Incremental Solution of Power Grids using Random Walks

Baktash Boghrati
University of Minnesota
e-mail: baktash@umn.edu

Sachin Sapatnekar
University of Minnesota
e-mail: sachin@umn.edu

***Abstract*—It is common for a designer to consider making several small changes to a power grid, corresponding to "what if" scenarios, in an attempt to improve its performance. To evaluate the effects of each incremental change, the circuit must go through incremental analysis. This paper presents a computationally efficient and accurate method for fast and accurate incremental analysis using random walks to identify a region of influence (RoI) of a change, so that this RoI can then be analyzed by any other solver. Our experimental results demonstrate the accuracy and computational efficiency of this method.**

I. Introduction

The solution of systems of linear algebraic equations is an important subproblem in a number of areas of VLSI design, e.g., power grid analysis, thermal analysis under a finite difference discretization, and quadratic placement [1]. The linear equations in three problems, in fact, have very similar characteristics. In this paper, we will focus on power grid analysis, although our results are applicable to the other problems.

We represent the equations for the power grid as:

$$G\mathbf{V} = \mathbf{E}, \tag{1}$$

where $G \in \Re^{n\times n}$ is the left hand side matrix, $\mathbf{V} \in \Re^n$ is the vector of unknowns, and $\mathbf{E} \in \Re^n$ is the right hand side vector. Then G is symmetric (i.e., the $(i,j)^{\text{th}}$ element of G, $g_{ij} = g_{ji}\ \forall\ 1 \le i,j \le n$), and diagonally dominant, i.e., $g_{ii} \ge \sum_{j,j\neq i} \mid g_{ij} \mid$. Mainstream methods for solving such systems include direct methods such as LU/Cholesky factorization and iterative methods. Lately, there has been an upsurge of interest in the use of random walk-based solvers for solving systems with diagonally dominant left hand sides. Recent work in [2] has developed methods that make these solvers competitive with direct and iterative solvers, through the reuse of random walk information. The work in [3] showed a relation between these solvers and the LU factors of G.

We address the problem of performing incremental analysis using random walks for the steady state case, and point out that the idea can be extended to transient analysis easily. Since the design process is inherently highly iterative, a designer may change a part of an otherwise complete design and wish to determine its effects on the system through incremental reanalysis. A large number of such changes may be tried out on the same topology to evaluate "what if" scenarios before one is accepted. Here, the original unperturbed system has already been solved, and the perturbed system satisfies the following properties:

- If the perturbation is small, the perturbed system has a solution that is *close* in value to the original solution; most variables remain unchanged, and many of those that change do so by small amounts.
- A small perturbation has the property of *locality*, i.e., the nodes corresponding to the altered variables are in a region of the power grid that is close (in some physical or topological sense) to the perturbation [4].

This work was supported in part by the NSF under award CCR-0634802.

Both ideas lead to the notion that solving the entire system from scratch is wasteful, and efficient solutions, leveraging the solution of the unperturbed system, may be obtainable.

Moving from intuition to a concrete solution strategy is a nontrivial task, and this is the subject of this work. One possible method would be to partition the network, create macromodels for each partition, and solve the problem hierarchically, as in [5]. However, this may require a large number of partitions, involving large amounts of computation. Other approaches may employ iterative solvers [6] using the unperturbed solution as an initial guess, sensitivity methods [7], and the fictitious domain method [8]. These methods suffer from the fact that they are required to solve the full system again, for the entire chip, and do not fully utilize the two properties above. An iterative solver is also described in [9], but operates on smaller, denser systems.

The rest of the paper is organized as follows. We first define the concept of a region of influence in Section II. Section III then presents a review of the basic random walk method, after which Section IV describes our novel incremental solver. Finally, experimental results are presented in section V.

II. The Region of Influence

Figure 1 pictorially illustrates the intuitive idea of locality described in Section I. If the rectangle represents the entire chip, and the dark perturbed region shows the area where the circuit has been altered, there is a volume around the perturbed region where the original solution is significantly affected; beyond this lies the unaffected region where the change in the solution is sufficiently small. We refer to the altered region as the *region of influence* (RoI) of the perturbation.

