Incremental and On-demand Random Walk for Iterative Power Distribution Network Analysis *

Yiyu Shi† Wei Yao†
†Electrical Engineering Dept., UCLA
† Los Angeles, California, 90024

Jinjun Xiong* Lei He†
*IBM Thomas J. Watson Research Center
*Yorktown Heights, New York, 10598

Abstract— Power distribution networks (PDNs) are designed and analyzed *iteratively*. Random walk is among the most efficient methods for PDN analysis. We develop in this paper an incremental and on-demand random walk to reduce iterative analysis time. During each iteration, we map the design changes as positive or negative random walks for observed nodes. To update PDN analysis result, we only need to apply these extra positive or negative walks, instead of doing all walks from scratch. We show that different execution orders for these walks do not affect accuracy but do affect the runtime because of the cancellation between positive and negative walks. Considering this cancellation effect, we optimize the walk order by solving a min-energy electromagnetic particles placement problem and, as a result, further reduce the runtime to about $8\times$ compared to the worst order. Experiments show that, compared to random walk from scratch, our algorithm has similar accuracy but reduces the iterative analysis time by up to $18\times$ for on-chip PDN sizing, and by up to $13\times$ for package ball assignment with substrate routing. In addition, our incremental random walk has a linear time complexity with respect to the number of observed nodes and is more suitable for on-demand analysis, compared to random walk from scratch and its big warm-up cost.

I. Introduction

The PDN design is becoming essential to the success of high performance VLSI chips. With process scaling and decreasing supply voltage, the severe IR drop problem limits the overall circuit performance. A well-designed supply network is therefore an indispensable part of current VLSI design. However, the design of PDN is largely an iterative task, and is usually improved through a series of verification and correction steps after the initial design is done. During each iteration, if the whole network has to be re-evaluated for design modifications, the runtime penalty is huge because of the large scale of PDN. Therefore, considering the design change, an efficient and accurate PDN analysis method is needed to reduce the runtime.

One way to handle design change is that if we know all the places that need to be changed in advance, they can be modeled as symbols and a macromodel can be built accordingly. Existing methods belonging to this type include the symbolic model order reduction method [1] and the hierarchical random walk [2]. However, as pointed out in [3], it is very difficult to mark out all the places to be changed *prior* to the simulation.

The other type of approach, on the other hand, does not require a priori information about design changes. These methods make use of the simulation result of old design and perform $incremental$ updates based on it. Such methods include large change sensitivity methods [4] and the recently proposed fictitious domain method [3].

Although these two type of methods take design uncertainties into consideration, unfortunately, they all still require solving for all nodes in the PDN. A full PDN usually possesses millions of nodes and is time consuming to solve. On the other hand, people are usually interested in a particular selected subset of nodes, depending on the application. For example, for timing closure we may want to examine the voltage drop along the critical timing paths [5]. For decoupling capacitance budgeting, we may want to examine the noise on particular nodes throughout the optimization [6]. In all these cases, on-demand analysis procedure is desired. Here, on-demand means that only selected nodes in the PDN are analyzed.

Three types of methods have been developed for on-demand analysis: Pade-type model order reduction techniques were proposed in [7–10], where the reduced model retains the input and output ports as well as the moments of the transfer functions. But these reduction techniques are not effective in handling PDN because of the numerous interface nodes and high density coupling effects. Another type of method is multi-grid partition or reduction [11–13]. Kozhaya [11] approximates the PDN by a smaller but less accurate grid, solves it in reduced space and then projects it back. But this method may suffer from accuracy loss if high granularity is used during approximation. Both Chiprout [12] and Singh [13] use partition-based techniques to hierarchically analyze the PDN. But their accuracy cannot be guaranteed because they use artificial boundary conditions for inter-dependence between sub-circuits. The last type of method is the random-walk-based simulator, which has been proposed in [2]. It can easily calculate the voltage at any particular node by performing random walks starting from that node. An efficient *incremental* and *on-demand* PDN analysis method, however, has not been reported.

