tend to use a similar style from design to design, and it is likely that this value will not change remarkably, once calibrated.

6.2.4 Fast Random Walks Example

In this section, we present an example to provide some intuition as to how the solver works and why it reduces the variance. Figure 6.3 shows a simple random walk game, in both the naïve and the scaled form. In this figure, each node is represented by a vertex where the shaded vertices are the home nodes with known values of zero. The numbers within each vertex correspond to the motel cost at the node, and the transition probabilities are shown on the edges. The scaled game has scale labels on the edges as well.

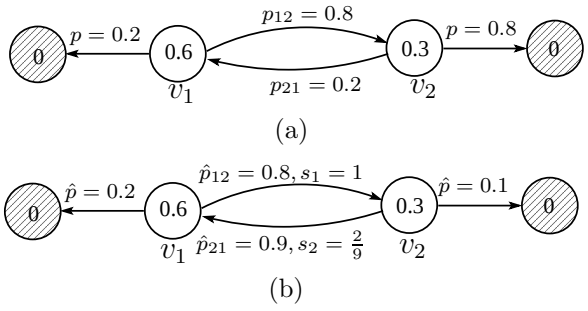


Figure 6.3: Simple example of the naïve 6.3(a) and scaled 6.3(b) games to demonstrate how the fast random walk solver works.

These games model the following set of equations:

$$\begin{bmatrix} 1 & -0.8 \\ -0.8 & 4 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.6 \\ 1.2 \end{bmatrix} \tag{6.18}$$

where the solution is $v_1 = 1$ and $v_2 = 0.5$. We use $\beta = 5$ and seek the the solution of node v_1.

In the naïve random walk game in Figure 6.3(a) the gain of the walks started from v_1 takes one of the forms:

$$V_i^k = \begin{cases} 0.6 + 0.9t \\ 0.9(1+t) \end{cases}, t = 0, 1, 2, \ldots \tag{6.19}$$

The first equation corresponds to a termination at the left home node, and the second to a termination at the right home node. Here, t is the number of times that the walker traverses the cycle v_1–v_2–v_1, and the walk length is $2t+1$ [$2t$] for a termination on the left [right] node. It is easy to see from this equation that the walk gain increases linearly with its length, and depending on the walk length, it falls in the interval.

For the scaled random walk in Figure 6.3(b), some vigorous algebraic manipulations show that the gain of the walks started from v_1 takes one of the forms:

$$V_i^k = \begin{cases} 0.9 \sum_{l=0}^{t} (2/9)^l \\ 0.9 \sum_{l=0}^{t} (2/9)^l - 0.3(2/9)^t \end{cases}, t = 0, 1, 2, \ldots \tag{6.20}$$

In this game, as t increases (i.e., the walk length increases), the walk gain increases at most to 1.16. Therefore, for this scaled game, the walk gains fall into the interval $[0.6, 1.16)$ which has a much smaller variance compared to the naïve game.

In practice for a properly chosen β, the loop gain in this example and total accumulated walk scale in general case will be less than or equal to one. To see this, consider an ideal power network with the solution of all one and the scaled game for solving it. Comparing Equations (3.5) (with $v_i = 1$) and Equation (6.8) we can see that in the scaled game, the RHS $m_i/\alpha = m_i/\left(\max_i (m_i)\beta\right) \leq m_i$, the RHS of the naïve game. Therefore, we must have $s_i \leq 1$. For the general case, the solution of the networks of interest for this work deviates no more than, say, 20% and setting β large enough ensures the scaling factors are less than or equal to one.

For efficiency purposes, we stop the walk as soon as the accumulated scale falls below a threshold close to machine precision since the motel costs are bounded and continuing the walk will have no effect on the solution due to round off error.

6.3 Experimental Results

The proposed fast random walk solver and the naïve random walk solver are implemented in C++ and compared on a UNIX machine with a 2GHz CPU and 8GB of memory. To ensure a fair comparison, the fast random walk solver is implemented by adding the scaling scheme into the naïve solver so that the core random walk engine is the same for both. These solvers are applied to three benchmarks summarized in Table 6.1. This table represents the statistics of the LHS matrix for a DC analysis. For each matrix, we list the size, i.e., the number of unknowns, the number of non-zeros in G, and the average number of non-zeros per row, as a sparsity metric.

