STOCHASTIC PRECONDITIONING FOR DIAGONALLY DOMINANT MATRICES*

HAIFENG QIAN† AND SACHIN S. SAPATNEKAR‡

Abstract. This paper presents a new stochastic preconditioning approach for large sparse matrices. For the class of matrices that are row-wise and column-wise irreducibly diagonally dominant, we prove that an incomplete $\mathrm{LDL}^{\mathrm{T}}$ factorization in a symmetric case or an incomplete LDU factorization in an asymmetric case can be obtained from random walks, and used as a preconditioner. It is argued that our factor matrices have better quality, i.e., better accuracy-size tradeoffs, than preconditioners produced by existing incomplete factorization methods. Therefore a resulting preconditioned Krylov-subspace iterative solver requires less computation than traditional methods to solve a set of linear equations with the same error tolerance. The advantage increases for larger and denser matrices. These claims are verified by numerical tests, and we provide techniques that can potentially extend the theory to non-diagonally-dominant matrices.

Key words. incomplete factorization, iterative solver, preconditioning, random walk

AMS subject classifications. 15A12, 15A23, 60G50, 65C05, 65F10

1. Introduction.

Preconditioning is a crucial part of an iterative solver. Suppose a set of linear equations is $A\mathbf{x} = \mathbf{b}$, where A is a given square nonsingular matrix that is large and sparse, $\mathbf{b}$ is a given vector, and $\mathbf{x}$ is the unknown solution vector to be computed. A preconditioner is a square nonsingular matrix T such that an iterative solver can solve the transformed system $TA\mathbf{x} = T\mathbf{b}$ with a higher convergence rate.

The quality of a preconditioner matrix T is how closely it approximates A^{-1}, measured by the condition number of TA or the spectral radius of $(I - TA)$. Existing preconditioning techniques can be roughly divided into two categories: explicit methods and implicit methods [3]. In explicit methods, which are often referred to as approximate inverse methods, the preconditioner T is in the form of a matrix or a polynomial of matrices [3], [4], [28]. In implicit methods, the preconditioner T is in the form of $(A')^{-1}$, where A' approximates A and is easier to solve [3], [28]. Although explicit preconditioning methods have the advantage of being easily parallelizable, implicit methods have been more successfully developed and more widely used. A prominent class of implicit preconditioners are those based on incomplete LU (ILU) factorization: for example, ILU(0), ILU(k) and ILUT are popular choices in numerical computation [2], [6], [28]. Another aspect of the quality of an ILU preconditioner is its stability, i.e., the condition numbers of the ILU factors, especially for indefinite matrices [7]; for diagonally dominant matrices, this is less an issue since most ILU methods, including ours, guarantee that the ILU factors are also diagonally dominant.

The subject of this paper, except for Section 8, is the class of matrices that are row-wise and column-wise irreducibly diagonally dominant, defined in Definition 1.2.

DEFINITION 1.1. *The matrix graph of a square matrix A of dimension N is a directed graph with N nodes labeled $1, 2, \cdots, N$, such that, for any $i \neq j$, an edge exists from node i to node j if and only if $A_{i,j} \neq 0$.*

DEFINITION 1.2. *A square matrix A is said to be row-wise and column-wise irreducibly diagonally dominant if it satisfies the following conditions for any i:*

*This research was supported in part by the NSF under awards CCF-0634802 and CCF-0205227, and by an IBM fellowship.

†IBM T. J. Watson Research Center, Yorktown Heights, New York (qianhaifeng@us.ibm.com).

‡Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, Minnesota (sachin@ece.umn.edu).

1. $A_{i,i} \geq \sum_{j \neq i} |A_{i,j}|$.
2. $A_{i,i} \geq \sum_{j \neq i} |A_{j,i}|$.
3. $\exists k$ *such that* $A_{k,k} > \sum_{j \neq k} |A_{k,j}|$ *and that there exists a directed path from node* i *to node* k *in the matrix graph of* A.
4. $\exists k$ *such that* $A_{k,k} > \sum_{j \neq k} |A_{j,k}|$ *and that there exists a directed path from node* k *to node* i *in the matrix graph of* A.

It is worth noting that: if row i is strictly diagonally dominant, the third condition is trivially satisfied; if column i is strictly diagonally dominant, the fourth condition is trivially satisfied; all diagonal entries are positive; A is positive definite. If A is symmetric, its matrix graph can be represented by an undirected graph, and then the first and second conditions merge into one, and the third and fourth merge as well.

For a matrix A that satisfies Definition 1.2, the most widely used preconditioners are the variants of incomplete triangular factorization [2], [19], [28]. If A is asymmetric, an ILU preconditioner is in the form of $T = \left(\tilde{L}\tilde{D}\tilde{U}\right)^{-1}$ where $\tilde{L}$ is a lower triangular matrix with unit diagonals, $\tilde{D}$ is a diagonal matrix, and $\tilde{U}$ is an upper triangular matrix with unit diagonals. To produce such a preconditioner, various existing techniques all perform Gaussian elimination on A, and each uses a specific strategy to drop insignificant entries during the process: ILU(0) applies a pattern-based strategy, and allows $\tilde{L}_{i,j} \neq 0$ or $\tilde{U}_{i,j} \neq 0$ only if $A_{i,j} \neq 0$ [28]; ILUT applies a value-based strategy, and drops an entry from $\tilde{L}$ or $\tilde{U}$ if its value is below a threshold [28]; a more advanced strategy can be a combination of pattern, threshold and other size limits such as maximum number of entries per row. Another ILU strategy, referred to as Modified ILU (MILU), is to compensate the diagonal entries of $\tilde{D}$ during the factorization process to guarantee that the row sums of $\tilde{L}\tilde{D}\tilde{U}$ are equal to those of A [28]. Combining the MILU strategy with the previously mentioned dropping strategies results in MILU(0), MILUT, and so on. If A is symmetric, the ILU becomes the incomplete Cholesky (IC) preconditioner in the form of $T = \left(\tilde{L}\tilde{D}\tilde{L}^{\mathrm{T}}\right)^{-1}$, where $\tilde{L}$ is a lower triangular matrix with unit diagonals and $\tilde{D}$ is a diagonal matrix. The various ILU strategies all apply in their symmetric forms, and result in the preconditioners IC(0), ICT, MIC(0), MICT, and so on. For symmetric M-matrices, another approach is the support-graph method, which also produces a preconditioner in the form of $T = \left(\tilde{L}\tilde{D}\tilde{L}^{\mathrm{T}}\right)^{-1}$, but with the property that $\tilde{L}\tilde{D}\tilde{L}^{\mathrm{T}}$ corresponds to a subgraph of the matrix graph of A; the main difference between (non-recursive) support-graph and IC is that entries are dropped from the original matrix A initially, rather than from the partially factored matrix during the factorization process [5], [18], [30].

The proposed preconditioning technique in this paper belongs to the category of implicit preconditioners based on incomplete factorization, and our innovation is a stochastic procedure for building the incomplete triangular factors. It is argued algorithmically that our factor matrices have better quality, i.e., better accuracy-size tradeoffs, than preconditioners produced by existing incomplete factorization methods. Therefore the resulting preconditioned Krylov-subspace solver, which we refer to as the *hybrid solver*, requires less computation than traditional methods to solve a set of linear equations with the same error tolerance, and the advantage increases for larger and denser matrices. We use numerical tests to compare our method against IC(0), ICT, MICT, and support-graph preconditioners on symmetric diagonally dominant benchmarks, and compare against ILU(0), ILUT, and MILUT on asymmetric diagonally dominant benchmarks. We also provide in Section 5.3 the relation to ex-

plicit factored-approximate-inverse preconditioners, and in Section 8 techniques that can potentially extend the theory to non-diagonally-dominant matrices.

Parts of this paper were initially published in [25] and [26], which dealt with only symmetric diagonally dominant M-matrices in specific engineering problems. This manuscript includes unpublished mathematical proofs in Section 3, unpublished Sections 4 and 8 on the generalization of the theory, unpublished techniques in Section 6 to improve the performance, as well as comprehensive numerical results.

For clarity of the presentation, Sections 2 and 3 will describe the main framework of our theory in the context of a narrower class of matrices: if a matrix satisfies Definition 1.2 and is a symmetric M-matrix, it is referred to as an *R-matrix* in this paper as a shorthand notion. Then in Section 4, we will show that any matrix A that satisfies Definition 1.2 can be handled with two extra techniques.

We will now review previous efforts of using stochastic methods to solve systems of linear equations. Historically, the theory was developed on two seemingly independent tracks, related to the analysis of potential theory [8], [15], [17], [20], [21], [23] and to the solution of systems of linear equations [11], [15], [31], [32], [34]. However, the two applications are closely related and research along each of these tracks has resulted in the development of analogous algorithms, some of which are equivalent.

The second track will be discussed here. The first work of a random-walk based linear equation solver is [11], although it was presented as a solitaire game of drawing balls from urns. It was proven in [11] that, if the matrix A satisfies certain conditions, a game can be constructed and a random variable[1] X can be defined such that its expected value $E[X] = (A^{-1})_{ij}$, where $(A^{-1})_{ij}$ is an entry of the inverse matrix of A. Two years later, the work in [34] continued this discussion in the formulation of random walks, and proposed the use of another random variable, and it was argued that, in certain special cases, this variable has a lower variance than X, and hence is likely to converge faster. Both [11] and [34] have the advantage of being able to compute part of an inverse matrix without solving the whole system, in other words, localizing computation. Over the years, various descendant stochastic solvers have been developed [15], [31], [32], though some of them do not have the locality property.

Early stochastic solvers suffer from accuracy limitations, and this was remedied by the sequential Monte Carlo method proposed in [14] and [22], through iterative refinement. Let $\mathbf{x}'$ be an approximate solution to $A\mathbf{x} = \mathbf{b}$ found by a stochastic solver, let the residual vector be $\mathbf{r} = \mathbf{b} - A\mathbf{x}'$, and let the error vector be $\mathbf{z} = \mathbf{x} - \mathbf{x}'$; then $A\mathbf{z} = \mathbf{r}$. The idea of the sequential Monte Carlo method is to iteratively solve $A\mathbf{z} = \mathbf{r}$ using a stochastic solver, and in each iteration, to compute an approximate $\mathbf{z}$ that is then used to correct the current solution $\mathbf{x}'$. Although the sequential Monte Carlo method has existed for over forty years, it has not resulted in any powerful solver that can compete with direct and iterative solvers, due to the fact that random walks are needed in every iteration, resulting in a relatively high overall time complexity.

2. Stochastic Linear Equation Solver. In this section, we study the underlying stochastic mechanism of the proposed preconditioner. It is presented as a stand-alone stochastic linear equation solver; however, in later sections, its usage is not to solve equations, but to build an incomplete factorization.

2.1. The Generic Algorithm. Let us consider a random walk "game" defined on a finite undirected connected graph representing a street map, for example, Figure 2.1. A walker starts from one of the nodes, and every day, he/she goes to an

[1] The notations are different from the original ones used in [11].

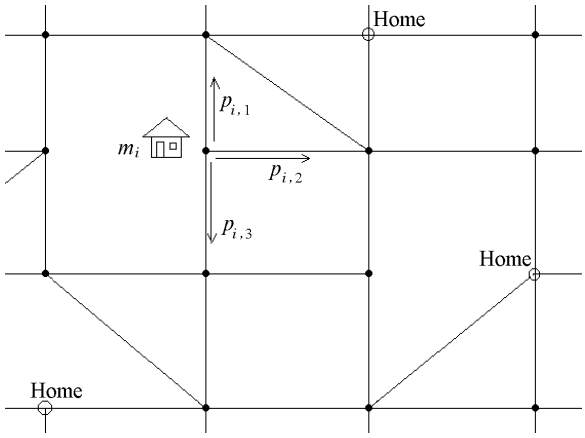

FIG. 2.1. *An instance of a random walk "game."*

adjacent node l with probability $p_{i,l}$ for $l = 1, 2, \cdots, \text{degree}(i)$, where i is the current node, degree(i) is the number of edges connected to node i, and the adjacent nodes are labeled $1, 2, \cdots, \text{degree}(i)$. The transition probabilities satisfy

$$\sum_{l=1}^{\text{degree}(i)} p_{i,l} = 1. \tag{2.1}$$

The walker pays an amount m_i to a motel for lodging everyday, until he/she reaches one of the homes, which are a subset of the nodes. Note that the motel price m_i is a function of his/her current location, node i. The game ends when the walker reaches a home node: he/she stays there and gets awarded a certain amount of money, m_0. We now consider the problem of calculating the expected amount of money that the walker has accumulated at the end of the walk, as a function of the starting node, assuming he/she starts with nothing. The gain function is therefore defined as

$$f(i) = E[\text{total money earned } | \text{walk starts at node } i]. \tag{2.2}$$

It is obvious that

$$f(\text{one of the homes}) = m_0. \tag{2.3}$$

For a non-home node i, again assuming that the nodes adjacent to i are labeled 1, 2, $\cdots, \text{degree}(i)$, the f variables satisfy

$$f(i) = \sum_{l=1}^{\text{degree}(i)} p_{i,l} f(l) - m_i. \tag{2.4}$$

For a game with N non-home nodes, there are N linear equations similar to the one above, and the solution to this set of equations gives the exact values of f at all nodes.

