Iranian Journal of Numerical Analysis and Optimization
Vol. 9, No. 2, (2019), pp 1–16
DOI:10.22067/ijnao.v9i2.70454

Research Article

A new approximate inverse preconditioner based on the Vaidya's maximum spanning tree for matrix equation $AXB = C$

K. Rezaei, F. Rahbarnia* and F. Toutounian

Abstract

We propose a new preconditioned global conjugate gradient (PGL-CG) method for the solution of matrix equation $AXB = C$, where A and B are sparse Stieltjes matrices. The preconditioner is based on the support graph preconditioners. By using Vaidya's maximum spanning tree preconditioner and BFS algorithm, we present a new algorithm for computing the approximate inverse preconditioners for matrices A and B and constructing a preconditioner for the matrix equation $AXB = C$. This preconditioner does not require solving any linear systems and is highly parallelizable. Numerical experiments are given to show the efficiency of the new algorithm on CPU and GPU for the solution of large sparse matrix equation.

AMS(2010): 65F10.

Keywords: Krylov subspace methods; matrix equation; approximate inverse preconditioner; global conjugate gradient; support graph preconditioner; Vaidya's maximum spanning tree preconditioner.

---

*Corresponding author
Received 23 January 2018; revised 27 May 2018; accepted 13 August 2018
K. Rezaei
Department of Applied Mathematics, Faculty of Mathematical Sciences, Ferdowsi University of Mashhad, Iran e-mail: k.rezai1985@gmail.com

F. Rahbarnia
Department of Applied Mathematics, Faculty of Mathematical Sciences, Ferdowsi University of Mashhad, Iran. e-mail: Rahbarnia@ferdowsi.um.ac.ir

F. Toutounian
Department of Applied Mathematics, Faculty of Mathematical Sciences, Ferdowsi University of Mashhad, Iran.
The Center of Excellence on Modeling and Control Systems, Ferdowsi University of Mashhad, Iran. e-mail: toutouni@math.um.ac.ir

1 Introduction

The solution of linear systems of equations is at the heart of many computations in science, engineering, and other disciplines; see [2, 8–10] and their references. Hence, many researches have been performed on various types of matrix equations; for example, see [1, 8, 11, 16, 18, 19, 27, 28].

The principal goal of this paper is to use support graph preconditioning techniques to solve the matrix equation

$$AXB = C, \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{m\times m}$ are large sparse Stieltjes matrices. The linear matrix equation (1) can be written as the following $nm \times nm$ linear system:

$$(B^T \otimes A)vec(X) = vec(C), \tag{2}$$

where $vec(X)$ is the vector of $\mathbb{R}^{nm}$ obtained by stacking the columns of the $n \times m$ matrix X and $\otimes$ denotes the Kronecker product ($A \otimes B = [a_{ij}B]_{ij}$). The CG algorithm [24] can be used to solve the linear system (2). However, for large problems, this approach cannot be applied directly. In addition, the number of iterations of conjugate gradient method for the solution of linear system of equations $Ax = b$ is bounded by the square root of the spectral condition number $\kappa(A)$ of A. The condition number is the ratio of extreme eigenvalues of A, $\kappa(A) = \lambda_{max}(A)/\lambda_{min}(A)$. Preconditioner accelerates the convergence of iterative methods for solving linear systems.

In this paper, we use the preconditioned global conjugate gradient (PGL-CG) method for obtaining the approximate solution of matrix equation (1). The preconditioner is based on the support graph preconditioners. Predecessors of support-graph methods can be found in the work from the late 80s by Notay, Beauwens, and collaborators in which graph-theoretic notions (principally paths) are used in the analysis of preconditioners; see [3,4,21–23]. These insights were extended by Vaydia [26], who described his work in a talk in 1991 but did not publish a paper. Vaidya used support-graph techniques to design a family of preconditioners based on spanning trees in graphs. Later, Gremban, Miller, and Zagha [14, 15] extended the technique and used it to construct another family of preconditioners. In Section 3, we use Vaidya's maximum spanning tree preconditioners of matrices A and B for developing fast and efficient preconditioner to precondition equation (2).

Throughout this paper, all matrices are assumed to be real. For two matrices $X, Y \in \mathbb{R}^{n\times s}$, the inner product $\langle X, Y\rangle_F = (Y^T X)$ is used and the associated norm is the Frobenius norm denoted by $\|.\|_F$.

