Deterministic Random Walk Preconditioning for Power Grid Analysis

Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology, Chicago, IL 60616, USA

ABSTRACT

Iterative linear equation solvers depend on high quality preconditioners to achieve fast convergence. For sparse symmetric systems arising from large power grid analysis problems, however, preconditioners generated by traditional incomplete Cholesky factorizations are usually of low quality, resulting in slow convergence. On the other hand, although preconditioners generated by random walks are quite effective to reduce the number of iterations, it takes considerable amount of time to compute them in a stochastic manner. In this paper, we propose a new preconditioning technique for power grid analysis, named *deterministic random walk*, that combines the advantages of the above two approaches. Our proposed algorithm computes the preconditioners in a deterministic manner to reduce computation time, while achieving similar quality as stochastic random walk preconditioning by modifying fill-ins to compensate dropped entries. We have proved that for such compensation scheme, our algorithm will not fail for certain matrix orderings, which otherwise cannot be guaranteed by traditional incomplete factorizations. We demonstrate that by incorporating our proposed preconditioner, a conjugate gradient solver is able to outperform a state-of-the-art algebraic multigrid preconditioned solver on public IBM power grid benchmarks for DC power grid analysis, while potentially remaining very efficient for transient analysis.

1. INTRODUCTION

Power supply noise is a critical issue that must be addressed in modern chip designs since it may affect important design metrics including timing and power consumption [1, 29, 6, 14]. Modeling devices as current sources and power distribution networks with parasitics as RLC grids, power grid analysis [6, 24] leverages circuit simulation techniques to estimate such noises for designers to take further actions. Though the power grid analysis problems have been extensively studied, it remains extremely challenge to build efficient power grid analysis tools to match the increasing complexity associated with power delivery [25].

The key problem of power grid analysis is to solve the following sparse system of linear equations for noise vector $\mathbf{x}$ given system matrix A and excitation vector $\mathbf{b}$:

$$A\mathbf{x} = \mathbf{b}. \tag{1}$$

In this paper, we consider DC analysis and transient analysis when there is no mutual inductance and thus assume A to be a nonsingular symmetric diagonally dominant matrix with nonpositive off-diagonal elements [2], which also implies that A is symmetric positive definite. Although the above system could be solved by direct methods like Cholesky factorization [10, 3], since the exact factors are large in size, such methods are only suitable when there is ample memory and when the system should be solved for many excitation vectors. The state-of-the-art researches on power grid analysis can be separated in two groups. The efforts of the first group lead to efficient algorithms that can obtain approximated solutions by exploiting structures of the power grid, e.g. hierarchical analysis [34], multigrid [24, 16, 30, 38, 39, 7], random walks [26, 19], Poisson solver [28], frequency domain methods [35, 13], and model order reduction techniques [31, 17, 18]. The second group [2, 37, 27, 8, 33, 32, 4, 36], which attracts a lot of interests recently due to the demand to accurately characterize the static and dynamic behaviors of power grids [20, 21], utilize iterative solvers, especially preconditioned conjugate gradient (PCG), to further improve the solution accuracy.

It is well-known that the quality of the preconditioners has a huge impact on the performance of the iterative solvers. We are particularly interested in stochastic random walk preconditioning [27] as we observed that for public IBM power grid benchmarks [25, 20], on average the l_2 norm of the residue is reduced by half per PCG iteration, which leads to much less iterations than those recently reported for traditional incomplete Cholesky (IC) factorizations and various other preconditioning techniques [32, 4, 36]. Moreover, since the preconditioner itself is an approximated root-free Cholesky (LDL^T) factorization of the system matrix, each PCG iteration is less involving than algebraic multigrid (AMG) preconditioning [33]. As pointed out by the authors [27], the superior quality comes from the fact that the columns of L are computed independently so that errors will not accumulate as in traditional IC/ILU preconditioners. However, since to compute the columns independently requires to perform Monte Carlo simulations on the power grid, even when the walks are extensively reused [27], it takes much more time than other techniques [23, 15, 36] that compute the factorizations in a rather deterministic manner. Stochastic random walk preconditioning is therefore not suitable when the system is only solved for a few times, e.g. for DC analysis.

In this paper, we propose a new preconditioner, named *de-*

terministic random walk (DRW), that overcomes the above difficulty while achieving similar quality compared to stochastic random walk preconditioning. Our key observation is that if the columns of L are computed sequentially, a random walk can be decomposed into a few random walks whose probabilities are either easily known or already computed. This enables us to avoid Monte Carlo simulations, resulting in a deterministic algorithm that computes L efficiently. As our algorithm will usually locate more entries in L than stochastic random walk, we propose to keep only the largest few in absolute value per column to maintain the sparsity of L, and to compensate the dropped ones by *decreasing* the remaining ones. Such compensation scheme differs our preconditioner from various traditional IC/ILU preconditioners [23, 12, 22] where the off-diagonal elements of L are only allowed to be increased or dropped (increased to 0) in order to guarantee the correctness of the algorithm for A being a symmetric M-matrix. As a comparison, by leveraging random walks, we are able to prove the correctness of our algorithm when more flexible dropping and compensation schemes are adopted for certain matrix orderings. Our proposed preconditioner also differs from hierarchical random walks [26, 19]. They explicitly approximate the possibly dense Schur complement after partial factorization by Monte Carlo simulations, while we implicitly factor the Schur complement without actually generating it first.