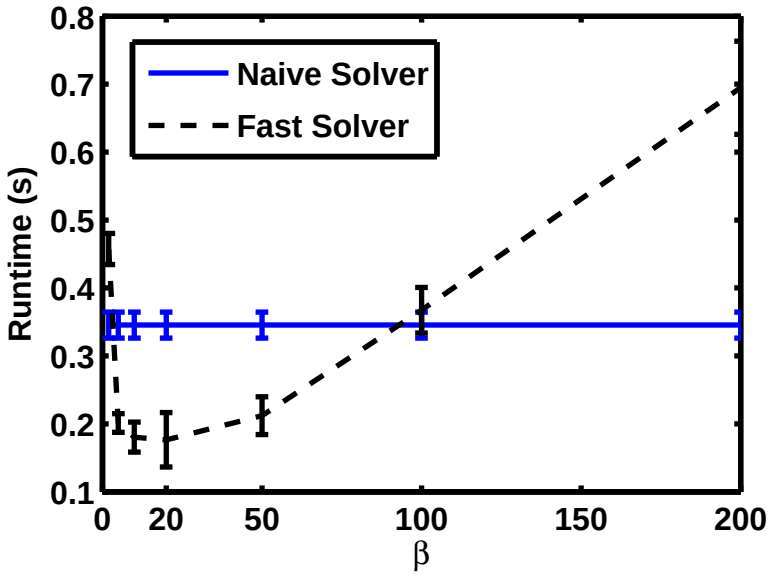
Table 6.1: Benchmark Details: Statistics of the LHS Matrix

Name	**Size**	**Number of Nonzeros**	**Average Nonzeros/Row**
c1	16194	98030	6.1
c2	26300	165810	6.3
c3	29551	178345	6.0

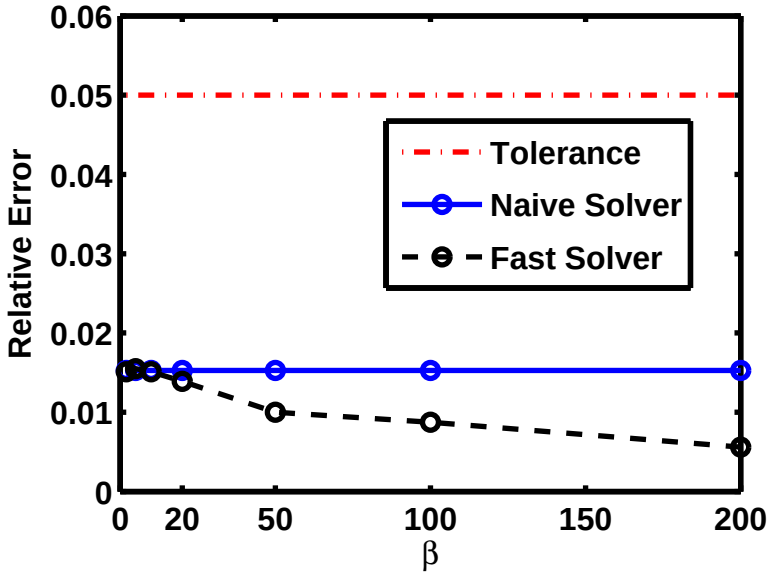
We first examine the efficiency and accuracy of the fast solver by finding the speedup of the fast solver over the naïve solver, for the same relative error. Next we compare the statistics of the walk gains for the naïve and fast solver, as defined by Equations (3.6) and (6.6), respectively, to study the variance reduction in the fast solver. In our experiments, we focus on the case that the solution of a single node in the network is of interest. We pick ten nodes randomly and apply the solvers for each of these nodes 1000 times and show that the average results are consistent for all the nodes in all the benchmarks. The nominal voltage of the network, V_{DD}, is $1.2V$.

Figure 6.4 shows the runtime and relative error of the naïve and fast solver versus β, for the given tolerance of 5%, for a single randomly selected node from benchmark $c2$.

Figure 6.4(a) suggests that for $5 \leq \beta \leq 50$, the fast solver achieves a significant

