Preconditioning methods based on spanning tree algorithms

D V Perevozkin¹,² and G A Omarova¹,²
¹Institute of computational mathematics and mathematical geophysics SB RAS, 6,Ac.
Lavrentieva ave., Novosibirsk, 630090, Russia
²Novosibirsk State University, 1, Pirogova str., Novosibirsk, 630090, Russia
E-mail: foxillys@gmail.com, gulzira@rav.sscc.ru

Abstract. The work continues the research performed by Chen and Toledo as a follow-up to
Vaidya’s talk. The preconditioners being discussed are based on spanning tree algorithms where
the input is the graph arising from the matrix of the system of linear algebraic equations (SLAE).
The methods are considered in application to solving various SLAEs with real symmetric and
non-symmetric matrices. The research shows that the application area of the preconditioners
in question may significantly differ from the domain indicated in the literature.

1. Introduction
This article is dedicated to the implementation and experimental study of preconditioners that
are built using graph theory, namely preconditioners that are based on spanning trees. The
very first works on the subject were published in the late 80s — early 90s [1, 2, 3, 4, 5]. These
ideas were extended by Vaidya [6] who described his work in a talk but did not publish a
paper. Later, in the 2000s, Vaidya’s algorithm was implemented and experimentally studied in
[7]. It was shown that these preconditioners exhibit some good properties for a narrow class
of matrices. The authors claim that for symmetric positive definite and diagonally dominant
(SPD-DDD) matrices computational cost and convergence rate depend only on the matrix structure
and not its coefficients. They also performed several comparisons with other preconditioners, in
particular incomplete Cholesky factorization. The primary conclusions that were obtained are
the following:

• Vaidya’s preconditioners exhibit favourable convergence rate and their behaviour is very
different from that of preconditioners based on incomplete Cholesky (IC) factorization;
• for diagonally dominant SPD matrices, these preconditioners are only sensitive to matrix
portraits, not coefficients, which is again different from IC;
• for a large variety of 2D problems, their performance is better or comparable to that of IC;
• for some 3D problems, they outperform IC.

In [8] some of the properties were proven for a narrow class of matrices.

The objective of this paper is to study the behaviour of Vaidya’s preconditioners on SLAEs
arising from different real-world applications, including SLAEs with non-symmetric matrices.
For the latter, a modification of the method described in [7] is proposed.
The work is organized as follows. The first section briefly describes the algorithm presented in [7]. The second section considers the authors’ interpretation of the original ideas. The third section is devoted to the numerical experiments. Finally, a conclusion is drawn from the observations and a connection to the original research is made.

2. Description of the original algorithm

Let \( A = \{a_{i,j}\}_{i,j=1}^n \) be an \( n \times n \) SPDDD (symmetric positive definite and diagonally dominant) matrix. Based on \( A \), we build graph \( G_A \) as follows. \( G_A = (V_A, E_A) \), where \( V_A = \{1, \ldots, n\} \) is the set of graph vertices and \( E_A = \{(i, j), \ i \neq j, \ a_{i,j} \neq 0\} \) is the set of weighted edges, with the weight of edge \((i, j)\) being equal to \( |a_{i,j}| \). Since \( A \) is symmetric, \( G \) is undirected.

At the first step, a maximum spanning tree \( T \) is built for \( G_A \) using Prim’s algorithm [9].

Given a parameter \( t \), a recursive procedure splits the obtained tree \( T \) into \( k \) subtrees \( T_1, \ldots, T_k \), where \( T_i = (V_i, E_i), \ i = 1, \ldots, k \). The splitting is performed in such a manner that the number of vertices in each subtree is bound between \( n/t \) and \( dn/t + 1 \) where \( d \) is the maximum number of children that vertices in \( T \) have. Ideally, it is desirable to split the tree into subtrees with nearly the same number of vertices. In practice, however, it is not always obtainable.

The next step involves building graph \( G_M \) that contains all edges of \( T \) plus the heaviest edges between \( V_i \) and \( V_j \) for all \( i \) and \( j \):

\[
G_M = T \cup \{(v_i, v_j)|v_i \in V_i, \ v_j \in V_j, (v_i, v_j) \to \max\}. \tag{1}
\]

This graph serves as a basis for building the preconditioning matrix \( M = \{m_{i,j}\}_{i,j=1}^n \). For each edge \((i, j)\) in \( G_M \), there is a corresponding nonzero element \( m_{i,j} \) equal to \( a_{i,j} \). All other elements in \( M \) are zero, except for the diagonal, which is reconstructed so that \( Ae = Me \) holds, where \( e = (1, \ldots, 1)^T \).

3. Description of the modified algorithm

The following section summarizes the authors’ modification of the ideas mentioned in [6, 7].