We demonstrate that a PCG solver using our proposed DRW preconditioner can efficiently explore the trade-offs between the preconditioner size, the time to compute the preconditioner, and the time to solve a problem iteratively. Using similar preconditioner sizes as stochastic random walk preconditioning, we reduce the time to compute the preconditioner by 3.9× and the time to solve a problem iteratively by 2.4× on public IBM power grid benchmarks [25, 20], which is essential for efficient transient analysis. On the other hand, by using a preconditioner size similar to that of A, we further reduce the time to compute the preconditioner without degrading its quality dramatically. This leads to an efficient solver for DC analysis that is able to outperform the state-of-the-art AMG-PCG solver PowerRush [33].

The rest of this paper is organized as follows. Incomplete factorizations and stochastic random walk preconditioning are reviewed in Section 2. Our proposed DRW preconditioner is presented in Section 3. After experimental results are discussed in Section 4, Section 5 concludes the paper.

2. PRELIMINARIES

2.1 Matrices and Graphs

For a $n \times 1$ vector $\mathbf{x}$, we denote its ith element by x_i for $1 \le i \le n$, and a vector obtained from its ith to jth elements by $\mathbf{x}_{i:j}$. For a $n \times n$ matrix X, we denote its element on the row i and the column j by $x_{i,j}$ for $1 \le i \le n$ and $1 \le j \le n$. The submatrix of X spanning the rows i_1 to i_2 and the columns j_1 to j_2 is denoted by $X_{i_1:i_2,j_1:j_2}$. For ease of presentation, we opt to omit $_{:i_2}$ or $_{:j_2}$ if $i_1 = i_2$ or $j_1 = j_2$ respectively. We define $\mathcal{D}(X)$ to be the diagonal of X, i.e. $diag(x_{1,1}, x_{2,2}, \ldots, x_{n,n})$. We say that X is *column-wise diagonally dominant* if $x_{k,k} \ge \sum_{i \ne k} |x_{i,k}|$, $\forall k$, and that X is *unit lower triangular* if $x_{k,k} = 1$ and $X_{1:k-1,k} = 0$, $\forall k$.

We define the *extended matrix graph* of X to be a weighted directed graph $G^*_X = (V^*_X, E^*_X, w^*_X)$. The vertex set V^*_X consists of the vertices $1, 2, \ldots, n$ corresponding to the rows

$$A = \begin{bmatrix} 3 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 2 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

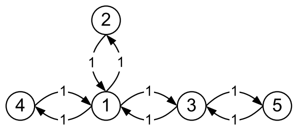

Figure 1: A matrix and its extended matrix graph

and columns of X and an additional vertex $n+1$. For any $x_{i,j} \ne 0$ where $i \ne j$, an edge (j,i) is introduced to E^*_X with the weight $w^*_X(j,i) = -x_{i,j}$. For any k satisfying $x_{k,k} \ne \sum_{i \ne k} x_{i,k}$, an edge $(k, n+1)$ is introduced with the weight $w^*_X(k, n+1) = x_{k,k} - \sum_{i \ne k} x_{i,k}$. Similarly, an edge $(n+1, k)$ is introduced with the weight $w^*_X(n+1, k) = x_{k,k} - \sum_{j \ne k} x_{k,j}$ if that weight is not zero. An example system matrix A and its extended matrix graph is shown in Fig. 1.

2.2 Incomplete LDL^T Factorizations

We then review incomplete Cholesky factorization in its root-free forms, i.e. incomplete LDL^T factorization, because of its close relationship to random walk based preconditioning techniques studied in [27] and in this paper.

Let n be the dimension of the system matrix A. Since A is symmetric positive definite (s.p.d.), it is well known [10] that there exists a unit lower triangular matrix $\hat{L}$ and a diagonal matrix $\hat{D}$ with all diagonal elements being positive such that $A = \hat{L}\hat{D}\hat{L}^T$. Since A is also a M-matrix (a matrix where the elements in its inverse are all nonnegative), its incomplete LDL^T factorization does exist [23] and can be computed iteratively for $k = 1, 2, \ldots, n$ as follows,

$$d_{k,k} = a_{k,k} - \sum_{j=1}^{k-1} l_{k,j} d_{j,j} l_{k,j}, \tag{2}$$

$$l_{i,k} = 0 \text{ or } \frac{1}{d_{k,k}}\Big(a_{i,k} - \sum_{j=1}^{k-1} l_{k,j} d_{j,j} l_{i,j}\Big), \forall k < i \le n. \tag{3}$$

The choices in Eq. (3) are determined by the dropping scheme, e.g. [15]. According to [22], the relationship between the exact and the incomplete LDL^T factorizations is,

$$d_{k,k} \ge \hat{d}_{k,k} > 0,\ 0 \ge l_{i,k} \ge \hat{l}_{i,k},\ 0 \ge d_{k,k} l_{i,k} \ge \hat{d}_{k,k} \hat{l}_{i,k}. \tag{4}$$

Eq. (4) indicates that the error due to dropped entries will accumulate and lead to a systematic difference between the exact and the incomplete factorizations, which will cause incomplete Cholesky preconditioners to perform unfavorably in a PCG solver. An intuitive idea to correct such difference is to decrease both $d_{k,k}$ and $l_{i,k}$ during factorization for compensation. However, if one attempts to do so within the factorization process outlined by Eq. (2) and (3), there is no guarantee that the computation could proceed, i.e. $d_{k,k} \ne 0$, and there is no guarantee that the obtained factorization will remain s.p.d., i.e. $d_{k,k} > 0$.

2.3 Stochastic Random Walk Preconditioning

We slightly modify the definition of random walks from [27] so that the outcome of stochastic random walk preconditioning will be an approximated LDL^T factorization of A.