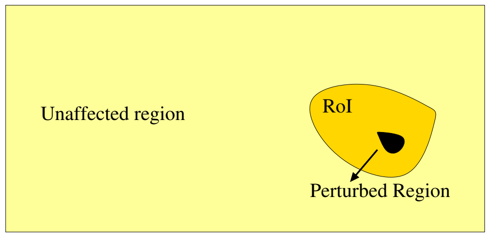


Fig. 1. A schematic of a design perturbation and its RoI.

The idea of using random walks to solve this problem has great intuitive appeal. Random walk techniques are inherently good at performing local analyses; unlike direct or iterative methods, they can find the solution to one variable without solving the entire system. The on-demand random walk method [10] reuses the original set of random walks and running random walks from selected nodes. However, it does not identify an RoI, does not utilize bookkeeping information, and its accuracy is constrained by the limited accuracy of random walks. We present an approach that leverages the bookkeeping information [2] collected during the initial system solution.

We introduce a novel incremental analysis method, using bookkeeping information. The method has two steps:

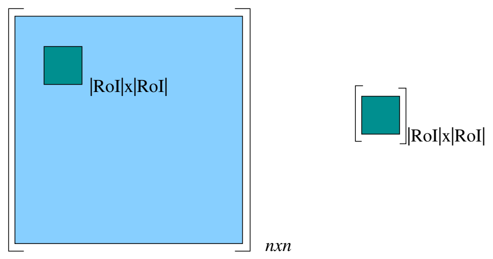


Fig. 2. Reduction in system size using RoIs.

- First, the RoI is estimated efficiently using random walks.
- Next, all variables outside the RoI are assumed unchanged, greatly reducing the large system of linear equations. The left portion of Figure 2 shows a schematic of the larger G matrix, containing a small submatrix corresponding to the RoI variables and equations. By setting all non-RoI variables to constants, we are left with a smaller linear system, shown at right in the figure; clearly, solving this is cheaper.

The second step can use *any* class of solver, including direct solvers, and can achieve arbitrarily good accuracy.

The philosophy of this approach is that the random walk method has sufficient fidelity to encapsulate the nature of variations, but is perhaps not accurate enough to capture small variations. Therefore, it is an excellent choice for identifying the RoI, but a more accurate solver should be used to detect the changes that may be seen in incremental analysis.

III. The Random Walk Method

We briefly review the random walk method in the context of power network analysis. Consider a node i in a power network, connected by resistors to each neighboring node $j = 1, \cdots, degree(i)$ by a conductance γ_{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:

$$\sum_{j=1}^{degree(i)} \gamma_{ij}(V_j - V_i) = I_i \qquad (2)$$

The terms in Equation (2) can be rearranged as:

$$V_i = \sum_{j=1}^{degree(i)} \frac{\gamma_{ij}}{\Gamma_i} V_j - \frac{I_i}{\Gamma_i} \qquad (3)$$

where $\Gamma_i = \sum_{j=1}^{degree(i)} \gamma_{ij}$. Equation (3) 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 (and equal to 1 if add in conductances to ground at i). For a power grid that has N non-VDD 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).

The random walk framework is based on an analogy between this system of equations and a random walk game on a network of roads. The nodes in this network 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, p_{ij}, of taking a specific road maps on to the corresponding weight, γ_{ij}/Γ_i, for the edge (Equation (3)). Each node has a motel that charges a cost of I_i/Γ_i 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), 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. Then the expected value of the amount of money, V_i, that the walker possesses at the end of the walks is given by

$$V_i = \sum_{j=1}^{degree(i)} p_{ij} V_j - m_i \qquad (4)$$

It is easy to see that Equations (3) and (4) correspond exactly. The notation, V_i, is overloaded to denote both the result of the random walk and the voltage of the node precisely because the two are identical, i.e., the expected value of the money in the walker's pocket, 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. 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:

$$V_i = \frac{1}{M_i}\left(\sum_{k \in \text{homes}} H_{ik} V_k - \sum_{j \in \text{motels}} J_{ij} \frac{I_j}{\Gamma_j}\right) \qquad (5)$$