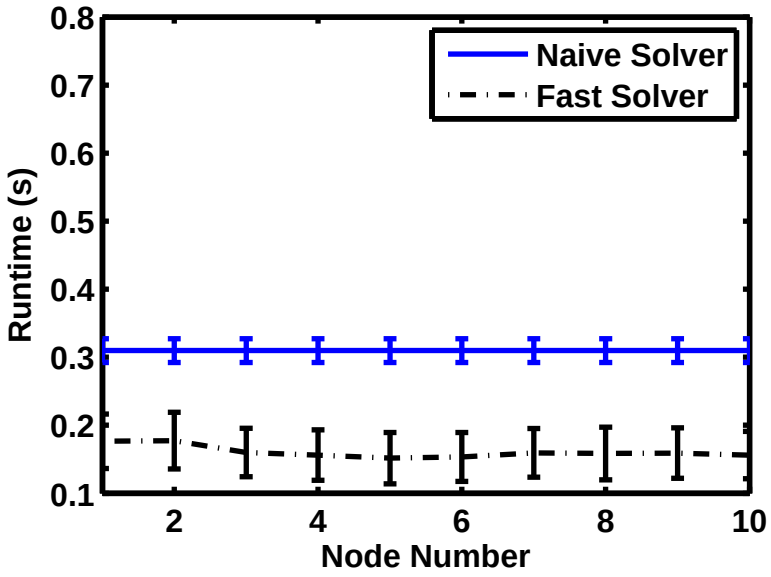


(a) Runtime

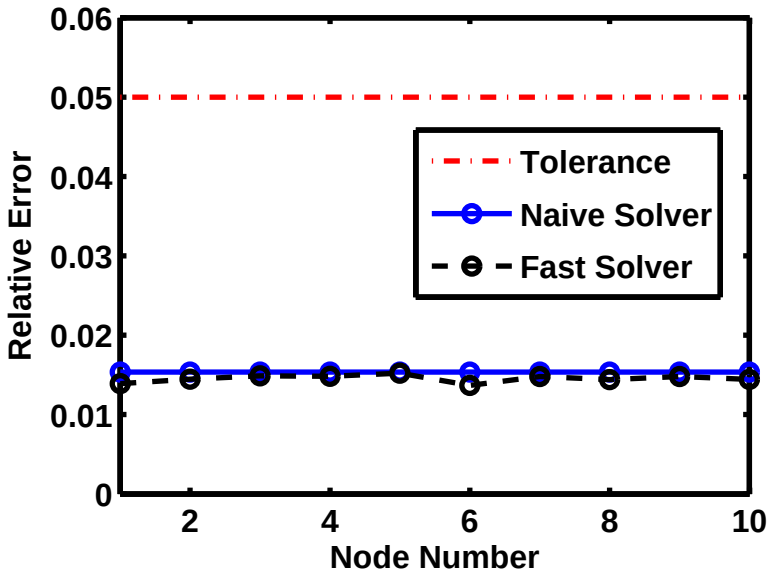


(b) Relative Error

Figure 6.4: Runtime and relative error vs. β for a random node from circuit c2.



(a) Runtime



(b) Relative Error

Figure 6.5: Runtime and relative error for 10 randomly chosen nodes (from c2) with $\beta = 20$.

speedup while the relative error, shown in Figure 6.4(b), remains less than or equal to the naïve solver. In Figure 6.4(a), the bars on the figure denote the standard deviation of the runtime over the 1000 runs.

Table 6.2: Runtime (sec) and speedup of the fast random walk solver for ten randomly chosen nodes from each benchmark for $\beta = 20$

Node	**Runtime (s) [Speedup]**		
	c1	**c2**	**c3**
1	0.067 [1.6×]	0.176 [1.8×]	0.169 [1.3×]
2	0.070 [1.5×]	0.177 [1.7×]	0.170 [1.3×]
3	0.070 [1.5×]	0.160 [1.9×]	0.152 [1.5×]
4	0.068 [1.5×]	0.156 [2.0×]	0.152 [1.5×]
5	0.065 [1.6×]	0.152 [2.0×]	0.153 [1.5×]
6	0.064 [1.6×]	0.153 [2.0×]	0.137 [1.6×]
7	0.054 [1.9×]	0.159 [1.9×]	0.154 [1.5×]
8	0.068 [1.5×]	0.159 [2.0×]	0.153 [1.5×]
9	0.064 [1.6×]	0.159 [1.9×]	0.155 [1.4×]
10	0.055 [1.9×]	0.156 [2.0×]	0.151 [1.5×]
Average	**0.065 [1.6×]**	**0.161 [1.9×]**	**0.155 [1.5×]**

As discussed in Section 6.2.3, the value of the parameter β is empirically chosen. Here, we set $\beta = 20$ for our experiments. Figure 6.5 shows the runtime and relative error of the solvers for ten randomly chosen nodes from benchmark $c2$ with tolerance of 5% and $\beta = 20$. In Figure 6.5(a) the bars on the figure show the standard deviation of the runtime over the 1000 different runs. It can be seen that the fast solver consistently runs faster than the naïve solver for all the nodes while the relative error is the same. Note that it is well-known that the actual error can be significantly below the tolerance, as seen in the figure: this is an artifact of the choice of stopping criterion in [32].

Details of the runtime and speedup of the fast random walk solver are listed in Table 6.2 for all the benchmarks where ten nodes are selected randomly from each benchmark and $\beta = 20$. Each row represents a node number and each column

represents the circuit that it comes from. This table indicates consistent runtime reduction for all the benchmarks and for each of the nodes of each benchmark.

Finally the scatter plot of Figure 6.6 shows the distribution of walk gain versus walk length of the naïve solver and the fast solver, as defined by Equations (3.6) and (6.6), for a randomly chosen node from benchmark $c2$ with solution of $1.06V$. As this figure indicates, the walk gain of the fast solver is much denser around the solution than that of the naïve solver. This indicates the effectiveness of the variance reduction technique described in Section 6.2.4.