Consider the matrix $T = A\mathcal{D}(A)^{-1}$, i.e. the matrix obtained from A by dividing its column k by $a_{k,k}$ for every k. A random walk can be defined in its extended matrix graph G^*_T treating edge weights as transition probabilities. Let $k \overset{C}{\rightsquigarrow} i$ be the set of random walks from k to i whose internal vertices satisfy the condition C for $1 \le k \le n$ and $1 \le i \le n+1$. Since A is symmetric nonsingular, $n+1$ is reachable from any other vertex. This implies that $\sum_{i=k+1}^{n+1} Pr[k \overset{\le k}{\rightsquigarrow} i] = 1$ and that the expectation, denoted by E_k, of the number of times that one random walk in $\bigcup_{i=k+1}^{n+1} k \overset{\le k}{\rightsquigarrow} i$ passes k is finite. The following relationship to the exact LDL^T factorization of A has been established in [27],

$$\hat{d}_{k,k} = \frac{a_{k,k}}{E_k}, \quad \hat{l}_{i,k} = -Pr[k \overset{\le k}{\rightsquigarrow} i], \quad \forall 1 \le k < i \le n. \qquad (5)$$

Based on Eq. (5), one can obtain an approximated LDL^T factorization of A via Monte Carlo simulations. The following properties of $\hat{D}$ and $\hat{L}$ are preserved for D and L.

Observation 1. *Diagonal elements of D are positive, off-diagonal elements of L are nonpositive, and L is column-wise diagonally dominant.*

Moreover, when L and D are compared to the outcome of incomplete LDL^T factorization as described in Eq. (4), an immediate advantage is that there is no systematic difference since estimations of the probabilities and expectations could be either larger or smaller than their exact values, and there is no error accumulation.

However, the fundamental difficulty of stochastic random walk preconditioning is that the length of a random walk cannot be bounded, and thus the algorithm only terminates in a probabilistic sense even when one bounds the number of Monte Carlo simulations. For example, for the system shown in Fig. 1, a random walk starting from 4 could pass the edges $1 \to 2$ and $2 \to 1$ for unbounded number of times before finally reaching 5. In the extreme case when there are "traps" in the system [26], e.g. when the weights of the edges $1 \to 2$ and $2 \to 1$ are much larger compared to others, it will take excessive time to compute the preconditioner. In order to overcome such difficulty, one may partition the system [34] and use hierarchical random walks [26, 19] to generate a macromodel of the global grid before applying stochastic random walk preconditioning. However, since the macromodel essentially is an approximation of the Schur complement after partial factorization, to compute and to store it explicitly could be very costly.

3. PRECONDITIONING WITH DETERMINISTIC RANDOM WALKS

3.1 Structure of Random Walks

We propose to circumvent the difficulty in bounding the lengths of the random walks by computing their probabilities iteratively after decomposing them into shorter ones.

First of all, we show that the probabilities and expectations in Eq. (5) can be obtained from another set of random walks, which we believe to be more fundamental. Consider a random walk $r \in k \overset{\le k}{\rightsquigarrow} i$ for $1 \le k < i \le n+1$. If r passes k for more than once, i.e. $r \notin k \overset{<k}{\rightsquigarrow} i$, then we can decompose r into two walks r_1 and r_2 such that $r_1 \in k \overset{<k}{\rightsquigarrow} k$ and $r_2 \in k \overset{\le k}{\rightsquigarrow} i$. Therefore, $Pr[k \overset{\le k}{\rightsquigarrow} i] = Pr[k \overset{<k}{\rightsquigarrow} i] + Pr[k \overset{<k}{\rightsquigarrow} k]Pr[k \overset{\le k}{\rightsquigarrow} i]$. Since any vertex can reach $n+1$, we must have $\sum_{i=k}^{n+1} Pr[k \overset{<k}{\rightsquigarrow} i] = 1$ and $Pr[k \overset{<k}{\rightsquigarrow} k] < 1$. The following lemma holds.

Lemma 1. *$\forall 1 \le k < i \le n+1$,*

$$Pr[k \overset{\le k}{\rightsquigarrow} i] = \frac{Pr[k \overset{<k}{\rightsquigarrow} i]}{1 - Pr[k \overset{<k}{\rightsquigarrow} k]}.$$

On the other hand, recall that E_k is the expectation of the number of times that a random walk in $\bigcup_{i=k+1}^{n+1} k \overset{\le k}{\rightsquigarrow} i$ passes k. The aforementioned decomposition also implies that $E_k = 1 \cdot \sum_{i=k+1}^{n+1} Pr[k \overset{<k}{\rightsquigarrow} i] + (1+E_k)Pr[k \overset{<k}{\rightsquigarrow} k]$, which simplifies to Lemma 2.

Lemma 2. *$\forall 1 \le k \le n$,*

$$E_k = \frac{1}{1 - Pr[k \overset{<k}{\rightsquigarrow} k]}.$$

Lemma 1 and 2, together with Eq. (5), show that one can factor A by computing $Pr[k \overset{<k}{\rightsquigarrow} i]$ for all $1 \le k \le i \le n$, though further decompositions are necessary to eliminate Monte Carlo simulations.