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 (5), we have:

$$V_i = \sum_{j=1}^{i-1} \frac{H_{ij}}{M_i} V_j - \sum_{j=i}^{N} \frac{J_{ij}}{M_i} \frac{I_j}{\Gamma_j} \qquad (6)$$

Rewriting Equation (6) in matrix form, we have:

$$\begin{aligned} \mathbf{V} &= Y\mathbf{V} + Z\mathbf{b} & (7)\\ \text{i.e., } \mathbf{V} &= (I - Y)^{-1} Z\mathbf{b} & (8) \end{aligned}$$

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 $b_i = -I_i/\Gamma_i$, 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 [3] and constitute the <u>bookkeeping information</u> for the random walks. In particular, [3] 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 (8) has a complexity of $O(n^2)$. We present a more computationally efficient method.

IV. The Incremental Solver

A. 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:

$$(G + \Delta G)(\mathbf{V} + \Delta\mathbf{V}) = \mathbf{E} + \Delta\mathbf{E} \tag{9}$$

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:

$$\begin{aligned} G\Delta\mathbf{V} + \Delta G\mathbf{V} &\approx \Delta\mathbf{E} \\ \text{i.e., } G\Delta\mathbf{V} &= \Delta\hat{\mathbf{E}}, \end{aligned} \tag{10}$$

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

The significance of Equation (10) 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, $\Delta\hat{\mathbf{b}}$:

$$(\Delta\hat{\mathbf{b}})_i = \Delta\hat{b}_i = \Delta\hat{E}_i/\Gamma_i, \tag{11}$$

The result of incremental analysis can be obtained by substituting the new right hand side into Equation (7).

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 as described in Section II, 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 [2], 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.

B. 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 (7), (10), and (11):

$$\Delta\mathbf{V} = Y\Delta\mathbf{V} + Z\Delta\hat{\mathbf{b}} \tag{12}$$

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

$$\Delta V_1 = \sum_{j=1}^{N} z_{1j}\Delta\hat{b}_j \tag{13}$$

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:

$$\text{i.e., } \Delta V_i = \sum_{j=1}^{i-1} y_{ij}\Delta V_j + \sum_{j=i}^{N} z_{ij}\Delta\hat{b}_j, \tag{14}$$

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 (13). In such a case, Equation (7) would become

$$\Delta\mathbf{V} = \hat{Z}\Delta\hat{\mathbf{b}}, \tag{15}$$

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 [11]. A random walk solver that works with this $\hat{Z}$ matrix can be very inefficient, and it has been shown in [2] 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*.

C. 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 (12)). 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 (12) yields:

$$\Delta\mathbf{V} = (I - Y)^{-1} Z\Delta\hat{\mathbf{b}} = (I + Y + Y^2 + Y^3 + \cdots)Z\Delta\hat{\mathbf{b}}, \quad (16)$$

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 (15) and comparing it with Equations (8) and (16), it is easily seen that:

$$\hat{Z} = (I - Y)^{-1} Z = Z + YZ + Y^2 Z + \cdots \quad (17)$$

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

$$\begin{aligned} \hat{Z}_i^{(0)} &= Z_i & (18)\\ \hat{Z}_i^{(1)} &= (Z + YZ)_i \\ &= Z_i + \sum_{j=1}^{i-1} y_{ij} Z_j & (19)\\ \hat{Z}_i^{(2)} &= (Z + YZ + Y^2 Z)_i \\ &= Z_i + \sum_{j=1}^{i-1} y_{ij} Z_j + \sum_{j=1}^{i-1}\sum_{k=1}^{j-1} y_{ij} y_{jk} Z_k & (20)\\ &\vdots \end{aligned}$$

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, for 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.,

$$\delta V_k^{(q)} = (\Delta V_k^{(q)} - \Delta V_k^{(q-1)}), \quad (21)$$

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 1 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

$$\delta V_i^{(l+1)} = \sum_{j=1}^{i-1} y_{ij} \delta V_j^{(l)}, \quad (22)$$

Proof: Truncating Equation (16) at the k^{th} power of Y, we have:

$$\Delta\mathbf{V}^{(k)} = (I - Y)^{-1} Z\Delta\hat{\mathbf{b}} = (Z + YZ + \cdots + Y^{(k)} Z)\Delta\hat{\mathbf{b}} \quad (23)$$