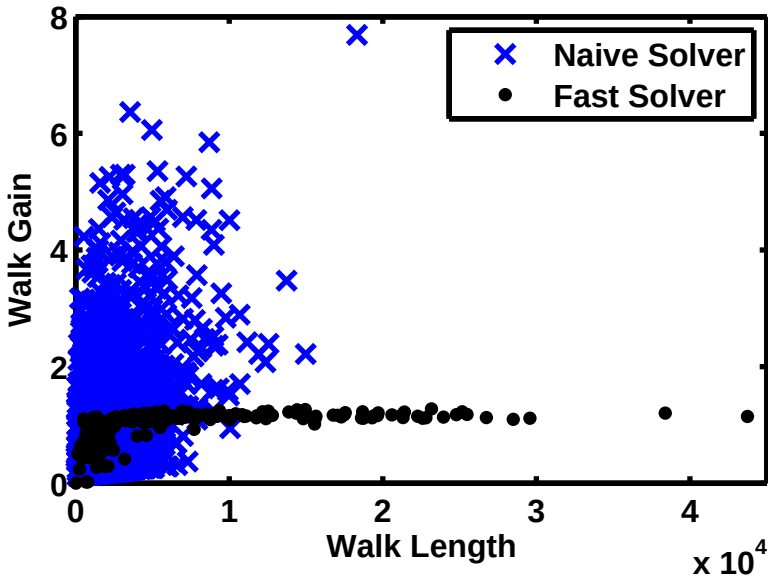


Figure 6.6: Distribution of the walk gain vs. walk length ($\beta = 20$, $v = 1.06$).

6.4 Extension

6.4.1 Extension of the Scaled Solver

In this work we focused on power networks of Equation (1.1) where the left hand side matrix is symmetric and diagonally dominant. Moreover we are assuming that the right hand side vector is all non-negative and the solution does not vary

drastically from node to node. This work can be extended to the cases that the solution can vary a lot for different nodes and the right hand side might have mixed signs.

This can be achieved by adding a preprocessing stage which is a coarse random walk solution of the entire system. Random walk solver provides a nice runtime-accuracy trade off and hence a solution of the network with moderate accuracy can be found efficiently. This approximate solution can be used to introduce a change of variable in Equation (1.1) (i.e. by dividing each unknown by its approximate solution) such that the new set of equations would have a small variation in their solution. Then the scaled solver of Section 6 can be used to find the solution of the new system efficiently.

6.4.2 Dynamic Calculation of Optimal β

As discussed in Section 6.2 the parameter β has a critical roll in effectiveness and stability of the scaled random walk solver. This parameter has been determined empirically which is shown to be working for all the benchmarks under test in this work. But the optimal value of β can be slightly different for different benchmarks. Specially in this work we focused on the power networks which normally have a fairly regular structure. In a more general case, we will be dealing with a variety of game structures which will demand for a dynamic procedure to obtain the optimal β.

The key to finding the optimal β dynamically is the variance of the walk gains in the scaled game of Section 6.2.1. This variance can be found using:

$$\sum_{k=1}^{n} \hat{a}_{ik} s_{ik}^2 \hat{\sigma}_k^2 \;=\; 2m_i v_i - m_i^2 - \sum_{k=1}^{n} \hat{a}_{ik} s_{ik}^2 v_k^2, \quad i = 1, 2, 3, \cdots, n \tag{6.21}$$

where in this equation n is the equation size, m_i is the motel cost of the i^{th} node as defined in Equation (3.5), and for a_{ik} we have:

$$\hat{a}_{ik} = \begin{cases} 1 & i = k \\ \hat{p}_{ik(j)} & k \neq i \end{cases} \tag{6.22}$$

where $k(j)$ represents the actual index of the j^{th} neighbor of the node i. In Equation (6.21) the scaling factors s_{ik}'s, and consequently, $\hat{p}_{ik}$'s are function of β and hence finding the optimal β will be reduced to minimize the variances in this equation.

Chapter 7

Conclusion

In this thesis, we have developed two methods for efficient incremental power grid analysis based on forward and backward random walk methods. The backward walk approach is particularly useful in computing the impact of incremental changes since it can efficiently compute the inverse of any individual column of the LHS matrix. The forward walk approach is useful for the case that the bookkeeping information is already known and few quick incremental updates to the solution is desired. These approaches are approximate and are used to determine a RoI around the area of the perturbation where the voltages are significantly impacted. Next, an exact direct solver may be employed to determine the precise solution by solving for voltages within the RoI while assuming that voltages outside the RoI are unchanged or at their approximate levels, as given by the random walk solver.

The theory of the backward random walks was developed in this work based on a basic concept introduced more than 50 years ago. We employed the backward random walk method to make the approach computationally practical, show a theoretical relation to LU factorization, and apply it to incremental analysis. Table 7 summarizes the similarities and differences of the forward and backward random walk game.

In addition, we introduced an accurate method for modeling the power grids

Table 7.1: Comparison of forward and backward random walks

	Forward	Backward
Construction based on	Rows of G	Columns of G
Computes	Rows of G^{-1} $v_i = (G^{-1})_{i,:}\mathbf{E}$ Effect of all RHS on solution of one node	Columns of G^{-1} $\mathbf{V}^j = (G^{-1})_{:,j}e_j$ Effect of one RHS on solution of all nodes
Game objective	Record the number of visits to motels	Same
Analogy	Sum of money paid during walks	Distribute a given sum of money among motels visited based on the visit frequency
Physical meaning of sum of money	Voltage of node i, v_i	Total voltage drop in the network due to one current load
$(G^{-1})_{i,j}$	Average number of visits to motel i over many walks started from j	Same

that contrary to the conventional power grid models could result in asymmetrical equations. We showed the effectiveness of the backward incremental random walk solver on the symmetrical as well as the more accurate asymmetrical power grid model.