Consider a set of independent random walks $\mathcal{N}_k$ such that all random walks in $\bigcup_{i=k}^{n+1} k \overset{<k}{\rightsquigarrow} i$ should start with. Let the vector $\mathbf{p}$ be the probability of the random walks in $\mathcal{N}_k$ that stop at a specific vertex, i.e. $p_i = Pr[k \overset{<k}{\rightsquigarrow} i \cap \mathcal{N}_k]$ for $1 \le i \le n$. A random walk $r \in k \overset{<k}{\rightsquigarrow} i - \mathcal{N}_k$ can be decomposed into two random walks $r_1 \in \mathcal{N}_k$ and r_2, where r_1 stops and r_2 starts at certain j. Then $j < i$ and $r_2 \in j \overset{<k}{\rightsquigarrow} i$. The walk r_2 can be further decomposed into m walks, one each from $j_1 \overset{\le j_1}{\rightsquigarrow} j_2$, $j_2 \overset{\le j_2}{\rightsquigarrow} j_3$, $\ldots$, $j_{m-1} \overset{\le j_{m-1}}{\rightsquigarrow} j_m$, and $j_m \overset{\le j_m}{\rightsquigarrow} i$, for certain $1 \le m < k$ and $j = j_1 < j_2 < \cdots < j_m < k$. For example, the random walk $4 \to 1 \to 2 \to 1 \to 3 \to 1 \to 3 \to 5$ in Fig. 1 is decomposed into 4 random walks $4 \to 1$, $1 \to 2$, $2 \to 1 \to 3$, and $3 \to 1 \to 3 \to 5$. Therefore,

$$\begin{aligned} Pr[k \overset{<k}{\rightsquigarrow} i] &= p_i + \sum_{m=1}^{k-1} \sum_{j_1 < j_2 < \cdots < j_m < k} Pr[j_m \overset{\le j_m}{\rightsquigarrow} i] \Big(\prod_{t=1}^{m-1} Pr[j_t \overset{\le j_t}{\rightsquigarrow} j_{t+1}] \Big) p_{j_1} \\ &= p_i + \sum_{m=1}^{k-1} (-\hat{L}_{i,1:k-1})(I - \hat{L}_{1:k-1,1:k-1})^m \mathbf{p}_{1:k-1}. \end{aligned}$$

Since $I - \hat{L}_{1:k-1,1:k-1}$ is a $k-1 \times k-1$ lower triangular matrix whose diagonal elements are all 0, we have,

Lemma 3. *$\forall 1 \le k \le i \le n$,*

$$Pr[k \overset{<k}{\rightsquigarrow} i] = p_i - \hat{L}_{i,1:k-1} \hat{L}^{-1}_{1:k-1,1:k-1} \mathbf{p}_{1:k-1}.$$

Clearly, if one choose a $\mathcal{N}_k$ such that the vector $\mathbf{p}$ can be computed without performing Monte Carlo simulations, then Lemma 1, 2, and 3 eliminate the need to simulate not only a random walk of unbounded number of steps but all of them. That motivates us to call our proposed technique *deterministic* random walk.

3.2 The Proposed Preconditioner

Based on Eq. (5) and Lemma 1, 2, and 3, we design the *deterministic random walk* (DRW) algorithm that computes an approximated LDL^T factorization of A. As shown in Fig. 2, the DRW algorithm depends on the choice of the random walk sets $\mathcal{N}_k$ and the function f that models the dropping and compensation scheme. Note that we do not explicitly initialize L and D since the elements that should be stored will all be computed in the algorithm.

```
Algorithm Deterministic Random Walk
Inputs
  $A$ :the system matrix.
  $\mathcal{N}_k$:sets of random walks for $1 \le k \le n$.
  $f$ :dropping and compensation scheme.
Outputs
  A diagonal matrix $D$ and a unit lower triangular
  matrix $L$.
1  For $k = 1$ to $n$:
2    Compute $\mathbf{p}$ from $\mathcal{N}_k$ and $A$.
3    For $i = k$ to $n$:
4      $q_i \leftarrow p_i - L_{i,1:k-1} L^{-1}_{1:k-1,1:k-1} \mathbf{p}_{1:k-1}$.
5    $d_{k,k} \leftarrow a_{k,k}(1 - q_k)$.
6    $L_{k+1:n,k} \leftarrow -f(q_k, q_{k+1}, \dots, q_n)$.
```

Figure 2: The DRW algorithm.

While it is difficult to quantitatively measure the differences between L and D generated by the DRW algorithm and their exact values, we should at least guarantee that the properties in Observation 1 are preserved for the correctness of the algorithm. We develop a set of conditions that various components of the DRW algorithm should satisfy as follows. First of all, we have proved the following lemma that states the properties of q_i's with some assumptions on the part of L that is already computed.

Lemma 4. *When reaching line 5 of the DRW algorithm, if $l_{i,j} \le 0$, $\forall (1 \le j < k) \wedge (j < i \le n)$ and $\sum_{i=j+1}^{n}(-l_{i,j}) \le 1$, $\forall 1 \le j < k$, then*

$$q_i \ge p_i \ge 0, \forall k \le i \le n, \quad \text{and} \quad \sum_{i=k}^{n} q_i \le \sum_{i=1}^{n} p_i \le 1.$$

The above lemma also implies that $q_k \le \sum_{i=1}^{n} p_i - \sum_{i=k+1}^{n} q_i \le \sum_{i=1}^{k} p_i$. Therefore, to guarantee $d_{k,k}$ on line 5 to be positive, it is sufficient to require $\sum_{i=1}^{k} p_i < 1$. Recall that the vertex $n+1$ is reachable from any other vertex in G_T^*. We can achieve so by reordering the rows and columns of A (and thus T) such that there exists an edge (k, i) for some i satisfying $k < i \le n+1$ in G_T^*. Since the columns of L are computed using f, the invariants of L as required by Lemma 4 can be maintained by choosing a proper f. Overall, the following theorem defines a set of sufficient conditions for the correctness of the DRW algorithm.