In the above equations obtained from a random walk game, the set of allowable matrices is a superset of the set of R-matrices[2]. In other words, given a set of linear

[2]A matrix from a game has all the properties of an R-matrix, except that it may be asymmetric.

equations $A\mathbf{x} = \mathbf{b}$, where A is an R-matrix, we can always construct a random walk game that is mathematically equivalent, i.e., such that the f values are the desired solution $\mathbf{x}$. To do so, we divide the i^{th} equation by $A_{i,i}$ to obtain

$$x_i + \sum_{j \neq i} \frac{A_{i,j}}{A_{i,i}} x_j = \frac{b_i}{A_{i,i}}, \tag{2.5}$$

$$x_i = \sum_{j \neq i} \left(-\frac{A_{i,j}}{A_{i,i}} \right) x_j + \frac{b_i}{A_{i,i}}. \tag{2.6}$$

Equation (2.4) and equation (2.6) have seemingly parallel structures. Let N be the dimension of the matrix A, and let us construct a game with N non-home nodes, which are labeled $1, 2, \cdots, N$. Due to the properties of an R-matrix, we have

- $\left(-\frac{A_{i,j}}{A_{i,i}} \right)$ is a non-negative value and can be interpreted as the transition probability of going from node i to node j.
- $\left(-\frac{b_i}{A_{i,i}} \right)$ can be interpreted as the motel price m_i at node i.

However, the above mapping is insufficient due to the fact that condition (2.1) may be broken: the sum of the $\left(-\frac{A_{i,j}}{A_{i,i}} \right)$ coefficients is not necessarily one. In fact, because all rows of the matrix A are diagonally dominant, the sum of the $\left(-\frac{A_{i,j}}{A_{i,i}} \right)$ coefficients is always less than or equal to one. Condition (2.1) can be satisfied if we add an extra transition probability of going from node i to a home node, by rewriting (2.6) as

$$x_i = \sum_{j \neq i} \left(-\frac{A_{i,j}}{A_{i,i}} \right) x_j + \frac{\sum_j A_{i,j}}{A_{i,i}} \cdot m_0 + \frac{b'_i}{A_{i,i}}, \text{ where } b'_i = b_i - \sum_j A_{i,j} \cdot m_0. \tag{2.7}$$

It is easy to verify that $\frac{\sum_j A_{i,j}}{A_{i,i}}$ is a non-negative value for an R-matrix, and that the following mapping establishes the equivalence between equation (2.4) and equation (2.7), while satisfying (2.1) and (2.3).

- $\left(-\frac{A_{i,j}}{A_{i,i}} \right)$ is the probability of going from node i to node j.
- $\frac{\sum_j A_{i,j}}{A_{i,i}}$ is the probability of going from node i to a home with award m_0.
- $\left(-\frac{b'_i}{A_{i,i}} \right)$ is the motel price m_i at node i.

The choice of m_0 is arbitrary because b'_i always compensates for the m_0 term in equation (2.7), and in fact m_0 can take different values in (2.7) for different rows i. Therefore the mapping from an equation set to a game is not unique. A simple scheme can be to let $m_0 = 0$, and then $m_i = -\frac{b_i}{A_{i,i}}$.

To find x_i, the i^{th} entry of solution vector $\mathbf{x}$, a natural way is to simulate a certain number of random walks from node i and use the average monetary gain in these walks as the approximated entry value. If this amount is averaged over a sufficiently large number of walks by playing the "game" a sufficiently large number of times, then by the Law of Large Numbers [35], an acceptably accurate solution can be obtained.

According to the Central Limit Theorem [35], the estimation error of the above procedure is asymptotically a zero-mean Gaussian variable with variance inversely proportional to M, where M is the number of walks. Thus there is an accuracy-runtime tradeoff. In implementation, instead of fixing M, one may employ a stopping

criterion driven by a user-specified error margin Δ and confidence level α:

$$P[-\Delta < x_i' - x_i < \Delta] > \alpha, \tag{2.8}$$

where x_i' is the estimated i^{th} solution entry from M walks.

2.2. Two Speedup Techniques. In this section, we propose two new techniques that dramatically improve the performance of the stochastic solver. They will play a crucial role in the proposed preconditioning technique.

2.2.1. Creating Homes. As discussed in the previous section, a single entry in the solution vector $\mathbf{x}$ can be evaluated by running random walks from its corresponding node in the game. To find the complete solution $\mathbf{x}$, a straightforward way is to repeat such procedure for every entry. This, however, is not the most efficient approach.

We propose a speedup technique by adding the following rule: after the computation of x_i is finished according to criterion (2.8), node i becomes a new home node in the game with an award amount equal to the estimated value x_i'. In other words, any later random walk that reaches node i terminates, and is rewarded a money amount equal to the assigned x_i'. Without loss of generality, suppose the nodes are processed in the natural ordering $1, 2, \cdots, N$, then for walks starting from node k, the node set $\{1, 2, \cdots, k-1\}$ are homes where the walks terminate (in addition to the original homes generated from the strictly-diagonally-dominant rows of A), while the node set $\{k, k+1, \cdots, N\}$ are motels where the walks pass by.

One way to interpret this technique is by the following observation about (2.4): there is no distinction between the neighboring nodes that are homes and those that are motels, and the only reason that a walk can terminate at a home node is that its f value is known and is equal to the award. In fact, any node can be converted to a home node if we know its f value and assign the award accordingly. Our new rule is simply utilizing the estimated $x_i' \approx x_i$ in such a conversion. Another way to interpret this technique is by looking at the source of the value x_i'. Each walk that ends at a new home and obtains such an award is equivalent to an average of multiple walks, each of which continues walking from there according to the original game settings.

With this new method, as the computation for the complete solution $\mathbf{x}$ proceeds, more and more new home nodes are created in the game. This speeds up the algorithm dramatically, as walks from later nodes are carried out in a game with a larger and larger number of homes, and the average number of steps in each walk is reduced. At the same time, this method helps convergence without increasing M, because, as mentioned earlier, each walk becomes the average of multiple walks. The only cost[3] is that the game becomes slightly biased when a new home node is created, due to the fact that the assigned award value is only an estimate, e.g. $x_i' \neq x_i$; overall, the benefit of this technique dominates its cost.

2.2.2. Bookkeeping. Direct solvers are efficient in computing solutions for multiple right-hand-side vectors after an initial matrix factorization, since only a forward/backward substitution step is required for each additional solve. Analogous to a direct solver, we propose a speedup mechanism for the stochastic solver.

Recall that in the procedure of constructing a random walk game discussed in Section 2.1, the topology of the game and the transition probabilities are solely determined by the matrix A, and hence do not change when the right-hand-side vector $\mathbf{b}$ changes. Only motel prices and award values in the game are linked to $\mathbf{b}$.

[3]The cost discussed here is in the context of the stochastic solver only. For the proposed preconditioner, this will no longer be an issue.

When solving a set of linear equations with the matrix A for the first time, we create a journey record for every node in the game, listing the following information.

- For any node i, record the number of walks performed from node i.
- For any node i and any visited motel node j, record the number of times that walks from node i visit node j.
- For any node i and any reached home node j, which can be either an initial home in the original game or a new home node created by the technique from Section 2.2.1, record the number of walks that start from i and end at j.

Then, if the right-hand-side vector $\mathbf{b}$ changes while the matrix A remains the same, we do not need to perform random walks again. Instead, we simply use the journey record repeatedly and assume that the walker takes the same routes, gets awards at the same locations, pays for the same motels, and only the award amounts and motel prices have been modified. Thus, after a journey record is created, new solutions can be computed by some multiplications and additions efficiently.

Practically, this bookkeeping is only feasible after the technique from Section 2.2.1 is in use, for otherwise the space complexity can be prohibitive for a large matrix.

This bookkeeping technique will serve as an important basis of the proposed preconditioner. There the bookkeeping scheme itself gets modified, and a rigorous proof is presented in Section 3.2 that the space complexity of the modified bookkeeping is upper-bounded by the space complexity of an exact matrix factorization.

3. Proof of Incomplete $\mathrm{LDL}^{\mathrm{T}}$ Factorization for R-matrices. In this section, we build an incomplete $\mathrm{LDL}^{\mathrm{T}}$ factorization of an R-matrix A by extracting information from the journey record of random walks. The proof is described in two stages: Section 3.1 proves that the journey record contains an approximate L factor, and then Section 3.2 proves that its non-zero pattern is a subset of that of the exact L factor. The formula of the diagonal D factor is derived in Section 3.3.

The factorization procedure is independent of the right-hand-side vector $\mathbf{b}$. Any appearance of $\mathbf{b}$ is symbolic, and the involved equations are true for any possible $\mathbf{b}$.

3.1. The Approximate Factorization. Suppose the dimension of the matrix A is N, and its k^{th} row corresponds to node k in Figure 2.1, $k = 1, 2, \cdots, N$. Without loss of generality, assume that in the stochastic solution, the nodes are processed in the natural ordering $1, 2, \cdots, N$. According to the speedup technique in Section 2.2.1, for random walks that start from node k, the nodes in the set $\{1, 2, \cdots, k-1\}$ are already solved and they now serve as home nodes where a random walk ends. The awards for reaching nodes $\{1, 2, \cdots, k-1\}$ are the estimated values of $\{x_1, x_2, \cdots, x_{k-1}\}$ respectively. Suppose that in equation (2.7), we choose $m_0 = 0$, and hence the motel prices are given by $m_i = -\frac{b_i}{A_{i,i}}$, for $i = k, k+1, \cdots, N$. Further,

- Let M_k be the number of walks carried out from node k.
- Let $H_{k,i}$ be the number of walks that start from node k and end at node $i \in \{1, 2, \cdots, k-1\}$.
- Let $J_{k,i}$ be the number of times that walks from node k pass the motel at node $i \in \{k, k+1, \cdots, N\}$.

Taking the average of the results of the M_k walks from node k, we obtain the following equation for the estimated solution entry:

$$x'_k = \frac{\sum_{i=1}^{k-1} H_{k,i} x'_i + \sum_{i=k}^{N} J_{k,i} \frac{b_i}{A_{i,i}}}{M_k}, \tag{3.1}$$

where x'_i is the estimated value of x_i for $i \in \{1, 2, \cdots, k-1\}$. Note that the awards received at the initial home nodes are ignored in the above equation since $m_0 = 0$. Moving the $H_{k,i}$ terms to the left side, we obtain

$$-\sum_{i=1}^{k-1} \frac{H_{k,i}}{M_k} x'_i + x'_k = \sum_{i=k}^{N} \frac{J_{k,i}}{M_k A_{i,i}} b_i. \tag{3.2}$$

By writing the above equation for $k = 1, 2, \cdots, N$, and assembling the N equations together into a matrix form, we obtain

$$Y\mathbf{x}' = Z\mathbf{b}, \tag{3.3}$$

where $\mathbf{x}'$ is the approximate solution produced by the stochastic solver; Y and Z are two square matrices of dimension N such that

$$\begin{aligned} Y_{k,k} &= 1, && \forall k, \\ Y_{k,i} &= -\frac{H_{k,i}}{M_k}, && \forall k > i, \\ Y_{k,i} &= 0, && \forall k < i, \\ Z_{k,i} &= \frac{J_{k,i}}{M_k A_{i,i}}, && \forall k \le i, \\ Z_{k,i} &= 0, && \forall k > i. \end{aligned} \tag{3.4}$$

These two matrices Y and Z are the journey record built by the bookkeeping technique in Section 2.2.2. Obviously Y is a lower triangular matrix with unit diagonal entries, Z is an upper triangular matrix, and their entries are independent of the right-hand-side vector $\mathbf{b}$. Once Y and Z are built from random walks, given any $\mathbf{b}$, one can apply equation (3.3) and find $\mathbf{x}'$ efficiently by a forward substitution.

It is worth pointing out the physical meaning of the matrix Y: the negative of an entry, $(-Y_{k,i})$, is asymptotically equal to the probability that a walk from node k ends at node i, when M_k goes to infinity. Another property of the matrix Y is that, if a walk from node k can never reach an original home node generated from a strictly-diagonally-dominant row of A, the row sum $\sum_i Y_{k,i} = \frac{M_k - \sum_i H_{k,i}}{M_k}$ is zero.