This implies that

$$\delta\mathbf{V}^{(l)} = (\Delta\mathbf{V}^{(l)} - \Delta\mathbf{V}^{(l-1)}) = Y^{(l)} Z\Delta\hat{\mathbf{b}} \quad (24)$$

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 (22) as:

$$\Delta V_i^{(l+1)} = \Delta V_i^{(l)} + \sum_{j=1}^{i-1} y_{ij} \delta V_j^{(l)}, \quad (25)$$

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 $O(n\cdot \mid$ RoI $\mid)$, where $\mid$ RoI $\mid$ is the size of the RoI.

D. 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,

$$e_{V_i^{(l)}} = \left|\delta V_i^{(l)}\right| < \epsilon \quad (26)$$

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 (22), $\delta V_j^{(l)} = 0$ 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 (26) as $\left|\sum_{j=1}^{i-1} y_{ij} \delta V_j^{(l-1)}\right| < \epsilon$, i.e.,

$$\left|\sum_{j\in S} y_{ij}\delta V_j^{(l-1)} + \sum_{j\in\bar{S}} y_{ij}\delta V_j^{(l-1)}\right| < \epsilon \quad (27)$$

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) $y_{ij} \geq 0 \ \forall \ i, j$.

E. Incremental Solver Algorithm

In practice, the random walk solver provides a reasonable trade-off between accuracy and speed [2], 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 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 (22). 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

TABLE I
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

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 $|\text{RoI}| \times |\text{RoI}|$ system of equations, as illustrated in Figure 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 [12] 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 $\ll n$.

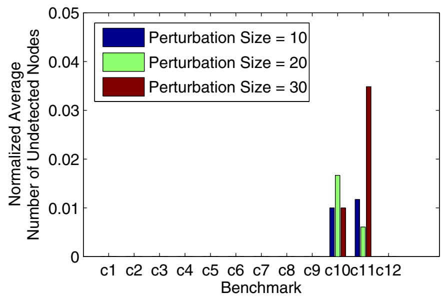
V. EXPERIMENTAL RESULTS

Our algorithm is tested and compared on a UNIX machine with 2GHz processor and 2GB RAM, and applied to 12 benchmarks summarized in Table I. 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.

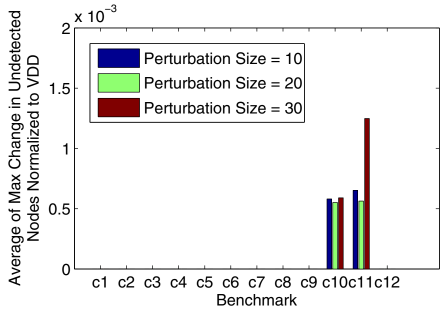
To demonstrate the accuracy of the RoI found by the proposed algorithm, we insert various perturbations to the benchmarks and find the corresponding RoI using the proposed algorithm as well as an exact direct solver. The *exact RoI* is the set of nodes for which the exact solution is perturbed by more 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 (11), 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 (11). 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



(a) Normalized number of undetected nodes



(b) Maximum change in the voltage of undetected nodes

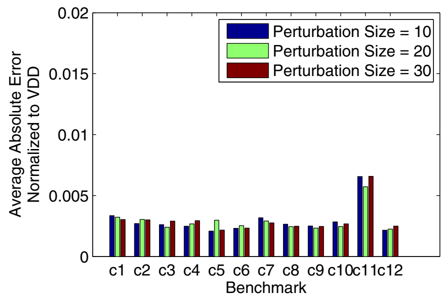
Fig. 3. Number of undetected nodes, normalized to the exact RoI size, and the maximum change in their voltage, for various perturbation region sizes (tolerance = $1\% V_{DD}$, perturbation value uniformly distributed in $(0, 0.1)$, averaged over 20 perturbations).

stopping criterion used to terminate the substitutions.