In this paper, we propose a novel incremental random walk algorithm with on-demand analysis. We map the design changes as positive or negative random walks for the nodes under observation. We also prove that a proper order of walks re-

*This paper is partially supported by NSF CAREER award CCR-0306682 and a UC MICRO grant sponsored by Actel and Fujitsu. Address comments to lhe@ee.ucla.edu.

duces the net random walks without losing accuracy because of the cancellation between positive and negative walks. We optimize the walk order by solving a min-energy electromagnetic particles placement problem and, as a result, further reduce the runtime to about $8\times$ compared to the worst order. Compared to random walk from scratch, our incremental algorithm has a similar accuracy but reduces the iterative analysis time by up to $18\times$ for on-chip PDN sizing that involves element's value changes, and by up to $13\times$ for package ball assignment with substrate routing that involves mainly topology changes. In addition, on-demand analysis has a runtime linear with respect to the number of observed nodes for incremental random walk, but not for non-incremental random walk.

The remaining paper is organized as follows: Section II reviews the random walk algorithm. In Section II, we present our novel incremental algorithm and demonstrate our electromagnetic model for the node ordering problem. Experimental results are presented in Section III and concluding remarks are given in Section IV.

II. Preliminary

Here we briefly review the random walk method from [2]. For any node i in a circuit, we can write down the following equation according to the Kirchhoff's current law

$$I_i = \sum_j g_{i,j}(v_i - v_j), \tag{1}$$

where v_i is the voltage response at node i, $g_{i,j}$ is the conductance between node i and j, and I_i is the injection current at node i. The summation is over all the adjacent nodes of node i, including the ground if necessary. Reorganize (1) as

$$v_i = \sum_j \frac{g_{i,j}}{g_i} v_j + \frac{I_i}{g_i}, \tag{2}$$

where $g_i = \sum_j g_{i,j}$ is the sum of conductance incident at node i.

(2) can be interpreted as follows: if we want to compute the voltage response v_i at a particular observation node i, we can perform a random walk to one of its adjacent node v_j with probability

$$p_{i,j} = \frac{g_{i,j}}{g_i} \tag{3}$$

and pays for an amount of $\frac{I_i}{g_i}$. Then the walker continues the walk from node j in a similar way, until arriving home (the node with known response, for example, the nodes connecting to Vdd). When a home is arrived, the walker gets an award equal to the response at that node.

It is proved in ([2]) that with enough number of random walks from the observation node i, the average payment of each walk will converge to v_i. In other words, during the random walk we record the number of visits at each node N_j, and after all the walks are done from node i, we can get

$$v_i = \frac{1}{N_0}\left(\sum_j N_j \frac{I_j}{g_j} + \sum_k N_k v_k\right), \tag{4}$$

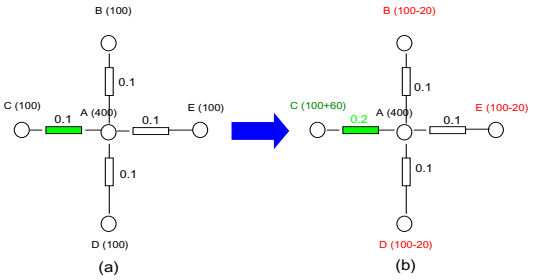

Fig. 1. Impact of design changes on random walk results.(a)original network (b)modified network

where the first summation is over all the nodes with unknown response, and the second is over all the home nodes and N_0 is the total number of random walks started from node i. (4) can be uniformly written as

$$v_i = \frac{1}{N_0}\sum_l N_l \mathcal{P}_l, \tag{5}$$

where the summation is over all the nodes and $\mathcal{P}_l$ is the "payment" of node l and is defined as

$$\mathcal{P}_l = \begin{cases} \frac{I_l}{g_l} & \text{if node l is not a home node} \\ v_k & \text{if node l is a home node} \end{cases} \tag{6}$$

From (5) we can see that as long as we can record the correct number of visits N_l at each node for the random walks starting from the observation node, we will be able to compute the correct response. Below we will discuss how to update them to get the correct results for design changes.

III. Incremental Random Walk

Before we discuss the general cases, let's first look at an example. Part of a large network is showed in Fig. 1 (a): all the four conductance connected at node A has an equal value of $0.1\Omega^{-1}$, and accordingly the walker has the same probability to go any direction. As a result, if node A is visited 400 times, we would expect that, on average, the walker will visit each of the adjacent nodes 100 times. Now suppose the conductance g_{ac} is changed to $0.2\Omega^{-1}$ as shown in Fig. 1(b). We would expect that out of the 400 walks starting from node A, node C would be visited 160 times, and each of the remaining three nodes would be visited 80 times. Therefore, to get the correct solution, we need to start 60 extra random walks from node C, and reduce 20 random walks from the remaining three nodes. If there are no other changes in the network, we can expect that, after the addition or removal of the random walks, we can get the correct visit number for all the nodes we visited, and get the correct response at the observation node. From this example, the total number of walks we need to add or reduce is 120, which is much smaller than the total walks needed to start from scratch, which is 400. Note that we have to keep the number of visits at node A (400) the same: A is visited by walker from the adjacent nodes, and if the number of visits at A changes, it will cause further changes at the adjacent nodes, thus causing a propagation of the change and increasing the complexity.