From equation (3.3), we have

$$Z^{-1}Y\mathbf{x}' = \mathbf{b}. \tag{3.5}$$

Since the vector $\mathbf{x}'$ in the above equation is an approximate solution to the original set of equations $A\mathbf{x} = \mathbf{b}$, it follows that[4]

$$Z^{-1}Y \approx A. \tag{3.6}$$

Because the inverse of an upper triangular matrix, Z^{-1}, is also upper triangular, equation (3.6) is in the form of an approximate "UL factorization" of A. The following

[4]For any vector $\mathbf{b}$, we have $\left(Z^{-1}Y\right)^{-1}\mathbf{b} = \mathbf{x}' \approx \mathbf{x} = A^{-1}\mathbf{b}$. Therefore, $A\left(Z^{-1}Y\right)^{-1}\mathbf{b} \approx \mathbf{b}$, and then $\left(I - A\left(Z^{-1}Y\right)^{-1}\right)\mathbf{b} \approx 0$. Since this is true for any vector $\mathbf{b}$, it must be true for eigenvectors of the matrix $\left(I - A\left(Z^{-1}Y\right)^{-1}\right)$, and it follows that the eigenvalues of the matrix $\left(I - A\left(Z^{-1}Y\right)^{-1}\right)$ are all close to zero. Thus we claim that $Z^{-1}Y \approx A$.

definition and lemma present a simple relation between UL factorization and the more commonly encountered LU factorization.

DEFINITION 3.1. *The operator* $\mathrm{rev}(\cdot)$ *is defined on square matrices: given a matrix* A *of dimension* N, $\mathrm{rev}(A)$ *is also a square matrix of dimension* N, *such that*

$$\mathrm{rev}(A)_{i,j} = A_{N+1-i,N+1-j}, \quad \forall i,j \in \{1,2,\cdots,N\}.$$

LEMMA 1. *Let* $A = LU$ *be the LU factorization of a square matrix* A, *then* $\mathrm{rev}(A) = \mathrm{rev}(L)\mathrm{rev}(U)$ *is true and is the UL factorization of* $\mathrm{rev}(A)$.

The proof of Lemma 1 is omitted. In simple terms, it states that the reverse-ordering of the LU factors of A are the UL factors of reverse-ordered A.

Applying Lemma 1 on equation (3.6), we obtain

$$\mathrm{rev}(Z^{-1})\mathrm{rev}(Y) \approx \mathrm{rev}(A). \tag{3.7}$$

Since A is an R-matrix and is symmetric, $\mathrm{rev}(A)$ must be also symmetric, and we can take the transpose of both sides, and have

$$\left(\mathrm{rev}(Y)\right)^{\mathrm{T}} \left(\mathrm{rev}(Z^{-1})\right)^{\mathrm{T}} \approx \mathrm{rev}(A), \tag{3.8}$$

which is in the form of a Doolittle LU factorization [10]: the matrix $\left(\mathrm{rev}(Y)\right)^{\mathrm{T}}$ is lower triangular with unit diagonal entries; the matrix $\left(\mathrm{rev}(Z^{-1})\right)^{\mathrm{T}}$ is upper triangular.

LEMMA 2. *The Doolittle LU factorization of a square matrix is unique.*

The proof of Lemma 2 is omitted. Let the exact Doolittle LU factorization of $\mathrm{rev}(A)$ be $\mathrm{rev}(A) = L_{\mathrm{rev}(A)}U_{\mathrm{rev}(A)}$, and its exact $\mathrm{LDL}^{\mathrm{T}}$ factorization be $\mathrm{rev}(A) = L_{\mathrm{rev}(A)}D_{\mathrm{rev}(A)}\left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}}$. Since (3.8) is an approximate Doolittle LU factorization of $\mathrm{rev}(A)$, while the exact Doolittle LU factorization is unique, it must be true that:

$$\left(\mathrm{rev}(Y)\right)^{\mathrm{T}} \approx L_{\mathrm{rev}(A)}, \tag{3.9}$$

$$\left(\mathrm{rev}(Z^{-1})\right)^{\mathrm{T}} \approx U_{\mathrm{rev}(A)} = D_{\mathrm{rev}(A)}\left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}}. \tag{3.10}$$

The above two equations indicate that from the matrix Y built by random walks, we can obtain an approximation to factor $L_{\mathrm{rev}(A)}$, and that the matrix Z contains redundant information. Section 3.3 shows how to estimate the matrix $D_{\mathrm{rev}(A)}$ utilizing only the diagonal entries of the matrix Z, and hence the rest of Z is not needed at all. According to equation (3.4), the matrix Y is the award register in the journey record and keeps track of end nodes of random walks, while the matrix Z is the motel-expense register and keeps track of all intermediate nodes of walks. Therefore the matrix Z is the dominant portion of the journey record, and by removing all of its off-diagonal entries, the modified journey record is significantly smaller than that in the original bookkeeping technique from Section 2.2.2. In fact, an upper bound on the number of non-zero entries in the matrix Y is provided in the next section.

3.2. The Incomplete Non-zero Pattern. The previous section proves that an approximate factorization of an R-matrix A can be obtained by random walks. It does not constitute a proof of incomplete factorization, because an incomplete factorization implies that its non-zero pattern must be a subset of that of the exact factor. Such a proof is the task of this section: to prove that an entry of $\left(\mathrm{rev}(Y)\right)^{\mathrm{T}}$ can be possibly non-zero only if the corresponding entry of $L_{\mathrm{rev}(A)}$ is non-zero.

For $i \neq j$, by Definition 3.1 and equation (3.4), the (i,j) entry of $(\mathrm{rev}(Y))^{\mathrm{T}}$ is

$$\left((\mathrm{rev}(Y))^{\mathrm{T}}\right)_{i,j} = Y_{N+1-j,N+1-i} = -\frac{H_{N+1-j,N+1-i}}{M_{N+1-j}}. \tag{3.11}$$

This value is non-zero if and only if $j < i$ and $H_{N+1-j,N+1-i} > 0$. In other words, at least one random walk starts from node $(N+1-j)$ and ends at node $(N+1-i)$.

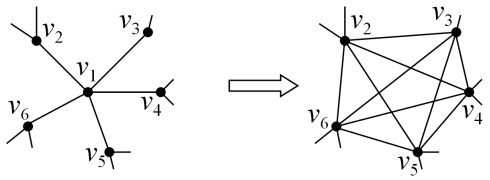

FIG. 3.1. *One step in symmetric Gaussian elimination.*

To analyze the non-zero pattern of $L_{\mathrm{rev}(A)}$, certain concepts from the literature of LU factorization are used here, and certain conclusions are cited without proof. More details can be found in [1], [10], [12], [13], [16]. Figure 3.1 illustrates one step in the exact Gaussian elimination of a matrix: removing one node from the matrix graph, and creating a clique among its neighbors. For example, when node v_1 is removed, a clique is formed for $\{v_2, v_3, v_4, v_5, v_6\}$, where the new edges correspond to fills added to the remaining matrix. At the same time, five non-zero values are written into the L matrix, at the five entries that are the intersections[5] of node v_1's corresponding column and the five rows that correspond to nodes $\{v_2, v_3, v_4, v_5, v_6\}$.

DEFINITION 3.2. *Given a graph $G = (V, E)$, a node set $S \subset V$, and nodes $v_1, v_2 \in V$ such that $v_1, v_2 \notin S$, node v_2 is said to be reachable from node v_1 through S if there exists a path between v_1 and v_2 such that all intermediate nodes, if any, belong to S.*

DEFINITION 3.3. *Given a graph $G = (V, E)$, a node set $S \subset V$, a node $v_1 \in V$ such that $v_1 \notin S$, the reachable set of v_1 through S, denoted $R(v_1, S)$, is defined as:*

$$R(v_1, S) = \{v_2 \notin S | v_2 \text{ is reachable from } v_1 \text{ through } S\}.$$

Note that if v_1 and v_2 are adjacent, there is no intermediate node on the path between them, then Definition 3.2 is satisfied for any node set S. Therefore, $R(v_1, S)$ always includes the direct neighbors of v_1 that do not belong to S.

Given an R-matrix A, let G be its matrix graph, let L be the complete L factor in its exact $\mathrm{LDL}^{\mathrm{T}}$ factorization, and let v_1 and v_2 be two nodes in G. Note that every node in G has a corresponding row and a corresponding column in A and in L. The following lemma can be derived from [13, p. 98], [16].

LEMMA 3. *The entry in L at the intersection of column v_1 and row v_2 is non-zero if and only if:*

1. *v_1 is eliminated prior to v_2 during Gaussian elimination.*
2. *$v_2 \in R(v_1, \{\text{nodes eliminated prior to } v_1\})$.*

[5] In this section, rows and columns of a matrix are often identified by their corresponding nodes in the matrix graph, and matrix entries are often identified as intersections of rows and columns. The reason is that such references are independent of the matrix ordering, and thereby avoid confusion due to the two orderings involved in the discussion.

We now apply this lemma on $L_{\mathrm{rev}(A)}$. Because the factorization of $\mathrm{rev}(A)$ is performed in the reverse ordering, i.e., $N, N-1, \cdots, 1$, the (i,j) entry of $L_{\mathrm{rev}(A)}$ is the entry at the intersection of the column that corresponds to node $(N+1-j)$ and the row that corresponds to node $(N+1-i)$. This entry is non-zero if and only if both of the following conditions are met.

1. Node $(N+1-j)$ is eliminated prior to node $(N+1-i)$.
2. $(N+1-i) \in R\,(N+1-j, S_j)$,
   where $S_j = \{\text{nodes eliminated prior to } N+1-j\}$.

Again, because the Gaussian elimination is carried out in the reverse ordering $N, N-1, \cdots, 1$, the first condition implies that $N+1-j > N+1-i$ and hence $j < i$. The node set S_j in the second condition is simply $\{N+2-j, N+3-j, \cdots, N\}$.

Recall that equation (3.11) is non-zero if there is at least one random walk that starts from node $(N+1-j)$ and ends at node $(N+1-i)$. Also recall that according to Section 2.2.1, when random walks are performed from node $(N+1-j)$, nodes $\{1, 2, \cdots, N-j\}$ are home nodes that walks terminate, while nodes $S_j = \{N+2-j, N+3-j, \cdots, N\}$ are the motel nodes that a walk can pass through. Therefore, a walk from node $(N+1-j)$ can possibly end at node $(N+1-i)$, only if $(N+1-i)$ is reachable from $(N+1-j)$ through the motel node set, i.e., node set S_j.

By now it is proven that both conditions for $\left(L_{\mathrm{rev}(A)}\right)_{i,j}$ to be non-zero are necessary conditions for equation (3.11) to be non-zero. Therefore, the non-zero pattern of $(\mathrm{rev}(Y))^{\mathrm{T}}$ is a subset of the non-zero pattern of $L_{\mathrm{rev}(A)}$. Together, this conclusion and equation (3.9) give rise to the following lemma.

LEMMA 4. *$(\mathrm{rev}(Y))^{\mathrm{T}}$ is the L factor of an incomplete* $\mathrm{LDL}^{\mathrm{T}}$ *factorization of the matrix* $\mathrm{rev}(A)$.

This lemma indicates that, from random walks, we can obtain an incomplete $\mathrm{LDL}^{\mathrm{T}}$ factorization of the matrix A in its reversed index ordering. The remaining approximate diagonal matrix D is derived in the next section.

3.3. The Diagonal Component. To evaluate the approximate D matrix, we take the transpose of both sides of equation (3.10), and obtain

$$\mathrm{rev}(Z^{-1}) \approx L_{\mathrm{rev}(A)} D_{\mathrm{rev}(A)}. \tag{3.12}$$

LEMMA 5. *For a non-singular square matrix A,* $\mathrm{rev}(A^{-1}) = (\mathrm{rev}(A))^{-1}$.