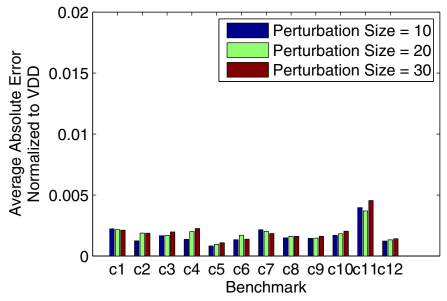
Figure 3(a) shows the number of undetected nodes, normalized to the exact RoI size, versus the size of the perturbed region. The numbers shown in all plots are averaged over 20 different sets of perturbations. This figure indicates that the number of undetected nodes is equal to zero for benchmarks $c1$ 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 3(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(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(a) and 4(b). Similarly, Figures 5(a) and 5(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 II against a very efficient public-domain iterative solver that uses an approximate conditioner based on random walks [3] 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 [3], and the remaining columns are related to our incremental solver. The bookkeeping time is the time required for

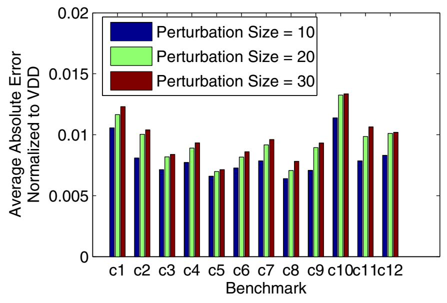


(a) Before refinement

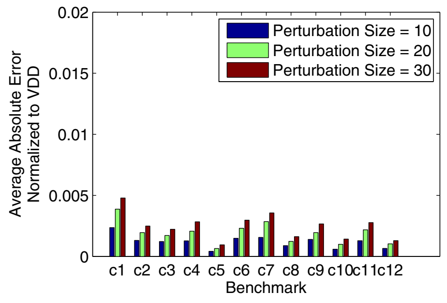


(b) After refinement

Fig. 4. 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

Fig. 5. 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)

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 [3] for c1 is $20 \times 0.29/2.75 = 2.1\times$ (Note that [3] 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 [3]), this expense disappears; the speedups are correspondingly greater: e.g., they approach $20\times$ for c12.

TABLE II
RUNTIME COMPARISONS (TOLERANCE = $1\%\ V_{DD}$, PERTURBATION REGION SIZE = 20, 30% PERTURBATION)

	[3] (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×

Table II 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 [3] increases accordingly while the time for the incremental solver remains the same.

REFERENCES

[1] H. Eisenmann and F. M. Johannes. Generic global placement and floorplanning. In *Proc. DAC*, pages 269–274, 1998.
[2] H. Qian, S. R. Nassif, and S. S. Sapatnekar. Random walks in a supply network. In *Proc. DAC*, pages 93–98, 2003.
[3] H. Qian and S. S. Sapatnekar. A hybrid linear equation solver and its application in quadratic placement. In *Proc. ICCAD*, pages 905–909, 2005.
[4] E. Chiprout. Fast flip-chip power grid analysis via locality and grid shells. In *Proc. ICCAD*, pages 485–488, 2004.
[5] M. Zhao et al. Hierarchical analysis of power distribution networks. In *Proc. DAC*, pages 150–155, 2000.
[6] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition, 1996.
[7] L. T. Pillage, R. A. Rohrer, and C. Visweswariah. *Electronic Circuit and System Simulation Methods*. McGraw-Hill, New York, NY, 1994.
[8] Y. Fu et al. A novel technique for incremental analysis of on-chip power distribution networks. In *Proc. ICCAD*, pages 817–823, 2007.
[9] Y. Ye, Z. Zhu, and J. R. Philips. Generalized krylov recycling methods for solution of multiple related linear equation systems in electromagnetic analysis. In *Proc. DAC*, pages 682–687, 2008.
[10] Y. Shi, W. Yao, J. Xiong, and L. He. Incremental and on-demand random walk for iterative power distribution network analysis. In *Proc. ASPDAC*, pages 185–190, 2009.
[11] W. Wasow. A note on the inversion of matrices by random walks. *Mathematical Tables and Other Aids to Computation*, 6(38):78–81, 1952.
[12] LAPACK. http://www.netlib.org/lapack/.