A. Algorithm Overview

In general, we denote the probability for the walker to walk from node i to its adjacent node j as $p_{i,j}$ ($1 \leq j \leq d_i$). Without loss of generality, we assume that due to the design change, $p_{i,j}$ is changed to $\tilde{p}_{i,j}$. Note that the topology change is also included in such formulation: $p_{i,j} = 0$ corresponds to the addition of a conductance, while $\tilde{p}_{i,j} = 0$ corresponds to the removal of a conductance. Finally, we denote N_j as the number of visits at node j before the change. Accordingly, the number of visits at node i needs to be changed by

$$\Delta N_c^i = \sum_j (\tilde{p}_{j,i} - p_{j,i}) N_j, \tag{7}$$

where the summation is over all the adjacent nodes of i with probability change. If ΔN_c^i is positive, then we need to start that amount of extra random walks from node i. Otherwise, we need to reduce $-\Delta N_c^i$ number of existing random walks from node i.

Note that (7) is only true when N_j is large in the statistical meaning. On the other hand, a small N_j indicates that random walks starting from the observation node have very small chance of visiting node j. Accordingly, we do not need to process the change as it only has a minor impact on the solution. This observation is very important as it can help us to significantly reduce the total number of new random walks required, especially in the case of flip chips with multiple power bumps where all the random walks are local.

Algorithm 1 Incremental Random Walk.

```
INPUT: Initial visit number at each node N_i, the network topology, the initial and the
updated probability list P_o = {p_{i,j}} and P_n = {p̃_{i,j}};
OUTPUT: The number of random walks need to be increased or reduced at each node
ΔN_i;
for each node with changed incident conductance do
  ΔN_i = Σ_j N_j × (p̃_{j,i} − p_{j,i}); The summation is over all the adjacent nodes
  of i;
  if ΔN_i ≥ N_o then
    Start ΔN_i number of random walks from node i;
  else if ΔN_i ≤ −N_o then
    Remove −ΔN_i number of random walks from node i;
  end if
end for
if Solution does not meet the stopping criterion [2] then
  Perform extra random walks starting from observation nodes;
end if
```

The overall algorithm is presented in Algorithm 1, where N_o is the threshold to decide whether the change need to be processed or not. According to Algorithm 1, to perform incremental random walk, we only need to start ΔN_i number of new random walks if ΔN_i is positive, and remove $-\Delta N_i$ number of existing random walks if negative. Note that we need to use the initial probabilities before the change for both positive and negative cases, as we are in essence making an adjustment to the initial random walk results. If the updated solution does not satisfy the error bound [2], extra random walks starting from observation node will be needed.

Obviously, it is straightforward to start a given number of random walks from a particular node; However, it is still not clear how to remove a given number of existing random walks, as it is not possible to keep track of the path for each walk due to the high space and time complexity.

B. Removal of Existing Random Walks

In this section, we propose a statistical approach to remove the existing random walks, which does not require any information about the existing walks. The algorithm is summarized in Algorithm 2.

Algorithm 2 Algorithm to reduce ΔN_c^i number of random walks from node i.

```
INPUT: Node i, the number of random walks need to be reduced ΔN_c^i;
OUTPUT: Updated number of visits at each node;
for k = 0; k ≤ N_c^i, k++ do
  Start a random walk from node i; currentNode = i;
  while currentNode ≠ home do
    Subtract the number of visits at current node by one;
    Choose one of the adjacent nodes as currentNode according to the initial proba-
    bilities on each edge;
  end while
end for
```

The main idea of the proposed algorithm is as follows: instead of removing ΔN_c number of random walks starting from node i directly, we perform ΔN_c number of extra random walks from node i. Whenever a node is visited, the number of visits at that node is decreased by one, as opposed to the normal random walk which should increased the number of visits by one. We claim that the effect of such a strategy is equal to the removal of ΔN_c random walks from node i. This is guaranteed by the following theorem.