The proof of Lemma 5 is omitted. Applying it to equation (3.12), we have

$$\begin{aligned} (\mathrm{rev}(Z))^{-1} &\approx L_{\mathrm{rev}(A)} D_{\mathrm{rev}(A)}, \\ I &\approx \mathrm{rev}(Z) L_{\mathrm{rev}(A)} D_{\mathrm{rev}(A)}. \end{aligned} \tag{3.13}$$

Recall that $\mathrm{rev}(Z)$ and $L_{\mathrm{rev}(A)}$ are both lower triangular, that $L_{\mathrm{rev}(A)}$ has unit diagonal entries, and that $D_{\mathrm{rev}(A)}$ is a diagonal matrix. Therefore, the (i,i) diagonal entry in the above equation is simply

$$\begin{aligned} (\mathrm{rev}(Z))_{i,i} \left(L_{\mathrm{rev}(A)}\right)_{i,i} \left(D_{\mathrm{rev}(A)}\right)_{i,i} &\approx 1, \\ \left(D_{\mathrm{rev}(A)}\right)_{i,i} &\approx \frac{1}{(\mathrm{rev}(Z))_{i,i}}. \end{aligned} \tag{3.14}$$

Applying Definition 3.1 and equation (3.4), we finally have the equation for computing the approximate D factor, given as follows:

$$\left(D_{\mathrm{rev}(A)}\right)_{i,i} \approx \frac{1}{Z_{N+1-i,N+1-i}} = \frac{M_{N+1-i} A_{N+1-i,N+1-i}}{J_{N+1-i,N+1-i}}. \tag{3.15}$$

It is worth pointing out the physical meaning of the quantity $\frac{J_{N+1-i,N+1-i}}{M_{N+1-i}}$. It is the average number of times that a walk from node $N+1-i$ passes node $N+1-i$ itself; in other words, it is the average number of times that the walker returns to his/her starting point before the game is over. Equation (3.15) indicates that an entry in the D factor is equal to the corresponding diagonal entry of the original matrix A divided by the expected number of returns.

4. Proof of Incomplete LDU/LDL$^{\mathrm{T}}$ Factorization for Diagonally Dominant Matrices. The previous two sections have presented the theory of stochastic preconditioning in the context of R-matrices. In this section, we show that the proposed preconditioner applies to any matrix A that satisfies Definition 1.2.

4.1. Asymmetric Matrices. Let us first remove the symmetry requirement on the matrix A. Recall that the construction of the random walk game and the derivation of equation (3.7) do not require A to be symmetric. Therefore, matrices Y and Z can still be obtained by (3.4) from random walks, which we will refer to as Y_A and Z_A in this section, and equation (3.7) remains true for an asymmetric matrix A. Suppose the exact LDU factorization of the matrix $\mathrm{rev}(A)$ is $\mathrm{rev}(A) = L_{\mathrm{rev}(A)}D_{\mathrm{rev}(A)}U_{\mathrm{rev}(A)}$, where $L_{\mathrm{rev}(A)}$ is a lower triangular matrix with unit diagonal entries, $U_{\mathrm{rev}(A)}$ is an upper triangular matrix with unit diagonal entries, and $D_{\mathrm{rev}(A)}$ is a diagonal matrix. It is easy to show, based on Lemma 2, that the LDU factorization is also unique. Substituting the factorization into equation (3.7), we have

$$\mathrm{rev}(Z_A^{-1})\mathrm{rev}(Y_A) \approx L_{\mathrm{rev}(A)}D_{\mathrm{rev}(A)}U_{\mathrm{rev}(A)}. \tag{4.1}$$

Based on the uniqueness of LDU factorization, it must be true that

$$\mathrm{rev}(Y_A) \approx U_{\mathrm{rev}(A)}, \tag{4.2}$$

$$\mathrm{rev}(Z_A^{-1}) \approx L_{\mathrm{rev}(A)}D_{\mathrm{rev}(A)}. \tag{4.3}$$

By equation (4.2), we can approximate $U_{\mathrm{rev}(A)}$ based on Y_A; by equation (4.3), and through the same derivation as in Section 3.3, we can approximate $D_{\mathrm{rev}(A)}$ based on the diagonal entries of Z_A. The remaining question is how to obtain $L_{\mathrm{rev}(A)}$.

Suppose we construct a random walk game based on A^{T} instead of A, and obtain matrices $Y_{A^{\mathrm{T}}}$ and $Z_{A^{\mathrm{T}}}$ based on equation (3.4). Then by equation (4.2), we have

$$\mathrm{rev}(Y_{A^{\mathrm{T}}}) \approx U_{\mathrm{rev}(A^{\mathrm{T}})}, \tag{4.4}$$

where $U_{\mathrm{rev}(A^{\mathrm{T}})}$ is the exact U factor in the LDU factorization of $\mathrm{rev}(A^{\mathrm{T}})$. It is easy to derive the following:

$$\mathrm{rev}(A^{\mathrm{T}}) = (\mathrm{rev}(A))^{\mathrm{T}} = \left(U_{\mathrm{rev}(A)}\right)^{\mathrm{T}} D_{\mathrm{rev}(A)} \left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}}, \tag{4.5}$$

$$L_{\mathrm{rev}(A^{\mathrm{T}})}D_{\mathrm{rev}(A^{\mathrm{T}})}U_{\mathrm{rev}(A^{\mathrm{T}})} = \left(U_{\mathrm{rev}(A)}\right)^{\mathrm{T}} D_{\mathrm{rev}(A)} \left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}}. \tag{4.6}$$

Based on the uniqueness of the LDU factorization, it must be true that

$$\left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}} = U_{\mathrm{rev}(A^{\mathrm{T}})}. \tag{4.7}$$

By (4.4) and (4.7), we finally have

$$\mathrm{rev}(Y_{A^{\mathrm{T}}}) \approx \left(L_{\mathrm{rev}(A)}\right)^{\mathrm{T}},$$

$$(\mathrm{rev}(Y_{A^{\mathrm{T}}}))^{\mathrm{T}} \approx L_{\mathrm{rev}(A)}. \tag{4.8}$$

In other words, we can approximate $L_{\mathrm{rev}(A)}$ based on $Y_{A^{\mathrm{T}}}$.

As a summary, when the matrix A is asymmetric, we need to construct two random walk games for A and A^{T}, and then based on the two Y matrices and the diagonal entries of one of the Z matrices[6], we can approximate the LDU factorization of $\mathrm{rev}(A)$ based on equations (3.4), (3.15), (4.2), and (4.8). Both the time complexity and space complexity of building the preconditioner become roughly twice those of the symmetric case: this is the same behavior as a traditional ILU.

The remainder of this section gives an outline of the proof that the non-zero patterns of the above approximate L and U factors by (4.2) and (4.8) are subsets of those of the exact factors. The proof is essentially the same as Section 3.2, and we will only point out the differences due to asymmetry.

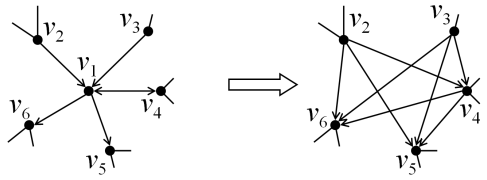

FIG. 4.1. *One step in asymmetric Gaussian elimination.*

Figure 4.1 illustrates one step in the exact Gaussian elimination of an asymmetric matrix: removing one node from the matrix graph, and adding an edge from each of its fan-in nodes to each of its fan-out nodes; the new edges correspond to fills added to the remaining matrix. In the example of Figure 4.1, when node v_1 is removed, edges are added from each of its fan-in nodes $\{v_2, v_3, v_4\}$ to each of its fan-out nodes $\{v_4, v_5, v_6\}$, with the exception that no edge is added from v_4 to itself. At the same time, three non-zero values are written into the L matrix, at the three entries that are the intersections of node v_1's corresponding column and the three rows that correspond to nodes $\{v_2, v_3, v_4\}$; three non-zero values are written into the U matrix, at the three entries that are the intersections of node v_1's corresponding row and the three columns that correspond to nodes $\{v_4, v_5, v_6\}$. Utilizing Figure 4.1, we can prove Lemma 6, which is the asymmetric version of Lemma 3; the proof is omitted.

DEFINITION 4.1. *Given a directed graph $G = (V, E)$, a node set $S \subset V$, and nodes $v_1, v_2 \in V$ such that $v_1, v_2 \notin S$, node v_2 is said to be reachable from node v_1 through S if there exists a directed path from v_1 to v_2 such that all intermediate nodes, if any, belong to S.*

DEFINITION 4.2. *Given a directed graph $G = (V, E)$, a node set $S \subset V$, a node $v_1 \in V$ such that $v_1 \notin S$, the reachable set of v_1 through S, denoted $R(v_1, S)$, is defined as:*

$$R(v_1, S) = \{v_2 \notin S | v_2 \text{ is reachable from } v_1 \text{ through } S\}.$$

LEMMA 6. *Suppose the exact LDU factorization of a square matrix A exists and is $A = LDU$. Let v_1 and v_2 be two nodes in the matrix graph of A. The entry in L at the intersection of column v_1 and row v_2 is non-zero if and only if:*

1. *v_1 is eliminated prior to v_2 during Gaussian elimination.*
2. *$v_1 \in R(v_2, \{\text{nodes eliminated prior to } v_1\})$.*

[6]Due to the uniqueness of the LDU, it does not matter the diagonals of which Z are used.

The entry in U at the intersection of column v_1 and row v_2 is non-zero if and only if:

1. *v_2 is eliminated prior to v_1 during Gaussian elimination.*
2. *$v_1 \in R\left(v_2, \{\text{nodes eliminated prior to } v_2\}\right)$.*

Applying Lemma 6 on the matrix $\text{rev}(A)$, and through the same procedure as Section 3.2, it can be proven[7] that the conditions in Lemma 6 are necessary conditions for the non-zero entries in (4.2) and (4.8). Therefore, the non-zero patterns of the approximate L and U factors by (4.2) and (4.8) are subsets of those of the exact factors, and hence we have the following lemma, the asymmetric version of Lemma 4.

LEMMA 7. *$\left(\text{rev}(Y_{A^{\mathrm{T}}})\right)^{\mathrm{T}}$ is the L factor, and $\text{rev}(Y_A)$ is the U factor, of an incomplete LDU factorization of the matrix $\text{rev}(A)$.*

4.2. Random Walk Game with Scaling. This section shows that our method does not require A to be an M-matrix: as long as the matrix A satisfies Definition 1.2, its off-diagonal entries can be positive or complex-valued.

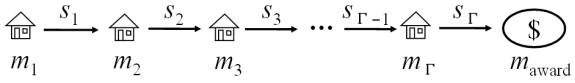

FIG. 4.2. *A random walk in the modified game with scaling.*

To remove the M-matrix constraint, a new game is designed by defining a scaling factor[8] s on each direction of every edge in the original game from Section 2.1. Such a scaling factor becomes effective when a random walk passes that particular edge in that particular direction, and remains effective until this random walk ends.

Let us look at the stochastic solver first. A walk is shown in Figure 4.2: it passes a number of motels, each of which has its price m_l, $l \in \{1, 2, \cdots, \Gamma\}$, and ends at a home with certain award value m_{award}. The monetary gain of this walk is defined as

$$\text{gain} = -m_1 - s_1 m_2 - s_1 s_2 m_3 - \cdots - \prod_{l=1}^{\Gamma-1} s_l \cdot m_\Gamma + \prod_{l=1}^{\Gamma} s_l \cdot m_{\text{award}}. \tag{4.9}$$

In simple terms, this new game is different from the original game in that each transaction amount during the walk is scaled by the product of the currently active scaling factors. Define the expected gain function f to be the same as in equation (2.2), and it is easy to derive the replacement of equation (2.4):

$$f(i) = \sum_{l=1}^{\text{degree}(i)} p_{i,l} s_{i,l} f(l) - m_i, \tag{4.10}$$

where $s_{i,l}$ denotes the scaling factor associated with the direction $i \to l$ of the edge between i and l, and the rest of the symbols are the same as defined in (2.4).

In this section, we require that each s factor has an absolute value of one. Therefore, the scaling in the new game never changes the magnitude of a monetary transaction, and only changes its sign or phase. Section 8 will discuss other scaling schemes.

Due to the degrees of freedom introduced by the scaling factors, the allowable matrix A in (4.10) is now any matrix that is diagonally dominant. Hence, given any matrix A that satisfies Definition 1.2, a corresponding game can be constructed.

[7] It is important to note that, in the random walk game constructed for A^{T}, the road directions are all reversed, compared to the game for A.

[8] A similar concept of scaling factors can be found in [11], though tailored to its game design.

For stochastic preconditioning, this new game design with scaling leads to the redefinition of the H and J quantities in Section 3.1 and in all subsequent formulations:

$$H_{k,i} = \sum_{\text{walk from } k \text{ to } i} \left(\prod_{\text{step } l} s_l \right), \qquad \forall k > i, \tag{4.11}$$

$$J_{k,i} = \sum_{\text{partial walk from } k \text{ to } i} \left(\prod_{\text{step } l} s_l \right), \quad \forall k \leq i. \tag{4.12}$$

Note that a single walk from k may contribute multiple terms in the summation in (4.12), if it passes node i multiple times, and each such time the partial walk is defined as the segment from the start k to the current passing of i; a partial walk in (4.12) always starts from the initial start at k, and not from any intermediate passing of k; in the case of $i = k$, the summation in equation (4.12) includes a zero-length partial walk for each walk performed from k, and the zero-length product of scaling factors is defined to be one. When all s factors are set to be one, the new game design is identical to the original one, and equations (4.11) and (4.12) simply degenerate to the original definitions of H and J in Section 3.1.