Theorem 1. *Assume that $\forall 1 \le k \le n$, there exists an edge (k, i) for some $k < i \le n+1$ in G_A^*. If f returns a vector whose elements are all nonnegative and the summation of them are no more than 1, then the DRW algorithm will not fail. Moreover, all diagonal elements of D are positive, L is column-wise diagonally dominant, and all its off-diagonal elements are nonpositive.*

It is worthwhile to mention here the **For** loop line 3 can be reorganized to facilitate sparse matrix operations. Let $\mathbf{q}$ be a vector where $\mathbf{q}_{1:k-1} = L^{-1}_{1:k-1,1:k-1}\mathbf{p}_{1:k-1}$ and $\mathbf{q}_{k:n}$ is computed as line 4. Then we have

$$\mathbf{q} = \begin{pmatrix} L_{1:k-1,1:k-1} & 0 \\ L_{k:n,1:k-1} & I \end{pmatrix}^{-1} \mathbf{p}. \tag{6}$$

In other words, $\mathbf{q}$ can be obtained from $\mathbf{p}$ by a partial forward substitution on L up to the first $k-1$ columns. Since it is reasonable to assume $\mathbf{p}$ to be sparse and L is also sparse, such computation can be done very efficiently without visiting all the $k-1$ columns leveraging the algorithm proposed in [9]. To be more specific, one should first perform a depth-first search in the extended matrix graph of $L_{1:k-1,1:k-1}$, starting from the vertices corresponding to the nonzero elements in $\mathbf{p}_{1:k-1}$ and stopping at k, to obtain a topological ordering of the reachable vertices, and then follow such ordering to use the columns of L.

Our DRW algorithm shares some similarities with incomplete LDL^T factorization, e.g. both compute a column of L using the columns that are already computed. We emphasize the two algorithm are quite different, even if the function f in the DRW algorithm is chosen to be the same as the dropping scheme in Eq. (3). Obviously, both Eq. (2) and (3) use not only $L_{1:n,1:k-1}$ but also $D_{1:k-1}$, though our DRW algorithm only depends on $L_{1:n,1:k-1}$. Such difference make our algorithm not sensitive to the error in the D matrix. On the other hand, Eq. (3) indicates that the column k of L is computed from the columns of L corresponding to the nonzero elements in $L_{k,1:k-1}$. Although such columns should be the same as the columns corresponding to the aforementioned set of reachable vertices in our algorithm when the factorizations are exact, they are different when elements are dropped from L. We observed that our algorithm will usually locate more columns than incomplete LDL^T factorization, which should appear in an exact factorization, resulting in a better approximation at the cost of more computation time. We believe that such cost is what one has to pay in order to achieve better quality via a more flexible dropping and compensation scheme.

3.3 Implementation Details

There are many ways to implement the various components of the DRW algorithm while still satisfying the requirements in Theorem 1. Here we present our choices when implementing the DRW algorithm for power grid analysis.

We reorder A to a Reverse Cuthill-McKee ordering (RCM) [5] before applying the DRW algorithm. This ordering is obtained by first starting a breadth-first search in G_A^* from the vertex $n+1$ treating all edges as undirected, and then rearranging the vertices in the reversed order that they are discovered. Clearly, since $n+1$ is reachable from any other vertices in G_A^*, we can guarantee that any vertex $k \ne n+1$ in the rearranged graph G_A^* has a neighbor $i > k$. We also observed from our experiments that such an ordering does achieve its original goal to benefit memory access patterns for sparse matrix vector product operations, which may justify its usage over variants of minimum degree ordering.

We choose $\mathcal{N}_k$ to be the set of all the one-step random walks from k. The $\mathbf{p}$ on line 2 of the DRW algorithm is computed directly from $A_{1:n,k}$. Note that we store the whole matrix A to make such computation simple instead of only storing half of A as A is symmetric. Other choices of $\mathcal{N}_k$ may improve the quality of the preconditioner, though we have not found it to be necessary since they will also cost more time to compute the preconditioner.

We specify the dropping and compensation scheme in two levels – globally and per column. Globally, two parameters should be provided: a filling factor γ specifying the desired ratio of the number of the nonzero off-diagonal elements in L to that in A, and a keep tolerance δ such that any element of L whose absolute value is larger should always be kept. We do not actually tune these parameters in our experiments. The parameter γ is used to constraint the size, and thus the memory usage, of the preconditioner. The parameter δ is

Table 1: Results Comparison for DC Analysis

name	statistics		DRW ($\gamma = 1$)		PowerRush [33]	
	n $\times 10^3$	nnz_A $\times 10^6$	t_{sol} (s)	Err. (uV)	t_{sol} (s)	Err. (uV)
ibmpg3	285	1.51	1.10	14/2	2.20	70/12
ibmpg4	953	4.05	2.92	5/1	2.94	230/29
ibmpg5	540	2.69	3.09	6/1	2.34	60/12
ibmpg6	834	4.13	5.25	6/1	5.92	90/12
ibmpgnew1	531	2.92	2.51	9/1	4.51	70/12
ibmpgnew2	531	2.92	2.53	10/2	4.49	70/12
total			17.41	50/8	22.40	590/90