Theorem 1 *If we start N_c^i number of random walks from node i, and then reduce ΔN_i^c number of random walks from node i by Algorithm 2, then*

$$v_i \approx \frac{1}{N_c^i - \Delta N_c^i} \sum_j N_j \mathcal{P}_j \tag{8}$$

where N_j is the updated number of visits at node j, and $\mathcal{P}_j$ is the corresponding "payment" as defined in (6). Compare with (5), this is equal to the situation where a total number of $N_c^i - \Delta N_c^i$ random walks have been started from node i.

PROOF. Denote $\tilde{N}_j$ as the number of visits at node j after N_i normal random walks, and $\Delta\tilde{N}_j$ as the number of visits at node j by Algorithm 3, then

$$N_j = \tilde{N}_j - \Delta\tilde{N}_j. \tag{9}$$

Moreover, from (5) we have

$$N_c^i v_i = \sum_j \tilde{N}_j \mathcal{P}_j, \tag{10}$$

and

$$\Delta N_c^i v_i = \sum_j \Delta\tilde{N}_j \mathcal{P}_j. \tag{11}$$

The correctness of (11) can be seen from the fact that by reversing the sign of the bookkeeping results, we are actually performing the normal random walks. From (9)-(11), Theorem 1 holds. □

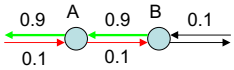

Fig. 2. Example to show the importance of the ordering.

C. Merging Positive and Negative Random Walks Through Walk Ordering

The runtime of the proposed incremental random walk algorithm is mainly decided by the total number of positive (random walk with positive bookkeeping) and negative (removal walk with negative bookkeeping) random walks from all nodes. We show, however, that this number can be significantly reduced by cleverly merging the positive and negative random walks with a proper ordering strategy.

To illustrate this point, suppose we start a positive random walk from some node A, and the walker arrived at a node B. Node B requires some negative random walks. Then naturally, we can stop the random walk at node B, although B is not home, and decrease the total number of negative random walks required at node B by one. The correctness of such method is obvious from the meaning of negative random walk: it tries to remove one of the existing random walks from node B, so we can just stop the current positive random walk.

The merging of random walks makes it critical to decide the order of the nodes to start random walks for the efficiency purpose. This can be seen from the the example shown in Fig. 2, which is part of a large network. If we start ΔN number of positive random walks from node A and the same number of negative random walks from node B, then starting from node A first would require the total number of $2\Delta N$ walks approximately, while from node B first only requires ΔN walks with each path of length 1, almost twice the difference in runtime.

We also perform test on a real industrial example with 1027 nodes and 10% of the resistance are changed by between 10% and 100%. We randomly permute the order of the nodes to start random walk from and perform Monte Carlo Simulation. The runtime distribution histogram is shown in Fig.3. From the figure we can see that different ordering can cause up to $4\times$ difference in runtime. Therefore, a good ordering strategy is critical for the efficiency of the algorithm.

Important as the ordering is, it is difficult to solve the ordering problem directly. However, we can analogize the problem to a classical problem in electro-magnetics and propose a heuristic algorithm to solve it. The intuition is as follows: We need to encourage more positive and negative random walks to be merged, and accordingly, we need to order those nodes according to the strength of them being "attracted" by other nodes with the opposite sign of random walks. This is quite similar to the attractive force between particles with positive and negative charges in electro-magnetics.

Accordingly, we can place a particle with positive (negative) charge at each node that requires positive (negative) random walks, and the charge amount is equal to the number of random walks required, i.e.,

$$q_i = \Delta N_i, \tag{12}$$

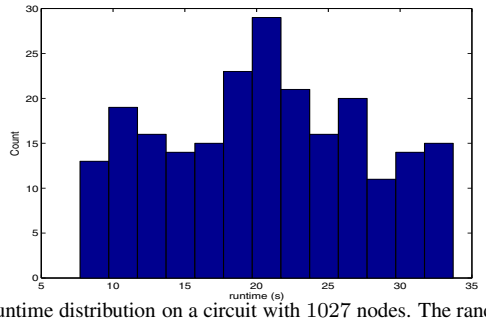

Fig. 3. Runtime distribution on a circuit with 1027 nodes. The random walks are started according to randomly generated order.

for any particular node i. An example is shown in Fig. 4.