Finally a fast scaled random walk solver has been developed for solving for the nodes in power grids one at a time. The scaled solver leverages the notion of importance sampling to reduce the variance in the walks to significantly improve the runtime of the random walk solver.

References

[1] S. Bodapati and F. N. Najm. High-level current macro-model for power grid analysis. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 385–390, 2002.

[2] H. H. Chen and D. D. Ling. Power supply noise analysis methodology for deep-submicron vlsi chip design. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 638–643, 1997.

[3] T. Chen and C. C. Chen. Efficient large-scale power grid analysis based on preconditioned krylov-subspace iterative methods. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 559–562, 2001.

[4] A. Dharchoudhury, R. Panda, D. Blaauw, R. Vaidyanathan, B. Tutuianu, and D. Bearden. Design and analysis of power distribution networks in power-pctm microprocessors. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 738–743, 1998.

[5] A. Dharchoudhury, R. Panda, D. Blaauw, R. Vaidyanathan, B. Tutuianu, and D. Bearden. Podea: power delivery efficient analysis with realizable model reduction. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 4, pages 608–611, 2003.

[6] J. Kozhaya, S. R. Nassif, and F. N. Najm. A multigrid-like technique for power grid analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(10):1148–1160, 2002.

[7] R. Panda, D. Blaauw, R. Chaudhury, V. Zolotovand B. Young, and R. Ramaraju. Model and analysis for combined package and on-chip power grid simulation. In *Proceedings of the ACM International Symposium on Low Power Electronics and Design*, pages 179–184, 2000.

[8] S. S. Sapatnekar and H. Su. Analysis and optimization of power grids. *IEEE Design & Test*, 20(3):7–15, May–June 2003.

[9] J. C. Shah, A. A. Younis, S. S. Sapatnekar, and M. M. Hassoun. An algorithm for simulating power/ground networks using pade approximations and its symbolic implementation. *IEEE Transaction on Circuits and Systems II: Analog and Digital Signal Processing*, 45:1372–1382, 1998.

[10] H. Su, K. H. Gala, and S. S. Sapatnekar. Fast analysis and optimization of power/ground networks. In *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pages 477–480, 2000.

[11] M. Zhao, R. V. Panda, S. S. Sapatnekar, and D. Blaauw. Hierarchical analysis of power distribution networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(2):159–168, 2002.

[12] M. N. Özişik. *Heat Transfer: A Basic Approach*. McGraw-Hill, New York, NY, 1985.

[13] H. Qian, S. S. Sapatnekar, and E. Kursun. Fast poisson solvers for thermal analysis. *ACM Transactionson Design Automation of Electronic Systems*, 17(23), 2012.

[14] Y. Zhan and S. S. Sapatnekar. A high efficiency full-chip thermal simulation algorithm. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 634–637, 2005.

[15] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: Principles and methods. *Proceedings of the IEEE*, 94(8):1487–1501, 2006.

[16] Wei Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M.R. Stan. Hotspot: a compact thermal modeling methodology for early-stage vlsi design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(5):501–513, 2006.

[17] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 269–274, 1998.

[18] A. P. Hurst, P. Chong, and A. Kuehlmann. Physical placement driven by sequential timing analysis. In *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pages 379–386, 2004.

[19] N. Viswanathan and C. C. Chu. Fastplace: efficient analytical placement using cell shifting, iterative local refinement and a hybrid net model. In *Proceedings of International Symposium on Physical Design*, pages 26–33, 2004.

[20] K. P. Vorwerk, A. Kennings, and A. Vannelli. Engineering details of a stable force-directed placer. In *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pages 573–580, 2004.

[21] K. P. Vorwerk and A. Kennings. An improved multi-level framework for forcedirected placement. In *Proceedings of ACM/IEEE Design Automation and Test in Europe*, pages 902–907, 2005.

[22] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, New York, NY, 1986.

[23] A. George and J. W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[24] R. Barrett, M. Berry, T. F. Chan, J. W. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. A. van der Vorst. *Templates for the*

Solution of Linear Systems: Building Blocks for Iterative Methods. SIAM, Philadelphia, PA, 1994.

[25] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA, 2003.

[26] R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, NJ, 1962.

[27] W. Wasow. A note on the inversion of matrices by random walks. *Mathematical Tables and Other Aids to Computation*, 6(38):78–81, 1952.

[28] G. E. Forsythe and R. A. Leibler. Matrix inversion by a monte carlo method. *Mathematical Tables and Other Aids to Computation*, 4(31):127–129, 1950.

[29] A. Srinivasan and V. Aggarwal. Stochastic linear solvers. In *Proceedings of the SIAM Conference on Applied Linear Algebra*, 2003.