always set to 0.05 as a safe guard in case the probabilities of the random walks are distributed evenly. Our experimental results are not sensitive to the value of δ unless it is too small, e.g. 0.001. Column-wise, we compute Γ as the number of nonzero off-diagonal elements for the column by assuming that the remaining nonzero off-diagonal elements should be distributed evenly across the remaining ones, or at least 2. For the function f, we first replace all but the largest Γ elements in $q_{k+1}, q_{k+2}, \ldots, q_n$ by 0. If any element being replaced is larger than δ, we will replace it back. Let the resulting elements be $q'_{k+1}, q'_{k+2}, \ldots, q'_n$. The function f will return the compensated vector as,

$$f(q_k, q_{k+1}, \ldots, q_n) = \frac{\sum_{i=k+1}^{n} q_i}{\sum_{i=k+1}^{n} q'_i}\left(\frac{q'_{k+1}}{1-q_k}, \frac{q'_{k+2}}{1-q_k}, \ldots, \frac{q'_n}{1-q_k}\right),$$

i.e. the dropped elements are added to the remaining ones proportionally. Note that if all the inputs are 0, f will simply return 0. Overall, our dropping scheme is a direct extension of that in [15], while the compensation scheme, though intuitive, cannot be applied there directly.

4. EXPERIMENTAL RESULTS

We implement our DRW algorithm with a PCG solver in C++ to solve power grid analysis problems. For comparison, we implement in our solver the incomplete LDL^T factorization as described in Section 2.2 using the data structure proposed by [15]. Moreover, the binary library for stochastic random walk preconditioning [27] and the source code of the direct solver CHOLMOD [3] are also integrated to our solver. Our solver is built by GCC version 4.3 and we run all the experiments on a 64-bit Linux workstation with a 2.4GHz Intel Q6600 processor and 8GB memory. Note that only one core is used for all the experiments.

We first validate our proposed DRW preconditioner and our PCG solver using public IBM power grid benchmarks for DC analysis [25, 20]. We set γ to 1 to reduce the time to compute our preconditioner and terminate the PCG iterations when the l_2 norm of the residue is reduced to 10^{-3} of its original value. The results of the largest 6 circuits are reported in Table 1 and are compared against the state-of-the-art AMG-PCG solver PowerRush [33]. Similar to [33], we preprocess the SPICE netlist to treat resistances less than $10^{-6}\Omega$ as shorts. In Table 1, we first show the dimension of A ("n") and the number of nonzero elements in A ("nnz_A"). Then, for each solver, we show the total solving time in seconds ("t_{sol}") and the maximum/average errors to the golden solution in uV ("*Err.*"). For our PCG solver with DRW preconditioning, the column "t_{sol}" includes the time to reorder A, to compute the preconditioner, to perform PCG iterations, and to compute the maximum/average errors. For PowerRush, the two columns are obtained from Table 1 in [33] and we compensate for the different processors used for the experiments by scaling down the total solving time using a factor of $\frac{2.13}{2.40}$. Although different processor microarchitectures may also affect the solving times, we believe that the comparison is reasonably fair so that we can conclude our PCG solver with DRW preconditioning can generate results with better accuracy in less time than PowerRush.

The memory usage of our DRW preconditioner can be calculated from the numbers of nonzero elements in L, denoted by nnz_L. For 4-byte integer indices and 8-byte floating point values, to store both L and D requires $12nnz_L$ bytes of memory. We observed that nnz_L faithfully follow the setting $\gamma = 1$ for the experiments shown in Table 1, i.e. they are almost the same as the values in the column "nnz_A", and thus the maximum memory usage for our DRW preconditioner is less than 50MB for all benchmarks.

We then compare our DRW preconditioner to three other preconditioners and report the results in Table 2. The first one is stochastic random walk [27]. We only use their library to generate the preconditioners since we found their solver to be always slower than ours. The second is the incomplete LDL^T factorization implemented by us. We use the same matrix ordering and the same dropping scheme as our DRW algorithm without compensation. The third one is obtained from our DRW preconditioner by removing the compensation scheme, and thus named DRW-NC (No Compensation). The parameter γ for incomplete LDL^T factorization, DRW-NC, and DRW are all set to 1.7 to match the total number of nonzero elements in L across all benchmarks as generated by stochastic random walk preconditioning. The PCG solver solves the same 100 random vectors for all the preconditioners and is terminated when the l_2 norm of the residue is reduced to 10^{-6} of its original value. For each preconditioner, we report the number of nonzero elements in L ("nnz_L"), the time to compute the preconditioner ("t_{pre}"), the average time to solve the problem in the PCG solver ("t_{pcg}"), and the average number of PCG iterations ("#it."). It can be seen that the maximum memory usage for our DRW preconditioner is less than 80MB.

The results in Table 2 indicate that with a proper choice of the dropping scheme, incomplete LDL^T factorization can be quite effective to precondition power grid analysis problems. Our DRW preconditioner is able to achieve an additional 1.7× reduction in both the number of PCG iterations and the time to solve the problem, mostly due to the compensation scheme; otherwise, the preconditioners generated by DRW-NC are marginally better than those generated by incomplete LDL^T factorization. On the other hand, for stochastic random walk preconditioning, although the number of PCG iterations is 20% less than our DRW preconditioner, it takes much more time to actually solve the problems in a PCG solver. We found that this is due to the matrix ordering used internally by the stochastic random walk code – compared to the RCM ordering used by the other three preconditioners, this ordering has incurred considerable amount of overhead when performing the sparse matrix vector product operations in the PCG solver. Overall, compared to stochastic random walk preconditioning, our proposed DRW preconditioner requires 3.9× less time to compute and reduces the time to solve the problem by 2.4×.