The first step is the same except for the fact that the matrix \( A \) is no longer supposed to be symmetric, which implies that \( G_A \) is directed. The algorithms used to generate spanning trees rely on the fact that their input is undirected graphs, so \( G_A \) requires special treatment.

Perhaps the simplest approach to this problem is to consider an undirected graph \( G'_A \) based on \( G_A \). Let us build \( G'_A \) by modifying the weights of the edges in the following way:

\[
w(i, j) = \max\{|a_{i,j}|, |a_{j,i}|\}. \tag{2}
\]

We use Kruskal’s algorithm [10] for the generation of the maximum spanning tree \( T \). In contrast with Prim’s algorithm, Kruskal’s algorithm doesn’t require the adjacency information for vertices and may operate using only the edge list. Another advantage is that for each edge it is only required to have either \((i, j)\) or \((j, i)\) in the list. The complexity of the algorithm is \( O(E \log E) \).

The next step is dividing the obtained tree into subtrees. For this purpose, we employ a modified version of breadth-first search (BFS) [11]. The algorithm starts at an arbitrary vertex in \( T \). The search proceeds on vertices that are not yet marked until it either covers \( n/t \) vertices or the queue becomes empty. All vertices traversed are marked with a distinct colour. The algorithm then initiates another search starting with the first unmarked vertex. As a result of this operation, \( T \) is painted into a number of different colours. Ideally, there will be \( t \) colours, but a greater number is likely.

In contrast with [7], we proceed with adding every edge between \( V_i \) and \( V_j \) that exists in \( G'_A \):
\[ G_M = T \cup \{(v_i, v_j) | v_i \in V_i, v_j \in V_j, (v_i, v_j) \in G'_A\} . \] (3)

4. Numerical experiments

For the experiments, we chose several SLAEs from the well-known SuiteSparse Matrix Collection (formerly the University of Florida Sparse Matrix Collection) [12]. Only problems with right-hand sides were considered.

In the following, Vaidya’s preconditioner was used with FGMRes [13]. The latter was allowed to run for at most 1000 iterations with restarts placed every 100 iterations. The stopping criterion was \[ \|Ax_i - b\|_2 < 10^{-7}\|b\|_2 \] where \( x_i \) is the current approximate solution.

We considered running Vaidya’s preconditioner with \( t = 10, 100, 1000, 10000 \), which is denoted by V10, V100, V1000 and V10000 respectively. The inversion of matrix \( M \) was performed using direct solver PARDISO from the Intel(R) MKL library. All experiments were run on a server with two Intel E5-2697A processors and 256 GB of memory.

In table 1 we assess Vaidya’s preconditioner convergence in comparison with ILU0 and with regard to matrix type. The leftmost column is the problem name, the second column is the matrix type. There are three matrix types distinguished: symmetric positive definite (SPD), symmetric indefinite (SI), and non-symmetric (NS). The rest of the cells present the number of iterations. A dash indicates that the combination of the methods ran into NaN and broke down.

| Table 1. Iteration counts |
|---------------------------|
| SLAE          | type | V10 | V100 | V1000 | V10000 | ILU0 |
|-----------------|------|-----|------|-------|-------|------|
| af_0_k101       | SPD  | 1000| 1000 | 1000  | 1000  | 1000 |
| bundle_adj      | SPD  | 1000| 1000 | 1000  | 1000  | 1000 |
| nasa4704        | SPD  | 1000| 1000 | 1000  | 1000  | 1000 |
| offshore        | SPD  | 1000| 1000 | 1000  | 1000  | 1000 |
| olafu           | SPD  | 1000| 1000 | 1000  | 1000  | 1000 |
| parabolic_fem   | SPD  | 1000| 602  | 54    | 17    | 1000 |
| Pres_Poisson    | SPD  | 1000| 1000 | 1000  | 1000  | 1000 |
| smt             | SPD  | 1000| 1000 | 1000  | 1000  | 1000 |
| thermal1        | SPD  | 469 | 157  | 54    | 17    | 1000 |
| boyd1           | SI   | 10  | 3    | 2     | 1     | 16   |
| c-73            | SI   | 1000| 1000 | 69    | 16    | 1000 |
| gsm_106857      | SI   | 1000| 1000 | 1000  | 1000  | 1000 |
| OPF_6000        | SI   | 1000| 1000 | 1000  | 49    | –    |
| sherman1        | SI   | 21  | 12   | 1     | 1     | 44   |
| bbnat           | NS   | 1000| 1000 | 1000  | 1000  | 1000 |
| dc1             | NS   | 113 | 113  | 113   | 113   | 87   |
| hcircuit        | NS   | 13  | 12   | 12    | 10    | –    |
| jan99jac120     | NS   | 17  | 17   | 17    | 17    | –    |
| laminar_duct3D  | NS   | 1000| 1000 | 1000  | 1000  | 1000 |
| poisson3Db      | NS   | 1000| 1000 | 1000  | 69    | 95   |
| venkat01        | NS   | 1000| 1000 | 1000  | 1000  | 15   |
| viscoplastic2   | NS   | 1000| 1000 | 1000  | 1     | 863  |
| wang3           | NS   | 153 | 59   | 19    | 1     | 61   |