Next, we define the directed distance from particle A to particle B as

$$d(A \to B) = \sum_{\phi_i \in \{\phi_c\}} \prod_{edge(j\to k)\in\phi_i} \frac{1}{p_{j\to k}} \tag{13}$$

where the summation is over ϕ_c, the set of all paths from node A to node B, and $p_{j\to k}$ is the probability from node j to node k on the paths. Note that in general, the distance is not commutative, i.e.,

$$d(A \to B) \neq d(B \to A) \tag{14}$$

The total number of directed paths from node A to node B is infinite, and therefore, (13) only have theoretical value. To practically compute the distance, we can approximate it as

$$d(A \to B) \approx \sum_{\phi_i \in \{\phi_c^K\}} \prod_{edge(j\to k)\in\phi_i} \frac{1}{p_{j\to k}} \tag{15}$$

where $\{\phi_c^K\}$ is the set of the paths with K largest inverse probability product. In other words, we only compute the largest K items and use it to approximate the total sum.

Note that

$$\prod_{edge(j\to k)\in\phi_i} \frac{1}{p_{j\to k}} = e^{\sum_{edge(j\to k)\in\phi_i} ln\frac{1}{p_{j\to k}}}. \tag{16}$$

Therefore, with the weight of each edge labeled as $ln\frac{1}{p_{j\to k}}$ ($\in (0,1)$), (15) can be efficiently solved by the K shortest path algorithms such as the Dijkstra's algorithm [14]. The value of K can be decided such that the the $(K+1)^{th}$ path has a very small probability product, i.e,

$$\prod_{edge(j\to k)\in\phi^{K+1}} \frac{1}{p_{j\to k}} < \epsilon, \tag{17}$$

where ϕ^{K+1} is the path with the $(K+1)^{th}$ largest probability product, and ϵ is a user-defined small number.

To further speedup, we can randomly sample a few paths, and use it as an estimation for the distance. For example, in Fig. 4 (b), the three sampled paths are shown in black, and the sum of the products of the inverse probability along each path can be used as an estimation of the distance from the positive charge to the negative charge.

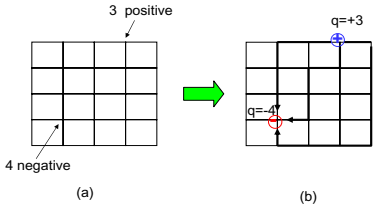

Fig. 4. Illustration of the problem mapping from (a) the nodes with positive/negative random walks to (b) the corresponding charge distribution.

TABLE I
RUN TIME COMPARISON FOR THE ORDERING ALGORITHM

Consider changes of elements' values			
Case	magnitude of changes	Runtime(sec)	
		random [min, max]	ordering alg.
1	1%	[12.7, 74.9]	12.9
2	5%	[13.9, 83.3]	13.6
3	10%	[19.7, 94.6]	20.5
4	40%	[24.4, 139.5]	24.9
5	100%	[31.2, 236.6]	32.8
Consider changes of topology			
Case	% of elements being changed	Runtime(sec)	
		random [min, max]	ordering alg.
1	1%	[1.7, 2.9]	1.8
2	5%	[4.9, 11.1]	5.2
3	10%	[9.6, 16.0]	10.4
4	30%	[42.3, 327.1]	44.2

Given the charge distribution and their distance, from electromagnetic theory, we know that the potential at a particular node A can be computed as

$$Po(A) = \sum_{\text{node i has opposite charge with A}} k\frac{q_i}{d(A \to i)}. \qquad (18)$$

where k is a constant. We can verify that the absolute value of $Po(A)$ actually reflects the number of random walks starting from node A that can be merged. Therefore, we can compute the potential for all the nodes, and sort them according to the descending order of the absolute values of potentials.

More importantly, we know that the potential has the locality property, i.e., when computing the potential for a particular node, we can set a threshold for the distance. Only the particles within a given distance need to be considered. This can further improve the runtime of the proposed algorithm.

IV. EXPERIMENTAL RESULTS

Algorithms presented in this paper have been implemented inside Ngspice package [15] using C language. In this section, we report experiments run on a UNIX workstation with Pentium IV 2.66G CPU and 2G RAM. The relative errors are all based on SPICE simulation.