Finally, we study the impact of memory usage to solver efficiency among a few solvers and report the results in Table 3

Table 2: Results Comparison for Different Preconditioners

name	Stochastic Random Walk				Incomplete LDL^T ($\gamma=1.7$)				DRW-NC ($\gamma=1.7$)				DRW ($\gamma=1.7$)			
	nnz_L $\times 10^6$	t_{pre} (s)	t_{pcg} (s)	#it.	nnz_L $\times 10^6$	t_{pre} (s)	t_{pcg} (s)	#it.	nnz_L $\times 10^6$	t_{pre} (s)	t_{pcg} (s)	#it.	nnz_L $\times 10^6$	t_{pre} (s)	t_{pcg} (s)	#it.
ibmpg3	2.22	10.45	1.27	18	2.36	0.37	0.75	25	2.36	1.23	0.70	23	2.37	1.38	0.53	17
ibmpg4	6.92	26.41	5.93	20	6.22	0.92	4.48	51	6.22	20.07	4.27	49	6.22	22.21	1.84	21
ibmpg5	4.15	25.71	3.04	21	4.19	0.56	2.49	52	4.19	2.54	2.44	49	4.20	2.68	1.70	34
ibmpg6	6.29	43.85	5.95	22	6.41	0.88	4.81	61	6.41	2.47	4.64	57	6.41	2.62	3.40	42
ibmpgnew1	4.45	18.11	3.70	19	4.59	0.74	2.05	29	4.59	3.36	1.91	27	4.59	3.79	1.22	17
ibmpgnew2	4.44	18.21	3.58	20	4.59	0.76	2.05	29	4.59	3.38	1.91	27	4.59	3.78	1.22	17
total	28.47	142.73	23.47	119	28.36	4.22	16.63	246	28.36	33.05	15.87	233	28.37	36.47	9.91	149
ratio		3.91	2.37	0.80		0.12	1.68	1.66		0.91	1.60	1.57		1.00	1.00	1.00

Table 3: Results Comparison for Different Solvers

solver	Total of All 6 Circuits			
	nnz_L $\times 10^6$	t_{pre} (s)	t_{pcg} (s)	#it.
Stochastic Random Walk *quality* = 0.6	24.8	84.47	28.17	161
Incomplete LDL^T ($\gamma=1.0$)	18.2	2.44	23.75	428
Stochastic Random Walk	28.5	142.73	23.47	119
Incomplete LDL^T ($\gamma=1.7$)	28.4	4.22	16.63	246
DRW ($\gamma=1.0$)	18.2	5.85	12.77	229
DRW ($\gamma=1.7$)	28.4	36.47	9.91	149
CHOLMOD [3]	221.8	61.50	0.87	–

with the rows sorted according to the time to solve the problem. Three of the solvers are based on the preconditioners in Table 2 excluding DRW-NC. Another three solvers are obtained by reducing the memory usage of the preconditioners. For DRW and incomplete LDL^T factorization, we reduce γ to 1.0. For stochastic random walk preconditioning, we set the parameter *quality* to 0.6 as recommended by the authors of [27]. Results of the direct solver CHOLMOD [3] are also reported, though the three columns should be interpreted as the number of nonzero elements in the exact Cholesky factor, the time to compute the factor, and the time to solve the problem directly, i.e. to perform a pair of forward/back substitutions. For Table 3, we can see that CHOLMOD is the best choice when there is ample memory and when the factor can be reused to solve many problems, e.g. for transient analysis. However, if the exact factors cannot be held in memory due to large problem sizes, or if the system is only solved for very few times, our proposed DRW preconditioning technique together with a PCG solver will be the best – even with a memory usage similar to that of the original system, the preconditioners generated by our DRW algorithm lead to less PCG iterations and less solving times than incomplete factorizations with much more fill-ins, and are thus suitable for both DC and transient analysis'.

5. CONCLUSIONS AND FUTURE WORKS

In this paper, we presented the deterministic random walk preconditioning technique for power grid analysis via preconditioned conjugate gradient solvers. Compared to stochastic random walk preconditioning, our proposed algorithm computed the preconditioners in a deterministic manner to reduce computation time. Compared to incomplete factorizations, our proposed algorithm leveraged a compensation scheme to improve preconditioner quality while maintaining correctness. Solvers built on top of our proposed preconditioner were able to outperform other state-of-the-art iterative solvers for DC and transient power grid analysis'.

While this paper focuses on symmetric matrices, our proposed DRW algorithm could be used to compute approximated LDU^T factorizations for asymmetric matrices directly or following the same idea to handle them in [27]. The requirement of nonpositive off-diagonal elements could also be removed via a transformation described in [11]. Further evaluations are required to extend our work to such cases.

6. REFERENCES

[1] H. Chen and D. Ling. Power supply noise analysis methodology for deep-submicron VLSI chip design. In *DAC*, pages 638–643, 1997.

[2] T.-H. Chen and C. C.-P. Chen. Efficient large-scale power grid analysis based on preconditioned Krylov-subspace iterative methods. In *DAC*, pages 559–562, 2001.

[3] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software*, 35(3):22:1–22:14, Oct. 2008.

[4] C.-H. Chou, N.-Y. Tsai, H. Yu, C.-R. Lee, Y. Shi, and S.-C. Chang. On the preconditioner of conjugate gradient method – a power grid simulation perspective. In *ICCAD*, pages 494–497, 2011.

[5] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *ACM National Conference*, pages 157–172, 1969.