The rest of the paper is organized as follows. In the next section, we implement the preconditioned global CG method for solving matrix equation (1) and we introduce Vaidya's maximum spanning tree preconditioner. In section 3, we present a new algorithm for computing the inverse of this kind of preconditioners. In section 4, numerical examples are given to illustrate the efficiency of the proposed preconditioner. Conclusions are summarized in Section 5.

2 Preconditioned GL-CG method for solving the matrix equation $AXB = C$

In this section, we consider the matrix equation $AXB = C$, where A and B are symmetric and positive definite and assume that the preconditioners P_A and P_B are available. The preconditioners P_A and P_B are the matrices that approximate A and B in some sense, respectively. It is assumed that P_A and P_B are also symmetric positive definite. Then, we can precondition system (1) as follows:

$$(P_B \otimes P_A)^{-1}(B \otimes A)vec(X) = (P_B \otimes P_A)^{-1}vec(C), \tag{3}$$

where the preconditioner $(P_B \otimes P_A)$ is a symmetric positive definite matrix. In addition, from the fact that $\|A \otimes B\| = \|A\|\|B\|$ [17], we have

$$\text{cond}((P_B \otimes P_A)^{-1}(B \otimes A)) = \text{cond}(P_B^{-1}B)\text{cond}(P_A^{-1}A).$$

The straightforward application of PCG algorithm [24] to the linear system (3) yields the following preconditioned global CG algorithm for solving the matrix equation (1).

Algorithm 1 PGL-CG for solving AXB=C

1. Compute $R_0 = C - AX_0B, Z_0 = P_A^{-1}R_0P_B^{-1}$ and $P_0 = Z_0$
2. **for** $j = 0, 1, \ldots,$ until convergence **do**
3. $\quad \alpha_j = \frac{<R_j, Z_j>_F}{<AP_jB, P_j>_F}$
4. $\quad X_{j+1} = X_j + \alpha_j P_j$
5. $\quad R_{j+1} = R_j - \alpha_j AP_jB$
6. $\quad Z_{j+1} = P_A^{-1}R_{j+1}P_B^{-1}$
7. $\quad \beta_j = \frac{<R_{j+1}, Z_{j+1}>_F}{<R_j, Z_j>_F}$
8. $\quad P_{j+1} = Z_{j+1} + \beta_j P_j$
9. **end for**

We focus on applying Vaidya's preconditioner of the first class to the matrices A and B for constructing the preconditioners P_A and P_B. In order to explain Vaidya's preconditioner, we first present the following definition from [7].

Definition 1. The underlying graph $G_A = (V_A, E_A)$ of an n-by-n symmetric matrix A is a weighted undirected graph whose vertex set is $V_A = \{1, 2, \ldots, n\}$

and whose edges set is $E_A = \{(i,j) : i \neq j, a_{i,j} \neq 0\}$. The weight of an edge (i,j) is $-a_{i,j}$. The weight of a vertex i is the sum of elements in the row i of A.

Graph preconditioner, introduced by Vaidya [26] in the early nineties, uses maximum-weight spanning tree (MWST) preconditioners to bound the condition number of a preconditioned system. Vaidya's method constructs a preconditioner M whose underlying graph G_M is a subgraph of G_A (graph of A). The graph G_M of preconditioner has the same set of vertices as G_A and a subset of the edges of G_A. Vaidya proposed two classes of preconditioners. The first class of MWST preconditioners guarantees a condition number bound of $O(n^2)$ for any $n \times n$ sparse diagonally dominant symmetric (SDD) matrix; see [7]. The second class of preconditioners is based on MWST augmented with a few extra edges. This class of preconditioners guarantees that the work in the linear solver is bounded by $O(n^{1.75})$ for any sparse diagonally dominant matrix. In this paper, we focus on applying Vaidya's preconditioners of the first class to a subclass of SDD matrices, the class of SDD matrices with nonpositive off-diagonal elements (Stieltjes matrices).