A. Walk Ordering Algorithm Validation

To start with, we first verify the effectiveness of our node ordering technique. We apply our algorithm to a PDN design as RLC network with 102471 nodes and 215 ports. We consider two different kinds of design changes here: (1) magnitude change, where we modify the RLC elements value and 15% elements are affected with 1% to 100% changes on values. And (2) topology change, where we add or remove up to 30% RLC elements. For each test case, we permute the ordering of random walks and obtain the min and max runtime by 10000 Monte Carlo simulations.

Based on runtime reported in Table I, our algorithm finds an ordering with analysis time close to the minimum one, and the runtime difference due to ordering is up to 8x. When changes are limited, the runtime variation is small. This is because that the random walk is almost isotropic in such cases, and different ordering should not bring much difference. Note that our ordering algorithm is used for all results discussed below.

TABLE II
PDN TESTBENCH DESIGN INFORMATION.

Design	# Nodes	# Resistors	# Ports	% of Changes
#1	5880	7105	291	10.1%
#2	19964	27005	779	8.0%
#3	68380	97625	1327	12.6%
#4	100372	159644	4960	11.3%
#5	325179	407960	9137	11.0%

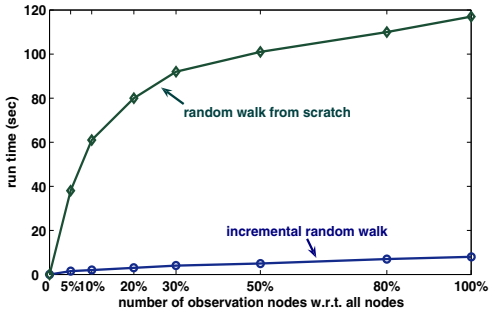

Fig. 5. Runtime v.s. the percentage of observation nodes for Design #2

B. Iterative Design Example I : PDN sizing

We first illustrate the impact of on-demand analysis for both non-incremental (random walk from scratch) and incremental random walks. We use Design#2 in Table II as an example. As shown by Fig. 5, non-incremental random walk has a big warm-up cost. For 10% observation nodes, the runtime only reduced by 50%. In contrast, on-demand analysis for incremental random walk has a time complexity linear with respect to the number of observation nodes. The speedup is 8× for 10% observation nodes and is 1.8× for 50% observation nodes. The higher speedup for fewer observation nodes is important, as design changes in practice is often limited for each design iteration.

We then consider iterative PDN sizing for circuits summarized in Table II. The runtime and accuracy between our incremental random walk with the random walk from scratch [2] and also the matrix solver in Ngspice [15] are reported in Table IV. From the table, with the increasing size of PDN, the runtime for matrix factorization based solver increases dramatically. On the other hand, random walk based algorithms take advantage of the small number of observation nodes and reduce the run time significantly through on-demand analysis. Moreover, Table IV shows that, with the same number of observation nodes, incremental random walk can obtain another runtime reduction up to 18×, compared to random walk from

TABLE III
ITERATIVE BALL ASSIGNMENT RUNTIME COMPARISON.

Design	# Balls	# Nets	noise(V*Ms)		total analysis time(sec)		% of incremental runs
			initial	final	RW [2]	Incre. RW	
#1	128	76	49.7	11.2 (22.5%)	8437	1278 (1/7×)	74%
#2	128	89	83.5	19.7 (23.6%)	11324	1899 (1/6×)	76%
#3	128	121	42.0	12.8 (30.5%)	19006	2431 (1/8×)	79%
#4	256	63	32.2	8.99 (27.9%)	23847	1781 (1/13×)	83%
#5	256	172	74.1	14.7 (19.8%)	29501	3673 (1/8×)	84%

TABLE IV
RUNTIME AND ACCURACY FOR PDN SIZING.

Design	runtime(sec)			relative error	
	NgSpice [15]	RW [2]	Incre. RW	RW [2]	Incre. RW
#1	11.6	15.0	2.0 (1/7.5×)	1.05%	1.56%
#2	282	37.8	2.8 (1/14×)	1.66%	1.89%
#3	41 min	89.9	4.9 (1/18×)	1.89%	1.22%
#4	387 min	163	11.2 (1/15×)	1.02%	0.78%
#5	N/A	328	29.7 (1/11×)	N/A	N/A

scratch, at a similar accuracy.