[6] A. Dharchoudhury, R. Panda, D. Blaauw, R. Vaidyanathan, B. Tutuianu, and D. Bearden. Design and analysis of power distribution networks in PowerPC microprocessors. In *DAC*, pages 738–743, 1998.

[7] Z. Feng and P. Li. Multigrid on GPU: Tackling power grid analysis on parallel SIMT platforms. In *ICCAD*, pages 647–654, 2008.

[8] Z. Feng, X. Zhao, and Z. Zeng. Robust parallel preconditioned power grid simulation on GPU with adaptive runtime performance modeling and optimization. *IEEE TCAD*, 30(4):562–573, Apr. 2011.

[9] J. R. Gilbert and T. Peierls. Sparse partial pivoting in time proportional to arithmetic operations. *SIAM Journal on Scientific and Statistical Computing*, 9(5):862–874, Sept. 1988.

[10] G. H. Golub and C. F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 3rd edition, 1996.

[11] K. D. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear*

Systems. PhD thesis, Carnegie Mellon University, 1996.

[12] I. Gustafsson. A class of first order factorization methods. *BIT*, 18(2):142–156, June 1978.

[13] X. Hu, W. Zhao, P. Du, A. Shayan, and C.-K. Cheng. An adaptive parallel flow for power distribution network simulation using discrete fourier transform. In *ASPDAC*, pages 125–130, 2010.

[14] Y.-M. Jiang and K.-T. Cheng. Analysis of performance impact caused by power supply noise in deep submicron devices. In *DAC*, pages 760–765, 1999.

[15] M. T. Jones and P. E. Plassmann. An improved incomplete Cholesky factorization. *ACM Transactions on Mathematical Software*, 21(1):5–17, Mar. 1995.

[16] J. Kozhaya, S. Nassif, and F. Najm. A multigrid-like technique for power grid analysis. *IEEE TCAD*, 21(10):1148–1160, Oct. 2002.

[17] Y.-M. Lee, Y. Cao, T.-H. Chen, J. Wang, and C.-P. Chen. HiPRIME: hierarchical and passivity preserved interconnect macromodeling engine for RLKC power delivery. *IEEE TCAD*, 24(6):797–806, June 2005.

[18] D. Li, S.-D. Tan, and B. McGaughy. ETBR: Extended truncated balanced realization method for on-chip power grid network analysis. In *DATE*, pages 432–437, 2008.

[19] P. Li. Power grid simulation via efficient sampling-based sensitivity analysis and hierarchical symbolic relaxation. In *DAC*, pages 664–669, 2005.

[20] Z. Li, F. Liu, R. Balasubramanian, and S. Nassif. TAU 2011 power grid analysis contest. In *TAU Workshop*, 2011.

[21] Z. Li, F. Liu, R. Balasubramanian, and S. Nassif. TAU 2012 power grid simulation contest. In *TAU Workshop*, 2012.

[22] T. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34(150):473–497, Apr. 1980.

[23] J. Meijerink and H. van der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M–matrix. *Mathematics of Computation*, 31(137):148–162, Jan. 1977.

[24] S. Nassif and J. Kozhaya. Fast power grid simulation. In *DAC*, pages 156–161, 2000.

[25] S. R. Nassif. Power grid analysis benchmarks. In *ASPDAC*, pages 376–381, 2008.

[26] H. Qian, S. Nassif, and S. Sapatnekar. Power grid analysis using random walks. *IEEE TCAD*, 24(8):1204–1224, Aug. 2005.

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

[28] J. Shi, Y. Cai, W. Hou, L. Ma, S.-D. Tan, P.-H. Ho, and X. Wang. GPU friendly fast poisson solver for structured power grid network analysis. In *DAC*, pages 178–183, 2009.

[29] G. Steele, D. Overhauser, S. Rochel, and S. Z. Hussain. Full-chip verification methods for DSM power distribution systems. In *DAC*, pages 744–749, 1998.

[30] H. Su, E. Acar, and S. Nassif. Power grid reduction based on algebraic multigrid principles. In *DAC*, pages 109–112, 2003.

[31] J. Wang and T. Nguyen. Extended Krylov subspace method for reduced order analysis of linear circuits with multiple sources. In *DAC*, pages 247–252, 2000.

[32] J. Yang, Y. Cai, Q. Zhou, and J. Shi. Fast poisson solver preconditioned method for robust power grid analysis. In *ICCAD*, pages 531–536, 2011.

[33] J. Yang, Z. Li, Y. Cai, and Q. Zhou. PowerRush: A linear simulator for power grid. In *ICCAD*, pages 482–487, 2011.

[34] M. Zhao, R. Panda, S. Sapatnekar, and D. Blaauw. Hierarchical analysis of power distribution networks. *IEEE TCAD*, 21(2):159–168, Feb. 2002.

[35] S. Zhao, K. Roy, and C.-K. Koh. Frequency domain analysis of switching noise on power supply network. In *ICCAD*, pages 487–492, 2000.

[36] X. Zhao, J. Wang, Z. Feng, and S. Hu. Power grid analysis with hierarchical support graphs. In *ICCAD*, pages 543–547, 2011.

[37] Y. Zhong and M. Wong. Fast algorithms for IR drop analysis in large power grid. In *ICCAD*, pages 351–357, 2005.

[38] Z. Zhu, B. Yao, and C.-K. Cheng. Power network analysis using an adaptive algebraic multigrid approach. In *DAC*, pages 105–108, 2003.

[39] C. Zhuo, J. Hu, M. Zhao, and K. Chen. Power grid analysis and optimization using algebraic multigrid. *IEEE TCAD*, 27(4):738–751, Apr. 2008.