In order to construct the MWST preconditioner P_A for A, we first construct the maximum-weight spanning tree T_A in G_A and then modify the diagonal elements of preconditioner P_A such that A and P_A have the same row sums. In other words, T_A is a connected graph with no cycles (i.e., a spanning tree), and the total weight of its edges is maximal among all spanning trees of G_A. The preconditioner P_A is a diagonally dominant Stieltjes matrix whose underlying graph is $G_{P_A} = T_A$, and whose row sums are identical to those of A.

When the condition number of the matrix $B \otimes A$ is high, it becomes necessary to develop a fast and efficient preconditioner for the iterative solution of (2). In order to precondition the system (2), we first construct Vaidya's preconditioners (maximum-weight spanning tree) P_A and P_B for A and B, respectively, and then we use $P_B \otimes P_A$ as a preconditioner for the matrix $B \otimes A$. The implementation of this preconditioner is based on computation of the inverse matrices P_A^{-1} and P_B^{-1}. In Section 3, we show that, by using the breadth first search (BFS) algorithm [25], we can easily compute these inverse matrices.

3 Computation of inverse of a MWST preconditioner

Let M be a symmetric positive definite matrix whose underlying graph T_M is a tree. In order to compute the inverse of M, we need the following definition.

Definition 2. The elimination matrix $L_{pq}(-\alpha) \in \mathbb{R}^{n\times n}$ with $p \neq q$, is an identity matrix with one nonzero off-diagonal entry in the row p and the column q. Therefore the entries of $L_{pq}(-\alpha)$ are as follows:

$$(L_{pq}(-\alpha))_{(i,j)} = \begin{cases} 1 & \text{if } i = j, \\ -\alpha & \text{if } (i,j) = (p,q), \\ 0 & otherwise. \end{cases}$$

Now we investigate the result of symmetric transformation

$$\overline{M} = L_{pq}(-\alpha) M L_{pq}^T(-\alpha). \tag{4}$$

Now $L_{pq}(-\alpha)M$ changes only the row p of M, while $ML_{pq}^T(-\alpha)$ changes only the column p. Thus, multiplying out equation (4) and using the symmetry of M, we get the explicit formulas

$$\begin{array}{ll} \bar{m}_{pj} = \bar{m}_{jp} = m_{pj} - \alpha m_{qj}, & j \neq p, \\ \bar{m}_{pp} = m_{pp} - 2\alpha m_{pq} + \alpha^2 m_{qq}, & \\ \bar{m}_{ij} = \bar{m}_{ji} = m_{ij} & \text{otherwise.} \end{array} \tag{5}$$

The idea of our method is to try to zero the off-diagonal elements of M by a series of transformations (4) and using the leaves of graph T_M and its subtrees.

Let us assume that the vertex q is a leaf in T_M and its neighbor is the vertex p. In order to zero the off-diagonal m_{pq}, accordingly, to set $\bar{m}_{pq} = 0$, equation (5) gives the following expression for the parameter α :

$$\alpha = \frac{m_{pq}}{m_{qq}}. \tag{6}$$

From (5) and (6), the entries of $\overline{M} = L_{pq}(-\alpha) M L_{pq}^T(-\alpha)$ are as follows:

$$\begin{array}{ll} \bar{m}_{pp} = m_{pp} - 2\alpha m_{pq} + \alpha^2 m_{qq}, & \\ \bar{m}_{pq} = \bar{m}_{qp} = 0, & \\ \bar{m}_{ij} = m_{ij} & \text{otherwise.} \end{array}$$

This process which eliminates the nonzero entries m_{pq} and m_{qp} of M, is equivalent to eliminate the edge (p,q) from the tree T_M. If $\widetilde{M}$ denotes the matrix obtained from the matrix $\overline{M}$ by removing the row q and the column q, then it is trivial that the underlying graph of this submatrix is the induced subtree of T_M on $V(T_M) - \{q\}$. Let $M_1 = M, p_1 = p, q_1 = q, \alpha_1 = \alpha, \overline{M}_1 = L_{pq}(-\alpha) M L_{pq}(-\alpha)^T$, and $M_2 = \widetilde{M}$; then, we can successively transform M to diagonal form by means of transformations of the type (4) in $(n-1)$ steps with the elimination matrices $L_{p_j,q_j}(-\alpha_j)$ and $\alpha_j = (m_{p_j,q_j}/m_{q_j,q_j}), j = 1, 2, \ldots, n-1$, which are defined by choosing the edges $(p_j, q_j), j = 1, 2, \ldots, n-1$ such that the vertex q_j is a leaf in the subtree T_{M_j}. To achieve this, we need to apply the BFS algorithm (Algorithm 2) to the maximum-weight spanning tree T_M to obtain the vector $V = [j_1, j_2, \ldots, j_n]$, which represents an array of vertices that are traversed and sorted by the BFS algorithm and $Level(u_j), j = 1, 2, \ldots, n$, which represent the level of traversed vertices $u_j, j = 1, 2, \ldots, n$ in the BFS tree.