After applying the new formulas (4.11) and (4.12), all the equations and derivations in Sections 3 and 4 are valid, and now we have proven the proposed preconditioning method for any matrix A that satisfies Definition 1.2.

5. Relation to and Comparison with ILU/IC/Approximate-inverse. In this section, the proposed hybrid solver for diagonally dominant matrices is presented in its entirety, by summarizing the previous three sections; its relation to and comparison with several existing preconditioning methods are discussed.

5.1. The Hybrid Solver. We begin by defining the $\text{rev}(\cdot)$ operator on vectors.

DEFINITION 5.1. *The operator* $\text{rev}(\cdot)$ *is defined on vectors: given vector* $\mathbf{x}$ *of length* N, $\text{rev}(\mathbf{x})$ *is a vector of length* N *and* $\text{rev}(\mathbf{x})_i = \mathbf{x}_{N+1-i}, \forall i \in \{1, 2, \cdots, N\}$.

It is easy to verify that $A\mathbf{x} = \mathbf{b}$ is equivalent to $\text{rev}(A)\text{rev}(\mathbf{x}) = \text{rev}(\mathbf{b})$. By now, we have collected the necessary pieces, and the hybrid solver is summarized in the pseudocodes in Algorithm 1 and Algorithm 2. Here conjugate gradient (CG) and biconjugate gradient (BCG) are listed as example iterative engines, and the proposed preconditioner can work with any iterative solver.

ALGORITHM 1. *The hybrid solver for symmetric matrices:*

```
Precondition {
   Run random walks, build the matrix Y and find diagonal
        entries of Z using equations (3.4)(4.11)(4.12);
   Build L_rev(A) using equation (3.9);
   Build D_rev(A) using equation (3.15);
}
Given b, solve {
   Convert Ax = b to rev(A)rev(x) = rev(b);
   Apply CG on rev(A)rev(x) = rev(b) with the
        preconditioner (L_rev(A) D_rev(A) L_rev(A)^T)^-1;
   Convert rev(x) to x;
}
```

ALGORITHM 2. *The hybrid solver for asymmetric matrices:*

```
Precondition {
   Run random walks in the game for A, build the matrix Y_A and
        find diagonal entries of Z_A using equations (3.4)(4.11)(4.12);
   Build U_rev(A) using equation (4.2) based on Y_A;
   Build D_rev(A) using equation (3.15) based on diagonals of Z_A;
   Run random walks in the game for A^T, build the matrix Y_A^T
        using equations (3.4)(4.11);
   Build L_rev(A) using equation (4.8) based on Y_A^T;
}
Given b, solve {
   Convert Ax = b to rev(A)rev(x) = rev(b);
   Apply BCG on rev(A)rev(x) = rev(b) with the
        preconditioner (L_rev(A) D_rev(A) U_rev(A))^{-1};
   Convert rev(x) to x;
}
```

5.2. Comparison with ILU/IC. This section presents an algorithmic argument that the proposed preconditioner has better quality than the traditional ILU or IC preconditioners. In other words, if the incomplete LDU/LDL$^{\mathrm{T}}$ factorization built by our method has the same number of non-zero entries as one that is built by a traditional approach, we expect a better approximation to the exact LDU/LDL$^{\mathrm{T}}$ factorization, and a preconditioned Krylov-subspace solver to converge with fewer iterations. Experimental condition number comparisons are provided in Section 7.

Let us use the symmetric ICT approach as an example of traditional preconditioning methods; a similar argument can be made for other existing techniques for either symmetric or asymmetric matrices, as long as they are sequential procedures based on Gaussian elimination. Suppose in Figure 3.1, when eliminating node v_1, the new edge between nodes v_2 and v_3 corresponds to an entry whose value falls below a specified threshold, then ICT drops that entry from the remaining matrix, and that edge is removed from the remaining matrix graph. Later when the algorithm reaches the stage of eliminating node v_2, because of that missing edge, no edge is created from v_3 to the neighbors of v_2, and thus more edges are missing, and this new set of missing edges then affect later computations accordingly. Therefore, an early decision of dropping an entry is propagated throughout the ICT process. On the one hand, this leads to the sparsity of the preconditioner, which is desirable; on the other hand, error accumulation occurs, and later columns of the resulting incomplete Cholesky factor may deviate from the exact Cholesky factor by a larger amount than the planned threshold. Such error accumulation gets exacerbated for larger and denser matrices.

Let us generalize the above argument to all traditional ILU/IC methods, and state it from a different perspective: traditional ILU/IC methods are all sequential procedures where later rows/columns are calculated based on previously computed rows/columns, which may contain errors due to dropped entries; such data dependency can be represented by a directed acyclic graph; the depth of this dependency graph increases for larger and denser matrices, and we argue that higher data-dependency depth implies stronger error accumulation.

The hybrid solver does not suffer from such error accumulation. According to equation (3.4), the calculation of $Y_{k,i}$, which will become a single entry in the resulting incomplete LDU/LDL$^{\mathrm{T}}$ factorization, replies on no other Y or Z entries. In other words, equation (3.4) provides a direct unbiased Monte Carlo estimator for each single

entry in an LDU/LDL$^{\mathrm{T}}$ factorization, and allows obtaining any single entry in an LDU/LDL$^{\mathrm{T}}$ factorization without computing any other entry. Practically, a row of the matrix Y can be computed together because the needed random walks are shared by its entries. When we run random walks from node k and collect the $H_{k,i}$ values to build the k^{th} row of the matrix Y according to equation (3.4), we only know that the nodes $\{1, 2, \cdots, k-1\}$ are homes, and this is the only information needed[9]. Therefore, the quality of the computed k^{th} row of the matrix Y does not affect other rows in any way; each row is responsible for its own accuracy, according to a criterion to be discussed in Section 6.1. In fact, in a parallel computing environment, the computation of each row of Y can be assigned to a different processor.

It is important to distinguish the error accumulation discussed in this section and the cost of bias discussed at the end of Section 2.2.1. For an iterative solver with implicit factorization-based preconditioning, there can be two different types of error accumulation: first, the process of building the preconditioner can accumulate error, as described earlier with the ICT example; second, the process of applying the preconditioner inside an iterative solver can accumulate error, i.e., the forward/backward substitution is a sequential solving process and later entries in the resulting vector are calculated based on earlier entries which contain errors. That bias discussed at the end of Section 2.2.1, in the context of the hybrid solver, maps to the second type, the error accumulation in applying the preconditioner; such bias or error propagation is inevitable in all iterative solvers as long as an implicit factorization-based preconditioner is in use. Our claim here is that the hybrid solver is free of error accumulation in building the preconditioner, and not in applying the preconditioner[10].

In summary, due to the absence of error accumulation in building the preconditioner, we expect the proposed stochastic preconditioning to outperform traditional ILU/IC methods, and we expect larger advantage on larger and denser matrices.

5.3. Relation to Factored Approximate Inverses. A factored-approximate-inverse preconditioner approximates A^{-1} in the form of a product of, typically three, matrices, and are obtained by either a norm-minimization approach or an inverse triangular factorization [3], [4]. As an explicit preconditioning method, it has the advantages of parallelism as well as better stability, compared to ILU/IC, although its accuracy-size tradeoff is often inferior. It was shown in [4] that triangular factored approximate inverses can be viewed as approximations of the inverses of LDU/LDL$^{\mathrm{T}}$ factors. This section shows that such a factored-approximate-inverse preconditioner can be easily produced by our stochastic procedure, and that we again have the advantage of being free of error accumulation over existing methods.

Applying Lemma 5 on equation (4.3), we have

$$\begin{aligned} \mathrm{rev}(Z_A)^{-1} &\approx L_{\mathrm{rev}(A)} D_{\mathrm{rev}(A)}, \\ \mathrm{rev}(Z_A) &\approx D^{-1}_{\mathrm{rev}(A)} L^{-1}_{\mathrm{rev}(A)}, \end{aligned}$$

[9] As a side note, recall the property of the matrix Y that, for an R-matrix A, if a walk from node k can never reach an original home node generated from a strictly-diagonally-dominant row of A, the row sum $\sum_i Y_{k,i} = \frac{M_k - \sum_i H_{k,i}}{M_k}$ is guaranteed to be zero. This is a property of an exact LDL$^{\mathrm{T}}$ factorization of an R-matrix, and is maintained naturally in our preconditioner. This aspect is similar in flavor as MILU/MIC, which achieves this property by compensating their diagonal entries.

[10] After a row of the matrix Y is calculated, it is possible to add a postprocessing step to drop insignificant entries. The criterion can be any of the strategies used in traditional incomplete factorization methods. With such postprocessing, the hybrid solver still maintains the advantage of independence between row calculations. This is not included in our implementation.

$$D_{\mathrm{rev}(A)}\mathrm{rev}(Z_A) \approx L^{-1}_{\mathrm{rev}(A)}. \tag{5.1}$$

Since Z_A can be built by equation (3.4) and $D_{\mathrm{rev}(A)}$ can be built by equation (3.15), the above equation gives a stochastic approach to approximate the matrix $L^{-1}_{\mathrm{rev}(A)}$.

Suppose we construct a game based on A^{T} instead of A, and obtain matrices $Z_{A^{\mathrm{T}}}$ and $D_{\mathrm{rev}(A^{\mathrm{T}})}$ based on equations (3.4) and (3.15). Then by equation (5.1), we have

$$D_{\mathrm{rev}(A^{\mathrm{T}})}\mathrm{rev}(Z_{A^{\mathrm{T}}}) \approx L^{-1}_{\mathrm{rev}(A^{\mathrm{T}})}. \tag{5.2}$$

Based on the uniqueness of the LDU factorization, and by (4.6), it must be true that

$$L_{\mathrm{rev}(A^{\mathrm{T}})} \approx U^{\mathrm{T}}_{\mathrm{rev}(A)}. \tag{5.3}$$

Combining the above two equations, we get

$$D_{\mathrm{rev}(A^{\mathrm{T}})}\mathrm{rev}(Z_{A^{\mathrm{T}}}) \approx \left(U^{\mathrm{T}}_{\mathrm{rev}(A)}\right)^{-1},$$

$$D_{\mathrm{rev}(A^{\mathrm{T}})}\mathrm{rev}(Z_{A^{\mathrm{T}}}) \approx \left(U^{-1}_{\mathrm{rev}(A)}\right)^{\mathrm{T}},$$

$$\left(\mathrm{rev}(Z_{A^{\mathrm{T}}})\right)^{\mathrm{T}} D^{\mathrm{T}}_{\mathrm{rev}(A^{\mathrm{T}})} \approx U^{-1}_{\mathrm{rev}(A)}. \tag{5.4}$$

The above equation gives a stochastic approach to approximate the matrix $U^{-1}_{\mathrm{rev}(A)}$.

With equations (5.1) and (5.4), we can construct a factored-approximate-inverse preconditioner in the form of $\tilde{U}^{-1}_{\mathrm{rev}(A)}\tilde{D}^{-1}_{\mathrm{rev}(A)}\tilde{L}^{-1}_{\mathrm{rev}(A)}$, where $\tilde{U}^{-1}_{\mathrm{rev}(A)}$, $\tilde{D}^{-1}_{\mathrm{rev}(A)}$ and $\tilde{L}^{-1}_{\mathrm{rev}(A)}$ approximate $U^{-1}_{\mathrm{rev}(A)}$, $D^{-1}_{\mathrm{rev}(A)}$ and $L^{-1}_{\mathrm{rev}(A)}$ respectively. For a symmetric matrix A, since $U^{-1}_{\mathrm{rev}(A)} = \left(L^{-1}_{\mathrm{rev}(A)}\right)^{\mathrm{T}}$, only equation (5.1) is needed. It is worth pointing out the physical meaning of this preconditioner: the approximate inverse factor is contained in the motel register matrix Z rather than the award register matrix Y.

Comparing the above stochastic approach against existing methods for building triangular factored approximate inverses, we again note that existing methods are sequential procedures where later rows/columns are calculated based on previously computed rows/columns which contain errors, and that our approach has the advantage of being free of error accumulation. Therefore, the stochastic approximate-inverse preconditioning in this section may potentially give a better performance.

In this paper, we choose to focus on implicit stochastic preconditioning, and will not further discuss the stochastic approximate inverse method beyond this section.

6. Implementation Issues. This section describes several implementation aspects of the stochastic preconditioning. The goal is to minimize the cost of building the preconditioner, and to achieve a better accuracy-size tradeoff.