[30] C. J. K. Tan and M. F. Dixon. Antithetic monte carlo linear solver. In *Proceedings of International Conference on Computational Science*, pages 386–392, 2002.

[31] J. M Hammersley and D. C Handscomb. *Monte Carlo Methods*. Methuen & Co., London, U.K., and John Wiley & Sons, New York, NY, 1964.

[32] H. Qian, S. R. Nassif, and S. S. Sapatnekar. Random walks in a supply network. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 93–98, 2003.

[33] H. Qian and S. S. Sapatnekar. A hybrid linear equation solver and its application in quadratic placement. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 905–909, 2005.

[34] W. Guo, S.X.D. Tan, Z. Luo, and X. Hong. Partial random walk for large linear network analysis. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 5, 2004.

[35] T. Miyakawa, K. Yamanaga, H. Tsutsui, H. Ochi, and T. Sato. Acceleration of random-walk-based linear circuit analysis using importance sampling. In *Proceedings of the ACM/IEEE Great Lakes Symposium on VLSI*, pages 211–216, 2011.

[36] J. H. Curtiss. Sampling methods applied to differential and difference equations. In *Proceedings of IBM Seminar on Scientific Computation*, pages 87–109, 1949.

[37] R. Hersh and R. J. Griego. Brownian motion and potential theory. *Scientific American*, 220:67–74, 1969.

[38] C. N. Klahr. *A Monte Carlo method for the solution of elliptic partial differential equations*. John Wiley and Sons, New York, NY, 1962.

[39] A. W. Knapp. Connection between brownian motion and potential theory. *Journal of Mathematical Analysis and Application*, 12:328–349, 1965.

[40] M. E. Muller. Some continuous monte carlo methods for the dirichlet problem. *Annals of Mathematical Statistics*, 27:569–589, 1956.

[41] J. Singh and S. S. Sapatnekar. Topology optimization of structured power/ground networks. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 116–123, 2004.

[42] E. Chiprout. Fast flip-chip power grid analysis via locality and grid shells. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 485–488, 2004.

[43] M. Zhao, R. V. Panda, S. S. Sapatnekar, T. Edwards, R. Chaudhry, and D. Blaauw. Hierarchical analysis of power distribution networks. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 150–155, 2000.

[44] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[45] L. T. Pillage, R. A. Rohrer, and C. Visweswariah. *Electronic Circuit and System Simulation Methods*. McGraw-Hill, New York, NY, 1994.

[46] Y. Fu, R. Panda, B. Reschke, S. Sundareswaran, and M. Zhao. A novel technique for incremental analysis of on-chip power distribution networks. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 817–823, 2007.

[47] Y. Ye, Z. Zhu, and J. R. Philips. Generalized Krylov recycling methods for solution of multiple related linear equation systems in electromagnetic analysis. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 682–687, 2008.

[48] F. Castro, M. Sbert, and J. H Halton. Efficient reuse of paths for random walk radiosity. *Computers & Graphics*, 32(1):65–81, 2008.

[49] H. Qian, S. R. Nassif, and S. S. Sapatnekar. Power grid analysis using random walks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(8):1204–1224, August 2005.

[50] S. R Nassif. Power grid analysis benchmarks. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 376–381, 2008.

[51] I. Kuruganti and S. G. Strickland. Importance sampling for markov chains: computing variance and determining optimal measures. In *Proc. Winter Simulation Conference*, pages 273–280, 1996.

[52] S. Andradottir, D. P. Heyman, and T. J. Ott. Potentially unlimited variance reduction in importance sampling of Markov chains. *Advances in Applied Probability*, 28(1):166–188, 1996.

[53] P. W. Glynn and D. L. Iglehart. Importance sampling for stochastic simulations. *Management Science*, 35(11):1367–1392, November 1989.

[54] H. Qian and S. S. Sapatnekar. Stochastic preconditioning for diagonally dominant matrices. *SIAM Journal on Scientific Computing*, 30(3):1178–1204, March 2008.

[55] B. Boghrati and S. S. Sapatnekar. Incremental solution of power grids using random walks. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 757–762, 2010.

[56] R. D. Yates and D. J. Goodman. *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*. John Wiley and Sons, New York, NY, 1999.

[57] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[58] B. Boghrati and S. S. Sapatnekar. Incremental power network analysis using backward random walks. In *Proceedings of the Asia-South Pacific Design Automation Conference*, pages 41–46, 2012.

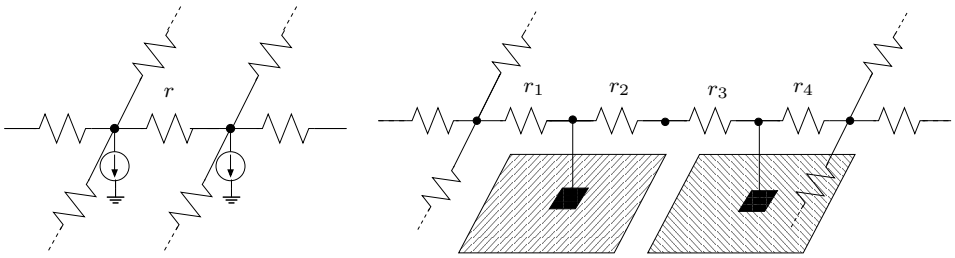
[59] MCNC Floorplan Benchmark Suite. Available at: `http://www.cse.ucsc.edu/research/surf/GSRC/MCNC`.