By using the array of vertices $V = [j_1, j_2, \ldots, j_n]$, we can diagonalize the matrix M in $n-1$ steps. In step k, $k = 1, 2, \ldots, n-1$, by choosing the vertex $q_k = j_{n-k+1}$ from V and considering its parent $p_k = i_{n-k+1}$ and the edge (p_k, q_k), we define the elimination matrix $L_{p_k,q_k}(-\alpha_k)$ for eliminating the off-diagonal element m_{p_k,q_k}. In Lemma 1, we show that the vertex $q_k = j_{n-k+1}$ at step k is a leaf in the subtree T_{M_k}. In what follows, we show that by using Level (u_j), $j = 1, 2, \ldots, n$, we can reduce the overall time of producing the inverse of M.

Algorithm 2 Breadth first search algorithm as BFS(G,s)

```
V = ∅
for each vertex u ∈ V(G) − s do
  state(u) = "undiscovered"
  p(u) = nil, i.e. no parent is in the BFS tree
end for
state(s) = "discovered"
V = V ∪ {s}
Level(s) = 0
p(s) = nil
Q = {s}
while Q ≠ ∅ do
  u = dequeue(Q)
  process vertex u as desired
  for each v ∈ Adj(u) do
    process edge (u, v) as desired
    if state(v) = "undiscovered" then
      state(v) = "discovered"
      V = V ∪ {v}
      p(v) = u
      Level(v) = Level(p(v)) + 1
      enqueue(Q, v)
    end if
    state(u) = "processed"
  end for
end while
```

Lemma 1. *Let $V = \{j_1, j_2, \ldots, j_n\}$ be the set of vertices obtained by the BFS algorithm. If we isolate the vertex j_n and $T_M^{(j_n)}$ denotes the induced subtree of T_M on the vertex set $V(T_M) - \{j_n\}$, then j_{n-1} is a leaf in the induced subtree $T_M^{(j_n)}$.*

Proof. Let i_n be the parent of j_n and $Level(j_n) = l$. According to the BFS algorithm, if $s < t$, then $Level(j_s) \leqslant Level(j_t)$. Suppose that we isolate the vertex j_n; then we must consider the $Level(j_{n-1})$. If $Level(j_{n-1}) = l$, then it is trivial that the vertex j_{n-1} is a leaf in the induced subtree $T_M^{(j_n)}$. If $Level(j_{n-1}) = l-1$, then it means that there is no vertex in level l, so the vertex j_{n-1} has no children, according to the BFS algorithm, and it is a leaf in the subtree $T_M^{(j_n)}$. □

Let M be Vaidya's maximum-weight spanning tree preconditioner for the diagonally dominant spd matrix A and let $V = \{j_1, j_2, \dots, j_n\}$ be the array obtained by the BFS algorithm. We observe that we can diagonalize M by $n-1$ elimination matrices $L_{p_k,q_k}(-\alpha_k)$, $k = 1, 2, \dots, n-1$, where $q_k = j_{n-k+1}$. So, the elimination process yields the diagonal matrix D as follows:

$$D = L_{n-1}L_{n-2}\dots L_1 M L_1^T \dots L_{n-2}^T L_{n-1}^T,$$

where $L_k = L_{p_k,q_k}(-\alpha_k), k = 1, 2, \dots, n-1$. Therefore, we have

$$M^{-1} = (L_{n-1}L_{n-2}\dots L_1)^T D^{-1}(L_{n-1}L_{n-2}\dots L_1).$$

In addition, by supposing that level$(j_n) = l$, we can write

$$M^{-1} = (G_1G_2\dots G_l)^T D^{-1}(G_1G_2\dots G_l),$$

where $G_k = L_{\nu_k+s_k-1}\dots L_{\nu_k}$ for $k = 1, \dots, l$, and s_k represents the number of vertices that have the level k in the graph T_M, and the matrices $L_{\nu_k+s_k-1}, \dots, L_{\nu_k}$ are generated by the vertices $j_{n-(\nu_k+s_k-1)+1}, \dots, j_{n-\nu_k+1}$, which have level k. In Lemma 2, we show that the nonzero off-diagonal elements of G_k are equal to the nonzero off-diagonal elements of matrices $L_{\nu_k+s_k-1}, \dots, L_{\nu_k}$.

Lemma 2. *Let $S_k = \{j_{n-(\nu_k+s_k-1)+1}, \dots, j_{n-\nu_k+1}\}$ be the set of vertices in level k of algorithm BFS and let $G_k = L_{\nu_k+s_k-1}\dots L_{\nu_k}$, where $L_r = L_{p_r,q_r}(-\alpha_r)$ for $r = \nu_k, \dots, \nu_{k+s_k-1}$ and p_r is the parent of $q_r = j_{n-r+1}$. Then the entries of G_k are as follows:*

$$G_k(i,j) = \begin{cases} 1 & \text{if } i = j, \\ -\alpha_r = -\frac{m_{p_r,q_r}}{m_{q_r,q_r}} & \text{if } (i,j) = (p_r,q_r),\ q_r = j_{n-r+1} \in S_k, \\ & \quad \text{and } (p_r,q_r) \in E(T_A), \\ 0 & \textit{otherwise}. \end{cases}$$

Proof. From the definition of elimination matrix $L_r = L_{p_r,q_r}(-\alpha_r)$, we have

$$L_r = L_{p_r,q_r}(-\alpha_r) = I - \alpha_r E_{p_r,q_r},$$

where E_{p_r,q_r} contains only 0s except for 1 in the (p_r,q_r)th position. From the fact that all the vertices in S_k have level k, for all $q_r(= j_{n-r+1}), q_{r'}(= j_{n-r'+1}) \in S_k$, we have

$$E_{p_r,q_r} \times E_{p_{r'},q_{r'}} = 0 \quad \text{for } q_r \neq p_{r'}.$$

So, by the induction on the number of elimination vertices in level k, we can easily show that

$$G_k = I - \sum_{r=\nu_k}^{\nu_k+s_k-1} \alpha_r E_{p_r,q_r},$$

which completes the proof. □

Finally, summarizing the previous results, we describe the tree inverse algorithm for computing the inverse of M as follows:

Algorithm 3 Tree inverse

$input(T_A, root)$
$(v, Level) = BFS(T_A, root)$
$\tilde{M} = I$
$d = Diag(A)$
for $k = l$ to 1 step -1 (l is the number of levels obtained from the BFS algorithm) **do**
  $G = I$
  for all vertices in level k **do**
    j =current vertex
    i =the parent of current vertex
    $\alpha = -\frac{m_{ij}}{m_{jj}}$
    $g_{ij} = \alpha$
    $d_{ii} = d_{ii} - \alpha a_{ij}$
  end for
  $\tilde{M} = G\tilde{M}$
end for
set $P_A^{-1} = \tilde{M}^T D^{-1} \tilde{M}$

4 Numerical experiments

In this section, we compare the experimental results obtained by solving the preconditioned system of equation (1). Four preconditioners will be compared: MWST, AINV (right-looking version) [5], incomplete Cholesky, and RIF [6] preconditioners. In addition, the following approaches are used for applying the MWST preconditioners:

1. We use the matrices P_A^{-1} and P_B^{-1} computed by the tree inverse algorithm (Algorithm 3).

2. We use the Cholesky factorization of the MWST preconditioners P_A and P_B for computing Z_{j+1} in lines 1 and 6 of Algorithm 1.

3. In order to reduce the fill-in, first, we apply the reverse Cuthill–McKee ordering [12,13] to the preconditioners P_A and P_B and then we use the Cholesky factorizations of the resulting matrices.

Finally, for large matrices, we compare the results obtained by the approach 1 on GPU and CPU.