6.1. Stopping Criterion. The topic of this section is the accuracy control of the preconditioner, that is, how should one choose M_k, the number of walks from node k, to achieve a certain accuracy level in estimating its corresponding entries in the LDU/LDL$^{\mathrm{T}}$ factorization. In Section 2.1, the stopping criterion in the stochastic solver is defined on the result of a walk; it is not applicable to preconditioning because here it is necessary for the criterion to be independent of the right-hand-side vector $\mathbf{b}$. In our implementation, a new stopping criterion is defined on a value that is a function of only the matrix A, as follows. Let $\Xi_k = E\,[\text{length of a walk from node } k]$, and let Ξ'_k be the average length of the M_k walks. The stopping criterion is

$$P[-\Delta < \frac{\Xi'_k - \Xi_k}{\Xi_k} < \Delta] > \alpha, \tag{6.1}$$

where Δ is a relative error margin, and α is a confidence level, for example $\alpha = 99\%$. Practically, this criterion is checked by the following inequality:

$$\frac{\Delta \Xi_k' \sqrt{M_k}}{\sigma_k} > Q^{-1}\left(\frac{1-\alpha}{2}\right), \tag{6.2}$$

where σ_k is the standard deviation of the lengths of the M_k walks, and Q is the standard normal complementary cumulative distribution function. Thus, M_k is determined dynamically, and random walks are run from node k until condition (6.1) is satisfied. Different choices of Δ and/or α result in different tradeoff points between accuracy and size of the proposed preconditioner. Our experience with 3D Laplacian matrices and VLSI circuit placement matrices, for which results will be presented in Section 7, suggests that a Δ value between 30% and 40% with $\alpha = 99\%$ gives a reasonably good tradeoff, and the condition (6.1) is typically satisfied within 100 walks. In practice, it is also necessary to impose a lower bound on M_k, e.g., 20 walks.

Note that this is not the only way to design the stopping criterion: it can also be defined on quantities other than Ξ_k (for example, the expected number of returns).

6.2. Exact Computations for One-step Walks. The technique in this section is a special treatment for the random walks with length 1, which we refer to as one-step walks. Such a walk occurs when an immediate neighbor of the starting node is a home node, and the first step of the walks happens to go there. The idea is to replace stochastic computations of one-step walks with their deterministic limits.

Without loss of generality, assume that the node ordering in the hybrid solver is the natural ordering $1, 2, \cdots, N$. Let us consider the M_k walks from node k, and suppose at least one of its immediate neighboring nodes is a home, which could be either an initial home if the k^{th} row of the matrix A is strictly diagonally dominant, or a node j such that $j < k$. Among the M_k walks, let $M_{k,1}$ be the number of one-step walks, and let $H_{k,i,1}$ be the portion of (4.11) contributed by one-step walks that go to node i, where node i is an arbitrary node such that $i < k$. For the case that node i is not adjacent to node k, $H_{k,i,1}$ is simply zero. Equation (3.4) can be rewritten as

$$Y_{k,i} = -\frac{H_{k,i}}{M_k} = -\frac{H_{k,i,1}}{M_k} - \left(\frac{M_k - M_{k,1}}{M_k}\right) \cdot \left(\frac{H_{k,i} - H_{k,i,1}}{M_k - M_{k,1}}\right). \tag{6.3}$$

Applying the H values in (4.11), the mapping between (2.7) and (4.10), and the fact that every scaling factor s has unit magnitude, we can derive the following:

$$\lim_{M_k \to \infty} \frac{H_{k,i,1}}{M_k} = s_{\text{step } k \text{ to } i} \cdot P\left[\text{first step goes to node } i\right] = -\frac{A_{k,i}}{A_{k,k}}, \tag{6.4}$$

$$\begin{aligned} \lim_{M_k \to \infty} \frac{M_k - M_{k,1}}{M_k} &= P\left[\text{first step goes to a non-absorbing node}\right] \\ &= \sum_{j>k} P\left[\text{first step goes to node } j\right] = \frac{\sum_{j>k} |A_{k,j}|}{A_{k,k}}. \end{aligned} \tag{6.5}$$

We modify equation (6.3) by replacing the term $\frac{H_{k,i,1}}{M_k}$ and the term $\frac{M_k - M_{k,1}}{M_k}$ with their limits given by the above two equations, and obtain a new formula for $Y_{k,i}$:

$$Y_{k,i} = \frac{A_{k,i}}{A_{k,k}} - \left(\frac{\sum_{j>k} |A_{k,j}|}{A_{k,k}}\right) \cdot \left(\frac{H_{k,i} - H_{k,i,1}}{M_k - M_{k,1}}\right). \tag{6.6}$$

The remaining stochastic part of this new equation is the term $\frac{H_{k,i}-H_{k,i,1}}{M_k-M_{k,1}}$, which can be evaluated by considering only random walks whose length is at least two; in other words, one-step walks are ignored. In implementation, this can be realized by simulating the first step of walks by randomly picking one of the non-absorbing neighbors of node k; note that then the number of random walks would automatically be $(M_k - M_{k,1})$, and no adjustment is needed.

With a similar derivation, the $Z_{k,k}$ formula[11] in (3.4) can be modified to

$$Z_{k,k} = \frac{1}{A_{k,k}} - \frac{\sum_{j>k} |A_{k,j}|}{A_{k,k}^2} + \left(\frac{\sum_{j>k} |A_{k,j}|}{A_{k,k}^2} \right) \cdot \left(\frac{J_{k,k} - J_{k,k,1}}{M_k - M_{k,1}} \right), \tag{6.7}$$

where $J_{k,k,1}$ is the portion of (4.12) contributed by one-step walks. Obviously $J_{k,k,1} = M_{k,1}$, and therefore

$$Z_{k,k} = \frac{1}{A_{k,k}} + \left(\frac{\sum_{j>k} |A_{k,j}|}{A_{k,k}^2} \right) \cdot \left(\frac{J_{k,k} - M_{k,1}}{M_k - M_{k,1}} - 1 \right). \tag{6.8}$$

The remaining stochastic part of this new equation, the term $\frac{J_{k,k}-M_{k,1}}{M_k-M_{k,1}}$, again can be evaluated by considering only random walks with length being at least two. Practically, such computation is concurrent with evaluating $Y_{k,i}$'s based on equation (6.6).

The benefit of replacing (3.4) with equations (6.6) and (6.8) is twofold:

- Part of the evaluation of $Y_{k,i}$ and $Z_{k,k}$ entries is converted from stochastic computation to its deterministic limit, and the accuracy is potentially improved. For a node k where all neighbors have lower indices, i.e., when all neighbors are home nodes, equations (6.6) and (6.8) become exact: they translate to the exact entry values in the complete LDU/LDL$^{\mathrm{T}}$ factorization.
- By avoiding simulating one-step walks, the amount of computation in building the preconditioner is reduced. For a node k where all neighbors are homes, the stochastic parts of (6.6) and (6.8) disappear, and no walks are needed.

6.3. Reusing Walks. Without loss of generality, assume that the node ordering in the hybrid solver is the natural ordering $1, 2, \cdots, N$. A sampled random walk is completely specified by the node indices along the way, and hence can be viewed as a sequence of integers $\{k_1, k_2, \cdots, k_\Gamma\}$, such that $k_1 > k_\Gamma$, that $k_1 \leq k_l, \forall l \in \{2, \cdots, \Gamma - 1\}$, and that an edge exists between node k_l and node k_{l+1}, $\forall l \in \{1, \cdots, \Gamma - 1\}$. If a sequence of integers satisfy the above requirements, it is referred to as a legal sequence, and can be mapped to an actual random walk.

Due to the fact that a segment of a legal sequence may also be a legal sequence, it is possible to extract multiple legal sequences from a single simulated random walk, and use them also as random walks in the evaluation of equation (3.4) or its placement, (6.6) and (6.8). However, there are rules that one must comply with when extracting these legal sequences. A fundamental premise is that random samples must be independent of each other. If two walks share a segment, they become correlated. Note that if two walks have different starting nodes, they never participate in the same equation (6.6) or (6.8), and hence are allowed to share segments; if two walks have the same starting nodes, however, they are prohibited from overlapping. Moreover, due to the technique in the previous section, any one-step walk should be ignored.

[11] Recall that we only need diagonal entries of the matrix Z.

(a) $\{2, 4, 6, 4, 5, 7, 6, 3, 2, 5, 8, 1\}$
(b) $\{4, 6, 4, 5, 7, 6, 3\}$ $\{5, 7, 6, 3\}$ $\{5, 8, 1\}$

FIG. 6.1. *An example of (a) the legal sequence of a simulated random walk and (b) three extra walks extracted from it.*

Figure 6.1 shows an example of extracting multiple legal sequences from a single simulated random walk. The sequence $\{2, 5, 8, 1\}$ cannot be used because it has the same starting node as the entire sequence; the sequence $\{4, 5, 7, 6, 3\}$ cannot be used because it has the same starting node as $\{4, 6, 4, 5, 7, 6, 3\}$ and the two sequences overlap[12]. On the other hand, $\{5, 7, 6, 3\}$ and $\{5, 8, 1\}$ are both extracted because they do not overlap and hence are two independent random walks.

Considering all of the above requirements, the procedure for an R-matrix is shown in Algorithm 3, where the extracted legal sequences are directly accounted for in the M, H and J accumulators, which are defined as in Section 3.1. Note that the simulated random walk is never stored in memory, and the only extra storage due to this technique is the stacks, which contain a monotonically increasing sequence of integers at any moment. For a random walk game with scaling for a non-R-matrix, more storage is needed to keep track of products of scaling factors, and the increments of H and J variables should be the proper products of scaling factors instead of ones.

ALGORITHM 3. *Extract multiple walks from a single simulation, for an R-matrix:*

```
stack1.push( k_1 );
stack2.push( 1 );
For l = 2, 3, ···, until the end of walk, do {
   While( k_l < stack1.top() ){
      If( l > stack2.top()+1 ){
         k' = stack1.top();
         M_{k'} = M_{k'} + 1;
         H_{k',k_l} = H_{k',k_l} + 1;
         J_{k',k'} = J_{k',k'} + 1;
      }
      stack1.pop();
      stack2.pop();
   }
   If( k_l > stack1.top() ){
      stack1.push( k_l );
      stack2.push( l );
   }
   else J_{k_l,k_l} = J_{k_l,k_l} + 1;
}
```

This technique reduces the preconditioning runtime by fully utilizing the information contained in each simulated walk, such that it contributes to equations (6.6) and (6.8) as multiple walks. It guarantees that no two overlapping walks have the same starting node, and hence does not hurt the accuracy of the produced precondi-

[12]It is also legitimate to extract $\{4, 5, 7, 6, 3\}$ instead of $\{4, 6, 4, 5, 7, 6, 3\}$. However, the premise of random sampling must be fulfilled: the decision of whether to start a sequence with $k_2 = 4$ must be made without the knowledge of numbers after k_2, and the decision of whether to start a sequence with $k_4 = 4$ must be made without the knowledge of numbers after k_4. The strategy in Algorithm 3 is to start a sequence as early as possible, and hence produces $\{4, 6, 4, 5, 7, 6, 3\}$ instead of $\{4, 5, 7, 6, 3\}$.

tioner. The only cost of this technique is that the node ordering must be determined beforehand, and hence pivoting is not allowed during the incomplete factorization[13].

6.4. Matrix Ordering. In existing factorization-based preconditioning techniques, matrix ordering affects the performance. The same statement is true for the proposed stochastic preconditioner. In general, since we perform an incomplete $\mathrm{LDU/LDL^T}$ factorization on the reverse ordering of the matrix A, we can utilize any existing ordering method on A and then reverse the resulting ordering; in this way, any benefit of that ordering method can be inherited by us. For example, a reversed Approximate Minimum Degree ordering (AMD) [1] is likely to improve the accuracy-size tradeoff; a reversed Reverse Cuthill-McKee ordering (RCM) [9], which becomes the original Cuthill-McKee ordering, is likely to improve cache efficiency.

7. Numerical Results. Twelve symmetric benchmark matrices are used to compare the stochastic preconditioner against IC(0), ICT, MICT, and support-graph preconditioners; twelve asymmetric benchmark matrices are used to compare the stochastic preconditioner against ILU(0), ILUT, and MILUT preconditioners.