C. Iterative Design Example II : Package Ball Assignment with Substrate Routing

Finally, to study the speedup with topology changes that could be challenging for incremental random walk, we apply our incremental random walk algorithm to simultaneous package ball assignment and substrate routing. Note that different ball assignments correspond to different substrate routing and accordingly, different cross-talk noise between routes. We can reduce the noise between different substrate routes through iterative ball assignment optimization, which could lead to substantial topology changes in the RLC network. During each iteration, incremental random walk algorithm can be used to efficiently calculated the noise for the I/O ports.

We report iterative ball assignment results for several design cases in Table III. During iteration, random walks from scratch may be invoked *automatically* if the error bound can not be met by incremental walk within given bound on total random walks. The percentage of actual incremental runs are also listed in Table III. From the table, after iterative design, the noise for each case is reduced significantly. With incremental random walk, the total analysis time is reduced by up to $13\times$, compared to random walk from scratch.

V. CONCLUSION AND DISCUSSION

In this paper, we develop an incremental random walk with on-demand analysis. PDN topology changes and element value changes are represented as positive or negative random walks for observed nodes. We also prove that a proper order of walks reduces the net random walks without losing accuracy because of the cancellation between positive and negative walks. We optimize the walk order by solving a min-energy electromagnetic particles placement problem and, as a result, further reduce the runtime to about $8\times$ compared to the worst order. Compared to random walk from scratch, our incremental algorithm has a similar accuracy but reduces the iterative analysis time by up to $18\times$ for on-chip PDN sizing, and by up to $13\times$ for package ball assignment with substrate routing. To emphasize the effectiveness of our algorithm, note that PDN sizing involves only element value changes, while different ball assignment leads to potentially extensive topology changes. For these two extreme cases, our algorithm obtains similar levels of speedup.

Experiments also show that on-demand analysis has a runtime linear with respect to the number of observed nodes for incremental random walk, but not for non-incremental random walk. The difference is about $2\times$ when the number of observed nodes is limited. Therefore, the on-demand analysis speedup by incremental random walk versus random walk from scratch could be many orders of magnitude for a large scale circuit with localized changes during each design iteration.

REFERENCES

[1] G. Shi, B. Hu, and C. J. R. Shi, "On symbolic model order reduction," *IEEE Trans. on CAD*, 2006.

[2] H. Qian, S. R. Nassif, and S. S. Sapatnekar, "Power Grid Analysis Using Random Walks," *IEEE Trans. on CAD*, 2005.

[3] Y. Fu, R. Panda, B. Reschke, S. Sundareswaran, and M. Zhao, "A novel technique for incremental analysis of on-chip power distribution networks," in *IEEE/ACM ICCAD*, 2007.

[4] L. T. Pillage, R. A. Roher, and C. Visweswariah, *Electronic circuit and system simulation methods*. McGRAW-HILL Publishers, 1994.

[5] R. Ahmadi and F. N. Najm, "Timing analysis in presense of power supply and ground voltage variations," in *IEEE/ACM ICCAD*, 2003.

[6] Y. Shi, J. Xiong, C. C. Liu, and L. He, "Efficient decoupling capacitance budgeting considering operation and process variations," in *IEEE/ACM ICCAD*, 2007.

[7] E. Chiprout and T. V. Nguyen, "Power analysis of large interconnect grids with multiple sources using model reduction," in *European Conference on Circuit Theory and Design*, 1999.

[8] J. M. Wang and T. V. Nguyen, "Extended krylov subspace method for reduced order analysis of linear circuits with multiple sources," in *IEEE/ACM DAC*, 2000.

[9] T. Chen and C. C. Chen, "Efficient large-scale power grid analysis based on preconditioned krylov-subspace iterative methods," in *IEEE/ACM DAC*, 2001.

[10] Y. Shi, H. Yu, and L. He, "Samson: A generalized second-order arnoldi method for reducing multiple source linear network with susceptance," in *IEEE/ACM ISPD*, 2006.

[11] J. Kozhaya, S. R. Nassif, and F. N. Najm, "A multigrid-like technique for power grid analysis," *IEEE Trans. on CAD*, 2002.

[12] E. Chiprout, "Fast flip-chip power grid analysis via locality and grid shells," in *IEEE/ACM ICCAD*, 2004.

[13] J. Singh and S. S. Sapatnekar, "Partition-based algorithm for power grid design using locality," *IEEE Trans. on CAD*, 2006.

[14] J. Kleinberg and E. Tardos, *Algorithm Design*. Addison-Wesley, 2005.

[15] in *http://ngspice.sourceforge.net*.