[60] NANGate. Available at: `http://www.si2.org/openeda.si2.org/projects/nangatelib`.

Appendix A

Modifying Existing Benchmarks to Add More Detail

In this section, we describe how our test circuits showing a detailed extracted power grid were generated from an existing standard benchmark suite that is based on a lumped current source approximation [50].



(a) Extracted power grid (b) Detail extracted power grid

Figure A.1: Splitting a wire piece of extracted circuit to model it as a detailed extracted circuit, $r = r_1 + r_2 + r_3 + r_4$.

The format of the original benchmark circuits is similar to Figure 2.3, with lumped current sources at the intersections of long wire segments. Our modification that generates the detailed extracted circuit distributes the current sources at the power line intersections to specific locations on the power lines. To this end, the power line between any two nodes is first split into multiple pieces. As

an example, Figure A.1 shows a wire being split as if there were two cells laid out between these power grid nodes.

While it is possible to perform this split at very fine levels of granularity, there will be diminishing returns in accuracy beyond a point. In this work, our goal is to motivate the notion that the lumped approximation induces errors and to generate realistic power grids with asymmetric LHS matrices that could be exercised with the backward random walk solver. Therefore, for simplicity, we have assumed that there are two equally spaced cells connected to the grid between two adjacent nodes. Then, a portion of the current source at the power grid intersection is associated with each of the cells adjacent to it such that the total current is kept the same. Here the current load of the intersection current sources is distributed uniformly among the adjacent cells, effectively attempting to reverse-engineer the original process in which the cells were lumped together.

Once the cell currents are determined, each cell is modeled as a simple VCCS controlled by the voltage of the node connecting that cell to the power network, i.e., an independent current source with a parallel resistor. The parameters of this VCCS are then found with the aid of a *unit cell* for which these parameters are calibrated. A unit cell, as explained shortly, represents an average cell used in VLSI circuits. This unit cell is scaled such that its current draw at V_{dd} (or zero for GND network) is equal to the current draw of the cell it is replacing. The circuit model obtained with this procedure is the detailed extracted circuit of the power grid.

Next we look at the process of characterizing the unit cell. To this end, we determine:

- the statistics of cell usage in a typical set of circuits (here, we use the MCNC benchmark suite [59] to obtain these statistics and a NANGate library [60]).

- the current load model of each cell (an independent current source in parallel with a resistor), computed for the cell by measuring the variation in the current draw when V_{dd} is swept from 90% to 100% using small-signal SPICE

simulations, and fitting a line on the I-V characteristic.

Given the cell usage statistics, the unit cell is obtained by computing the expected value of the current source and conductance of the load model of the cells, averaged over the distribution of cells.

Appendix B

Proof of Lemma 1

For the circuits in Figure 2.4(a) and 2.4(b) to be equivalent, as seen from nodes a and b, I_a and I_b in both circuits must be equal for any V_a and V_b.

$$\begin{aligned} I_x &= I_x^0 + g_{xa}v_a + g_{xb}v_b + g_x v_x \\ I_y &= I_y^0 + g_{ya}v_a + g_{yb}v_b \\ I_z &= I_z^0 + g_{za}v_a + g_{zb}v_b \\ g_{ab} &= \frac{g_a g_b}{g_a + g_b} \end{aligned}$$

For the T-model of Figure 2.4(a), by KCL, the sum of all currents entering nodes x, a, and b are zero according to KCL. Incorporating the device equations and KVL, this leads to the relations:

$$\begin{aligned} v_x &= \frac{v_a(g_a - g_{xa}) + v_b(g_b - g_{xb}) - I_x^0}{g_a + g_b + g_x} \\ I_a &= g_a(v_a - v_x) \\ I_b &= g_b(v_b - v_x) \end{aligned}$$

where v_a and v_b are the voltage of nodes a and b respectively.

Eliminating v_x and defining $\hat{g}_x = g_a + g_b + g_x$ we get :

$$\begin{aligned} I_a &= g_a \left[\left(1 - \frac{g_a - g_{xa}}{\hat{g}_x} \right) v_a - \left(\frac{g_b - g_{xb}}{\hat{g}_x} \right) v_b + \frac{I_x^0}{\hat{g}_x} \right] \\ I_b &= g_b \left[- \left(\frac{g_a - g_{xa}}{\hat{g}_x} \right) v_a + \left(1 - \frac{g_b - g_{xb}}{\hat{g}_x} \right) v_b + \frac{I_x^0}{\hat{g}_x} \right] \end{aligned} \tag{B.1}$$

For the Π-model in Figure 2.4(b), similarly applying KCL, KVL, and the device equations at nodes a and b, we have:

$$\begin{aligned} I_a &= (g_{ya} + g_{ab})v_a + (g_{yb} - g_{ab})v_b + I_y^0 \\ I_b &= (g_{za} - g_{ab})v_a + (g_{zb} + g_{ab})v_b + I_z^0 \end{aligned} \tag{B.2}$$

For the T-model and Π-model to be equivalent, Equation (B.1) and Equation (B.2) must be equivalent for any I_a, I_b, v_a, and v_b. Therefore, the corresponding coefficients of these equations must be equal. Equating these coefficients completes the proof.