The first set of symmetric benchmarks are generated by SPARSKIT [29] by finite-difference discretization, with the regular seven-point stencil, of the 3D Laplace's equation $\nabla^2 u = 0$ with Dirichlet boundary condition. The matrices correspond to 3D grids with sizes 50-by-50-by-50, 60-by-60-by-60, up to 100-by-100-by-100, and a right-hand-side vector with all entries being 1 is used with each of them. They are listed in Tables 7.1, 7.3 and 7.4 as benchmarks m1 to m6. Another six application-specific benchmarks, m7 to m12, are reported in Tables 7.2, 7.3 and 7.4: they are symmetric placement matrices from VLSI designs, and are denser than the 3D-grid matrices. The twelve asymmetric benchmarks, m1' to m12' in Tables 7.5 and 7.6, are derived from m1 to m12 respectively: each of them is generated by randomly switching signs of the off-diagonal entries in a symmetric benchmark and randomly removing a fraction of its off-diagonal entries.

MATLAB is used to generate the IC(0), ICT, MICT, ILU(0), ILUT, and MILUT preconditioners. Three matrix ordering algorithms are available in MATLAB: minimum degree ordering (MMD) [12], AMD [1] and RCM [9]. AMD results in the best performance on the benchmarks and is used for all. TAUCS [33] is used to generate the support-graph preconditioners, which are based on the non-recursive version of the augmented maximum-weight-basis (AMWB) algorithm proposed in [5]. Our solver package [27] generates the stochastic preconditioners. The condition number measurements in Tables 7.1, 7.2 and 7.5 are all performed in MATLAB; the support-graph preconditioners and the stochastic preconditioners are read into MATLAB via binary files. The preconditioned CG and BCG solves in Tables 7.3 and 7.6 are all performed in MATLAB as well. The $T2$ runtimes in Table 7.4, which are not used in any comparison, are measured on CG solves by our solver package [27].

In Tables 7.1 and 7.2, and Figure 7.1, the condition number comparisons are based on roughly equal preconditioner sizes: the dropping thresholds of ICT and MICT are tuned, and the accuracy-size tradeoffs of the support-graph preconditioner as well as the proposed stochastic preconditioner are adjusted, such that the sizes of the factors produced by all four methods are similar, i.e., the S values in the tables are close.

A clear trend can be observed in Figure 7.1 that when the matrix size increases,

[13]For irreducibly diagonally dominant matrices, pivoting is not needed. For more general matrices to be discussed in Section 8, the usage of this technique may be limited. Note that this technique is optional, and without it, the advantages of the hybrid solver still hold.

TABLE 7.1

Condition number comparison of the stochastic preconditioner against ICT, MICT, and support-graph preconditioners on the symmetric 3D-grid benchmarks. N is the dimension of a matrix; E is the number of non-zero entries of a matrix; $C1$ is the condition number of the original matrix; S is the number of non-zero entries of the preconditioner (L and D matrices); $C2$ is the condition number after split preconditioning.

Matrix	N	E	$C1$	ICT		MICT		Support-graph		Stochastic	
				S	$C2$	S	$C2$	S	$C2$	S	$C2$
m1	1.25e5	8.60e5	1.05e3	1.72e6	19.4	1.72e6	30.3	1.73e6	623	1.71e6	4.72
m2	2.16e5	1.49e6	1.51e3	3.00e6	27.7	3.00e6	43.2	3.11e6	564	3.02e6	4.84
m3	3.43e5	2.37e6	2.04e3	4.80e6	37.4	4.80e6	58.4	4.79e6	587	4.87e6	5.13
m4	5.12e5	3.55e6	2.66e3	7.20e6	48.5	7.20e6	75.8	7.30e6	1.46e3	7.35e6	4.78
m5	7.29e5	5.05e6	3.36e3	1.03e7	61.0	1.03e7	95.2	1.03e7	866	1.06e7	5.05
m6	1.00e6	6.94e6	4.13e3	1.42e7	75.2	1.42e7	117	1.42e7	2.07e3	1.46e7	5.15

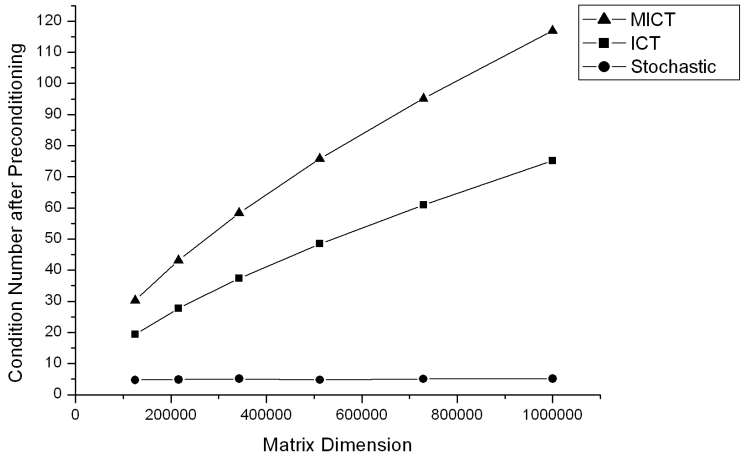

FIG. 7.1. *The condition number $C2$ after preconditioning as a function of the matrix dimension N for the results in Table 7.1.*

the performances of ICT and MICT both degrade, while the performance of the stochastic preconditioner remains relatively stable. This is consistent with our argument in Section 5.2: when the matrix is larger and denser, the effect of error accumulation in traditional methods becomes stronger, and the benefit of stochastic preconditioning becomes more prominent. The same explanation holds for the fact that the performance gap in Table 7.2 is bigger than in Table 7.1, and the reason is that the m7-m12 benchmarks are denser than m1-m6, i.e., m7-m12 have a higher average number of non-zeroes per row. In Table 7.2, the performances of ICT and MICT fluctuates significantly due to the different structural and numerical properties of the benchmarks m7-m12; again, the stochastic preconditioner remains stable.

In Table 7.3, we use a different metric to compare the stochastic preconditioning against the IC(0), ICT, MICT preconditioners, as well as the support-graph preconditioner at two different tradeoff points, one with a size similar to IC(0) and the other with a size similar to ICT. The complexity metric is the number of double-precision multiplications needed at the PCG solving stage for the equation set $A\mathbf{x} = \mathbf{b}$, in order to converge with an error tolerance of 10^{-6}, i.e., $\| \mathbf{b} - A\mathbf{x} \|_2 < 10^{-6} \cdot \| \mathbf{b} \|_2$. In Table 7.3, the I values are from MATLAB, and the M values are calculated as

TABLE 7.2

Condition number comparison of the stochastic preconditioner against ICT, MICT, and support-graph preconditioners on the symmetric VLSI placement benchmarks. N, E, $C1$, S and $C2$ are as defined in Table 7.1.

Matrix	N	E	$C1$	ICT		MICT		Support-graph		Stochastic	
				S	$C2$	S	$C2$	S	$C2$	S	$C2$
m7	7.28e4	1.10e6	1.96e5	1.28e6	743	1.28e6	1.06e3	1.26e6	1.60e3	1.28e6	6.66
m8	2.07e5	2.14e6	2.45e4	3.00e6	234	3.00e6	320	3.04e6	2.17e3	2.96e6	5.83
m9	2.67e5	3.19e6	1.13e5	4.06e6	278	4.06e6	315	4.08e6	1.20e4	4.05e6	6.25
m10	4.04e5	5.09e6	1.13e4	6.32e6	310	6.32e6	467	6.45e6	8.85e3	6.41e6	6.40
m11	4.39e5	8.06e6	5.73e4	8.07e6	181	8.07e6	211	8.07e6	1.02e4	8.09e6	6.77
m12	8.54e5	9.23e6	1.30e4	1.26e7	385	1.26e7	537	1.25e7	9.12e3	1.26e7	6.17

TABLE 7.3

Computational complexity comparison of CG using the stochastic preconditioner against using IC(0), ICT, MICT, and support-graph preconditioners, to solve the twelve symmetric benchmarks for one right-hand-side vector, with 10^{-6} error tolerance. N, E, S are as defined in Table 7.1; I is the number of iterations to reach 10^{-6} error tolerance; M is the total number of multiplications; R is our speedup ratio, measured by the corresponding M value divided by the M value of stochastic preconditioning.

Matrix		m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
N		1.35e5	2.16e5	3.43e5	5.12e5	7.29e5	1.00e6	7.28e4	2.07e5	2.67e5	4.04e5	4.39e5	8.54e5
E		8.60e5	1.49e6	2.37e6	3.55e6	5.05e6	6.94e6	1.10e6	2.14e6	3.19e6	5.09e6	8.06e6	9.23e6
Stochastic	S	1.71e6	3.02e6	4.87e6	7.35e6	1.06e7	1.46e7	1.28e6	2.96e6	4.05e6	6.41e6	8.09e6	1.26e7
	I	17	17	18	18	18	19	21	20	22	21	22	20
	M	8.14e7	1.43e8	2.43e8	3.65e8	5.25e8	7.63e8	8.30e7	1.78e8	2.72e8	4.10e8	5.72e8	7.57e8
IC(0)	S	4.93e5	8.53e5	1.36e6	2.03e6	2.89e6	3.97e6	5.87e5	1.17e6	1.73e6	2.75e6	4.25e6	5.04e6
	I	59	70	81	92	94	104	127	151	165	142	115	187
	M	1.38e8	2.84e8	5.23e8	8.88e8	1.29e9	1.96e9	3.26e8	8.02e8	1.27e9	1.73e9	2.11e9	4.25e9
	R	1.70	1.99	2.16	2.43	2.46	2.57	3.92	4.51	4.68	4.23	3.68	5.61
ICT	S	1.72e6	3.00e6	4.80e6	7.20e6	1.03e7	1.42e7	1.28e6	3.00e6	4.06e6	6.32e6	8.07e6	1.26e7
	I	21	25	29	32	36	40	60	72	88	71	56	100
	M	1.01e8	2.09e8	3.87e8	6.40e8	1.03e9	1.57e9	2.37e8	6.46e8	1.09e9	1.37e9	1.45e9	3.78e9
	R	1.24	1.47	1.59	1.75	1.96	2.06	2.85	3.63	4.01	3.35	2.54	4.99
MICT	S	1.72e6	3.00e6	4.80e6	7.20e6	1.03e7	1.42e7	1.28e6	3.00e6	4.06e6	6.32e6	8.07e6	1.26e7
	I	27	32	38	43	46	52	71	86	101	85	65	119
	M	1.30e8	2.68e8	5.07e8	8.60e8	1.31e9	2.04e9	2.80e8	7.72e8	1.25e9	1.65e9	1.69e9	4.50e9
	R	1.59	1.88	2.09	2.35	2.50	2.68	3.37	4.34	4.60	4.01	2.95	5.94
Support-graph #1	S	4.95e5	8.51e5	1.35e6	2.10e6	2.91e6	3.92e6	5.53e5	1.20e6	1.72e6	2.74e6	4.14e6	5.05e6
	I	315	349	491	492	584	628	274	438	870	678	999	1000
	M	7.40e8	1.42e9	3.16e9	4.82e9	8.05e9	1.18e10	6.84e8	2.35e9	6.69e9	8.27e9	1.81e10	2.27e10
	R	9.09	9.93	13.0	13.2	15.3	15.5	8.24	13.2	24.6	20.2	31.6	30.0
Support-graph #2	S	1.73e6	3.11e6	4.79e6	7.30e6	1.03e7	1.42e7	1.26e6	3.04e6	4.08e6	6.45e6	8.07e6	1.25e7
	I	195	194	205	319	251	387	200	315	763	545	720	633
	M	9.41e8	1.66e9	2.73e9	6.44e9	7.19e9	1.52e10	7.83e8	2.85e9	9.48e9	1.07e10	1.87e10	2.38e10
	R	11.6	11.7	11.3	17.6	13.7	20.0	9.42	16.0	34.9	26.1	32.7	31.4

$M = I \cdot (S \cdot 2 + E + N \cdot 4)$, which is the best possible implementation based on the PCG pseudo codes in [2], [28].

Again, Table 7.3 suggests that PCG with stochastic preconditioning requires the least amount of computation to reach the same convergence criterion, and that the speedup ratio R gets higher for larger and/or denser matrices.

The computational costs of building the proposed preconditioner are reported in Table 7.4. For each benchmark, the cost is shown in two forms: the amount of computation measured by the total number of random-walk steps, as well as the corresponding physical CPU runtime $T1$. A random-walk step is an element oper-

TABLE 7.4
Cost of the stochastic preconditioning, measured by the total number of random-walk steps performed, and by the physical preconditioning CPU times $T1$ of our implementation on a Linux workstation with 2.8GHz CPU frequency. $T2$ is the solving CPU time of our implementation with 10^{-6} error tolerance. The units for $T1$ and $T2$ are second.