For the cases where Vaidya’s preconditioner showed convergence, we also compared its performance with that of ILU0 and PARDISO. In table 2, the total solution time is given for each SLAE. For the sake of convenience, the cases where the method failed to converge within 1000 iterations are marked with a dash.
Table 2. Solution time

| SLAE       | type   | V10 | V100 | V1000 | V10000 | ILU0 | PARDISO |
|------------|--------|-----|------|-------|--------|------|---------|
| nasa4704   | SPD    | –   | –    | 0.058 | –      | 0.040|
| olafu      | SPD    | –   | –    | 0.461 | –      | 0.251|
| parabolic_fem | SPD | –   | 14.054 | 6.051 | 4.314 | –   | 2.593   |
| Pres_Poisson | SPD  | –   | –    | 0.325 | 0.849  | 0.159|
| thermal1   | SPD    | 1.641 | 0.815 | 0.565 | 0.542  | –   | 0.405   |
| boyd1      | SI     | 12.169 | 9.064 | 9.188 | 9.244  | 16.838| 8.894   |
| c-73       | SI     | –   | –    | 2.910 | 1.586  | –   | 0.949   |
| OPF_6000   | SI     | –   | –    | 0.283 | –      | 0.116|
| sherman1   | SI     | 0.080 | 0.018 | 0.019 | 0.021  | 0.013| 0.017   |
| dc1        | NS     | 2.655 | 2.655 | 2.690 | 2.664  | 2.596| 1.480   |
| hcircuit   | NS     | 0.422 | 0.428 | 0.448 | 0.431  | –   | 0.309   |
| jan99jac120 | NS   | 0.578 | 0.584 | 0.586 | 0.594  | –   | 0.527   |
| laminar_duct3D | NS | 39.342 | 39.448 | 37.678 | 43.840 | 6.783| 52.523  |
| poisson3Db | NS     | –   | –    | 3.892 | 1.480  | 1.664|
| viscoplastic2 | NS   | –   | –    | 0.582 | 1.135  | 0.497|
| wang3      | NS     | 0.308 | 0.215 | 0.215 | 0.200  | 0.071| 0.187   |

5. Conclusion

The original research [7] claims that the algorithm is beneficial for solving 2D and 3D elliptic PDEs discretized on regular meshes. Vaidya’s preconditioner is compared to IC and MIC (modified incomplete Cholesky). All methods are used with preconditioned CG against the problems mentioned above.

Probably the most significant difference between the original preconditioner and the modification is the step that reintroduces edges between the subtrees back into the graph. However, it should be noted that adding more edges should not decrease the convergence rate. As an example, one may consider an extreme case of adding all available edges back to the graph, yielding a preconditioner equal to $A^{-1}$.

In our observations, Vaidya’s preconditioner exhibits mixed results when used against SLAEs with SPD matrices. Surprisingly, it shows reasonable convergence rates and performance on some problems with non-symmetric and symmetric indefinite matrices. Given that, we may conclude that Vaidya’s preconditioners may be suitable for a broader class of matrices than is typically considered in the literature. Matrix properties that determine whether the method will converge seem to be something other than symmetricity and diagonal dominance. Another promising but yet unused feature is that the method produces tree-like structures that may be handled better by a dedicated solver rather than by a general-purpose one.

References
[1] Beauwens R 1984 Linear algebra and its applications 62 87–104
[2] Beauwens R 1994 Numerical linear algebra with applications 1 1–37
[3] Notay Y 1990 Preconditioned Conjugate Gradient Methods (Springer) pp 105–125
[4] Notay Y 1991 Linear algebra and its applications 154 711–722
[5] Notay Y 1992 Applied numerical mathematics 10 381–396
[6] Vaidya P 1991 Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners
[7] Chen D and Toledo S 2003 Electron. Trans. Numer. Anal. 16 30–49
[8] Bern M, Gilbert J R, Hendrickson B, Nguyen N and Toledo S 2006 SIAM Journal on Matrix Analysis and Applications 27 930–951
[9] Prim R C 1957 The Bell System Technical Journal 36 1389–1401
[10] Kruskal J B 1956 *Proceedings of the American Mathematical society* **7** 48–50
[11] Cormen T H, Leiserson C E, Rivest R L and Stein C 2009 *Introduction to algorithms* (MIT press)
[12] Davis T, Hu Y and Kolodziej S Suitesparse matrix collection URL [https://sparse.tamu.edu/](https://sparse.tamu.edu/)
[13] Saad Y 2003 *Iterative methods for sparse linear systems* (SIAM)