Appendix C

Proof of Theorem 1

Figure C.1 shows two intersection nodes a and b of a detailed model of a power grid with K intermediate nodes (see nodes 1 and 2 of Figure 2.2 where $K = 5$). In this figure connected cells are modeled as an independent current source in parallel with a resistor. Note that the simplified extracted power grid model is a special case of this general model in which all the cell conductances are zero and the current sources are lumped together. The MNA stamp of the coarse asymmetric VCCS model of this circuit block is:

$$\begin{bmatrix} I_a \\ I_b \end{bmatrix} = \begin{bmatrix} g_{aa} & g_{ab} \\ g_{ba} & g_{bb} \end{bmatrix} \begin{bmatrix} v_a \\ v_b \end{bmatrix} + \begin{bmatrix} I_a^0 \\ I_b^0 \end{bmatrix} \tag{C.1}$$

Note that this stamp formulates the electrical characteristics of this circuit block as seen from nodes a and b. Lemma 1 utilized the same electrical characteristics to show two circuit blocks are equivalent.

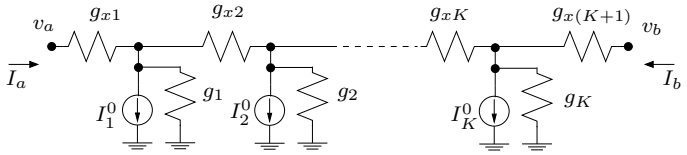


Figure C.1: Power grid model for two adjacent intersection nodes and the intermediate nodes in between

The LHS of the Equation (1.1) is made of the superposition of the conductance matrix of this stamp for all the intersection nodes. Therefore, to show the proposed properties of this theorem, it is enough to show that this stamp has these properties, mathematically:

$$\begin{aligned} g_{ba} &\leq 0 && \text{(C.2)} \\ g_{aa} &> 0 && \text{(C.3)} \\ |g_{aa}| &\geq |g_{ba}| && \text{(C.4)} \\ g_{ab} &\leq 0 && \text{(C.5)} \\ g_{bb} &> 0 && \text{(C.6)} \\ |g_{bb}| &\geq |g_{ab}| && \text{(C.7)} \end{aligned}$$

By linearity of this circuit, g_{aa} and g_{ba} can be written as:

$$\begin{aligned} g_{aa} &= I_a/v_a \\ g_{ba} &= I_b/v_b \\ I_k^0 &= 0,\ \forall k \in [a,b] \\ v_b &= 0 && \text{(C.8)} \end{aligned}$$

Figure C.2 shows the equivalent circuit of Figure C.1 in which $I_k^0 = 0,\ \forall k$ and $v_b = 0$. In this figure for $v_a > 0$, it is obvious that $I_a > 0$ and $I_b < 0$ therefore $g_{aa} > 0$ and $g_{ba} < 0$. Moreover, if there was no resistive path between a and b, then $I_b = 0$. Hence Equations (C.2) and (C.3) hold.

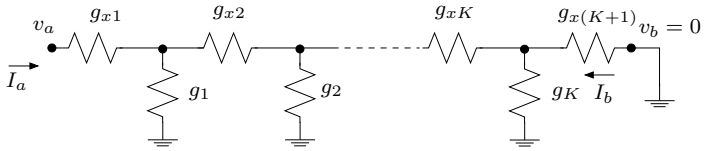


Figure C.2: Power grid model of C.1 with $I_k^0 = 0\ \forall k$ and $v_b = 0$

To show Equation (C.4), based on Equation (C.8), it is enough to show that in the circuit of Figure C.2, for a fixed v_a we have:

$$|I_a| \geq |I_b| \qquad \text{(C.9)}$$

Based on KCL, the equality holds if $g_k = 0,\ \forall k$ and otherwise we have $|I_a| > |I_b|$. Also if there was no resistive path between a and b, $I_a = I_b = 0$ and as a result $g_{aa} = g_{ba} = 0$. Therefore Equation (C.4) holds.

Te same argument shows that Equations (C.5), (C.6), and (C.7) hold if we set $I_k^0 = 0,\ \forall k$ and $v_a = 0$.

Finally, since every node in the power grid is connected to at least one other node in the grid, there is one non-zero stamp associated with every node and therefore, all diagonal entries of the LHS are greater than zero. Hence, Equations (C.3) and (C.6) hold for every node in the grid. □