Matrix	m1	m2	m3	m4	m5	m6	m7	m8	m9	m10	m11	m12
#step	3.67e7	6.86e7	1.16e8	1.83e8	2.74e8	3.91e8	2.26e7	4.89e7	6.80e7	9.11e7	1.14e8	1.99e8
$T1$	7.05	13.91	23.97	38.30	58.63	83.86	6.12	11.86	18.70	28.96	55.80	77.63
$T2$	2.38	5.64	10.71	17.68	28.56	41.04	2.04	5.29	8.34	14.45	22.80	31.10

ation in the proposed algorithm and involves one random number generation plus $O(\log(\text{degree}(i)))$ logic operations, where $\text{degree}(i)$ is the degree of the current node. The choice of random number generator (or quasirandom numbers as an alternative) is a separate issue, and our implementation uses one from [24, p. 279]. The physical solving runtime $T2$ is also included as a reference. Admittedly, the preconditioning runtime $T1$ is more than the typical runtime of a traditional incomplete factorization; however, it is not a large overhead, gets easily amortized over multiple re-solves, and is worthwhile given the speedup achieved in the solving stage, i.e., the lower $T2$. In Table 7.4, $T1$ is no more than three times $T2$; hence, given the potential $T2$ reduction suggested by Table 7.3, it is likely that for a relatively large matrix, our preconditioning would save overall runtime even for just a single right-hand-side vector.

TABLE 7.5
Condition number comparison of the stochastic preconditioner against ILUT and MILUT on the twelve asymmetric benchmarks. N, E, $C1$, $C2$ are as defined in Table 7.1; S' is the number of non-zero entries of the asymmetric preconditioner (L, D, U matrices).

Matrix		m1'	m2'	m3'	m4'	m5'	m6'	m7'	m8'	m9'	m10'	m11'	m12'
N		1.25e5	2.16e5	3.43e5	5.12e5	7.29e5	1.00e6	7.28e4	2.07e5	2.67e5	4.04e5	4.39e5	8.54e5
E		8.60e5	1.49e6	2.37e6	3.55e6	5.05e6	6.94e6	1.10e6	2.14e6	3.19e6	5.09e6	8.06e6	9.23e6
$C1$		552	632	713	785	743	856	8.47e3	1.93e3	1.85e4	1.46e3	2.78e3	1.39e3
Stochastic	S'	2.99e6	5.27e6	8.50e6	1.28e7	1.85e7	2.55e7	2.57e6	6.12e6	8.72e6	1.28e7	1.70e7	2.62e7
	$C2$	3.21	3.21	3.38	3.27	3.46	3.45	3.28	4.05	3.81	4.39	5.10	3.89
ILUT	S'	3.04e6	5.30e6	8.47e6	1.27e7	1.81e7	2.50e7	2.54e6	6.01e6	8.91e6	1.24e7	1.67e7	2.63e7
	$C2$	15.6	17.8	20.0	22.0	22.7	24.0	60.8	35.8	42.9	54.7	35.6	50.0
MILUT	S'	3.04e6	5.30e6	8.47e6	1.27e7	1.81e7	2.50e7	2.54e6	6.01e6	8.91e6	1.24e7	1.67e7	2.63e7
	$C2$	21.6	24.4	27.9	31.1	32.8	35.0	81.2	54.9	57.9	79.0	62.6	71.7

In Tables 7.5 and 7.6, the condition number and the computational complexity comparisons are repeated for the asymmetric benchmarks m1' to m12', and the stochastic preconditioning is compared against ILUT, MILUT, and ILU(0). The ILU(0) data points for m8' and m12' are unavailable due to MATLAB failures. In Table 7.6, based on the preconditioned BCG pseudo code in [2], the M formula is $M = I \cdot (S' \cdot 2 + E \cdot 2 + N \cdot 7)$. Again, Tables 7.5 and 7.6 suggest that the stochastic preconditioning results in the least condition numbers, and the least amount of computation to reach convergence, and that the advantages gets more prominent for larger and/or denser matrices.

A reference implementation of the stochastic preconditioning as well as the hybrid solver, is available to the public [27].

8. Future Work. So far it is required that, in the random walk game, every scaling factor s must have an absolute value of one. Therefore, the scaling in the game never changes the magnitude of a monetary transaction, and only changes its sign or phase. This section discusses implications of non-unitary scaling factors.

TABLE 7.6

Computational complexity comparison of BCG using the stochastic preconditioner against using ILU(0), ILUT, MILUT preconditioners, to solve the twelve asymmetric benchmarks for one right-hand-side vector, with 10^{-6} error tolerance. N, E are as defined in Table 7.1; S' is as defined in Table 7.5; I, M, R are as defined in Table 7.3.

Matrix		m1'	m2'	m3'	m4'	m5'	m6'	m7'	m8'	m9'	m10'	m11'	m12'
N		1.25e5	2.16e5	3.43e5	5.12e5	7.29e5	1.00e6	7.28e4	2.07e5	2.67e5	4.04e5	4.39e5	8.54e5
E		8.60e5	1.49e6	2.37e6	3.55e6	5.05e6	6.94e6	1.10e6	2.14e6	3.19e6	5.09e6	8.06e6	9.23e6
Stochastic	S'	2.99e6	5.27e6	8.50e6	1.28e7	1.85e7	2.55e7	2.57e6	6.12e6	8.72e6	1.28e7	1.70e7	2.62e7
	I	16	17	17	16	17	18	17	17	17	17	17	18
	M	1.37e8	2.56e8	4.11e8	5.82e8	8.87e8	1.29e9	1.34e8	3.05e8	4.37e8	6.57e8	9.04e8	1.38e9
ILU(0)	S'	8.60e5	1.49e6	2.37e6	3.55e6	5.05e6	6.94e6	1.10e6	N/A	3.19e6	5.09e6	8.06e6	N/A
	I	60	66	73	78	77	87	88	N/A	90	89	82	N/A
	M	2.59e8	4.93e8	8.68e8	1.39e9	1.95e9	3.02e9	4.32e8	N/A	1.32e9	2.06e9	2.90e9	N/A
	R	1.88	1.93	2.11	2.38	2.20	2.34	3.24	N/A	3.01	3.14	3.20	N/A
ILUT	S'	3.04e6	5.30e6	8.47e6	1.27e7	1.81e7	2.50e7	2.54e6	6.01e6	8.91e6	1.24e7	1.67e7	2.63e7
	I	25	27	29	31	32	34	45	47	46	43	38	54
	M	2.17e8	4.07e8	6.98e8	1.12e9	1.65e9	2.41e9	3.51e8	8.34e8	1.20e9	1.63e9	2.00e9	4.16e9
	R	1.58	1.60	1.70	1.92	1.86	1.86	2.63	2.73	2.75	2.48	2.21	3.01
MILUT	S'	3.04e6	5.30e6	8.47e6	1.27e7	1.81e7	2.50e7	2.54e6	6.01e6	8.91e6	1.24e7	1.67e7	2.63e7
	I	33	34	38	39	42	45	55	57	59	54	46	67
	M	2.86e8	5.13e8	9.15e8	1.41e9	2.16e9	3.19e9	4.29e8	1.01e9	1.54e9	2.05e9	2.42e9	5.16e9
	R	2.09	2.01	2.23	2.42	2.44	2.46	3.21	3.31	3.52	3.12	2.68	3.73

If the scaling factors are allowed to take arbitrary complex values, the allowable matrix A in (4.10) becomes any matrix such that the diagonal entries are non-zero. In other words, if a matrix A has non-zero diagonal entries, a random walk game exists such that the f values, if they uniquely exist, satisfy a set of linear equations where the matrix is A. However, if there exist scaling factors with absolute values over 1, numerical problems may potentially occur since the product of scaling factors may be unbounded. How to quantify this effect and to analyze the corresponding convergence rate, is an open question for future research.

Acknowledgments. The authors would like to thank Sani R. Nassif for his contribution to the stochastic solver, Yousef Saad for helpful discussions, and Howard Elman and the two reviewers for constructive suggestions in revising this manuscript.

REFERENCES

[1] P. R. Amestoy, T. A. Davis and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886-905, 1996.

[2] R. Barrett, M. Berry, T. F. Chan, J. W. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. A. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.

[3] M. Benzi and M. Tuma, "A sparse approximate inverse preconditioner for nonsymmetric linear systems," *SIAM Journal on Scientific Computing*, vol. 19, no. 3, pp. 968-994, 1998.

[4] M. Bollhofer and Y. Saad, "On the relations between ILUs and factored approximate inverses," *SIAM Journal on Matrix Analysis and Applications*, vol. 24, no. 1, pp. 219-237, 2002.

[5] E. Boman, D. Chen, B. Hendrickson and S. Toledo, "Maximum-weight-basis preconditioners," *Numerical Linear Algebra and Applications*, vol. 11, pp. 695-721, 2004.

[6] T. C. Chan and H. A. van der Vorst, "Approximate and incomplete factorizations," Technical Report, Department of Mathematics, University of Utrecht, The Netherlands, 1994.

[7] E. Chow and Y. Saad, "Experimental study of ILU preconditioners for indefinite matrices," *Journal of Computational and Applied Mathematics*, vol. 86, no. 2, pp. 387-414, 1997.

[8] J. H. Curtiss, "Sampling methods applied to differential and difference equations," *Proceedings of IBM Seminar on Scientific Computation*, pp. 87-109, 1949.

[9] E. Cuthill and J. McKee, "Reducing the bandwidth of sparse symmetric matrices," *Proceedings of the ACM National Conference*, pp. 157-172, 1969.

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

[11] G. E. Forsythe and R. A. Leibler, "Matrix inversion by a Monte Carlo method," *Mathematical Tables and Other Aids to Computation*, vol. 4, no. 31, pp. 127-129, 1950.

[12] A. George and J. W. H. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Review*, vol. 31, no. 1, pp. 1-19, 1989.

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

[14] J. H. Halton, "Sequential Monte Carlo," *Proceedings of the Cambridge Philosophical Society*, vol. 58, pp. 57-78, 1962.

[15] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*, Methuen & Co. Ltd., London, UK, 1964.

[16] P. Heggernes, S. C. Eisenstat, G. Kumfert and A. Pothen, "The computational complexity of the Minimum Degree algorithm," *Proceedings of 14th Norwegian Computer Science Conference*, pp. 98-109, 2001.

[17] R. Hersh and R. J. Griego, "Brownian motion and potential theory," *Scientific American*, vol. 220, pp. 67-74, 1969.

[18] A. Joshi, *Topics in Optimization and Sparse Linear Systems*, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 1997.

[19] D. S. Kershaw, "The incomplete cholesky-conjugate gradient method for the iterative solution of systems of linear equations," *Journal of Computational Physics*, vol. 26, pp. 43-65, 1978.

[20] C. N. Klahr, "A Monte Carlo method for the solution of elliptic partial differential equations," in *Mathematical Methods for Digital Computers*, chap. 14, John Wiley and Sons, New York, NY, 1962.

[21] A. W. Knapp, "Connection between Brownian motion and potential theory," *Journal of Mathematical Analysis and Application*, vol. 12, pp. 328-349, 1965.

[22] A. W. Marshall, "The use of multi-stage sampling schemes in Monte Carlo," *Symposium of Monte Carlo Methods*, pp. 123-140, John Wiley & Sons, New York, NY, 1956.

[23] M. E. Muller, "Some continuous Monte Carlo methods for the Dirichlet problem," *Annals of Mathematical Statistics*, vol. 27, pp. 569-589, 1956.

[24] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes in C: the Art of Scientific Computing*, second edition, Cambridge University Press, New York, NY, 1994.

[25] H. Qian, S. R. Nassif and S. S. Sapatnekar, "Random walks in a supply network," *Proceedings of the ACM/IEEE Design Automation Conference*, pp. 93-98, 2003.

[26] H. Qian and S. S. Sapatnekar, "A hybrid linear equation solver and its application in quadratic placement," *ACM/IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 905-909, 2005.

[27] H. Qian and S. S. Sapatnekar, The Hybrid Linear Equation Solver Release. Available at `http://mountains.ece.umn.edu/~sachin/hybridsolver`

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

[29] Y. Saad, SPARSKIT, version 2. Available at `http://www-users.cs.umn.edu/~saad/software/SPARSKIT/sparskit.html`

[30] D. Spielman and S. H. Teng, "Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems," available at `http://www.arxiv.org/abs/cs.NA/0607105`, 2006.

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

[32] C. J. K. Tan and M. F. Dixon, "Antithetic Monte Carlo linear solver," *Proceedings of International Conference on Computational Science*, pp. 383-392, 2002.

[33] S. Toledo, TAUCS, version 2.2. Available at `http://www.tau.ac.il/~stoledo/taucs`

[34] W. Wasow, "A note on the inversion of matrices by random walks," *Mathematical Tables and Other Aids to Computation*, vol. 6, no. 38, pp. 78-81, 1952.

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