The examples have been coded in MATLAB with double-precision and have been executed on a quad-processor 4.2 GHz i7 computer with 32 GBytes of main memory. In all examples, the initial iteration matrix is zero. We stop the iterations when

$$RError = \frac{\|R_k\|_F}{\|R_0\|_F} \leq \epsilon,$$

where R_k is the residual of the kth iterate and ϵ is a proper stopping tolerance. In all the tables, the CPU time is in second and a dagger (†) indicates that no convergence is achieved after 10000 iterations except for Tables 5 and 6, where the maximum number of iterations is 30000. We also set the stopping tolerance 10^{-9}. The matrix C is chosen such that the exact solution X has the entries $x_{ij} = i * j$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

For the first set of examples, we consider the matrix NOS6 from Harwell–Boeing collection [20] and the matrix ST_n, which is obtained by discretizing the poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f \quad in\ \Omega =]0,1[\times]0,1[$$

with the Dirichlet boundary condition on a uniform grid of mesh size $h = \frac{1}{n+1}$ via central differences. These matrices with their properties are presented in Table 1. In this table, cond denotes the condition number of the matrices in 2-norm.

Table 1: First set of test problems information

Test matrix	n	nnz	cond
ST_5	25	105	20.77
ST_{10}	100	460	69.8634
ST_{20}	400	1920	258.4520
ST_{30}	900	4380	564.9227
ST_{40}	1600	7840	989.2690
ST_{50}	2500	12300	1531.5
Nos6	675	3255	8×10^6

In Table 2, we compare the number of iterations (It) and the CPU iteration time (It-time) for the preconditioners: the approximate inverse with drop tolerance equal to 0.1 (AINV) [5] , the incomplete Cholesky factorization

Table 2: Number of iterations and CPU times to converge for the first set of examples

Matrices		$MWST_1$		$MWST_2$		$MWST_3$		AINV		IC(0)		RIF	
A	B	It	It-time	It	It-time	It	It-time	It	It-time	It	It-time	It	It-time
NOS6	ST_5	207	0.16	196	0.20	194	0.11	431	0.24	273	0.20	199	0.29
NOS6	ST_{10}	499	0.97	485	1.86	486	1.63	1110	2.03	575	1.71	405	1.59
NOS6	ST_{20}	1131	16.78	1115	35.49	1115	20.93	2738	34.58	1353	24.61	962	21.27
NOS6	ST_{30}	1817	84.99	1797	162.92	1797	100.26	4936	235.54	2285	136.82	1591	105.34
NOS6	ST_{40}	2489	308.60	2470	512.05	2470	335.12	7771	978.51	3381	463.15	2396	385.51
NOS6	ST_{50}	3167	884.18	3152	1283.7	3148	873.3	†	†	4570	1242.9	3332	1094.4
NOS6	NOS6	725	22.90	694	35.25	694	23.12	2016	38.51	1737	62.65	766	35.05

Table 3: Preconditioning times and total times for the first set of examples

Matrices		$MWST_1$		$MWST_2$		$MWST_3$		AINV		IC(0)		RIF	
A	B	P-time	T-time	P-time	T-time	P-time	T-time	P-time	T-time	P-time	T-time	P-time	T-time
NOS6	ST_5	0.45	0.61	4.61	4.81	0.85	0.96	0.40	0.64	1.08	1.28	0.27	0.56
NOS6	ST_{10}	0.49	1.46	4.70	6.56	0.87	2.50	0.40	2.43	1.11	2.82	0.29	1.82
NOS6	ST_{20}	0.65	17.43	7.25	42.74	1.10	22.03	0.48	35.06	1.44	26.05	0.39	21.66
NOS6	ST_{30}	0.92	85.91	23.91	188.83	2.10	102.36	1.16	236.7	2.97	139.79	0.89	106.23
NOS6	ST_{40}	1.34	309.94	85.63	598.13	4.88	340.00	4.89	983.49	6.92	470.07	3.53	389.04
NOS6	ST_{50}	2.15	886.33	248.50	1532.20	10.35	883.65	17.02	†	15.25	1258.15	10.84	1105.24
NOS6	NOS6	0.80	23.70	9.19	44.44	1.63	24.75	0.78	59.29	2.14	64.79	0.56	35.61

(IC(0)), MWST using the approaches 1–3 ($MWST_1$, $MWST_2$, $MWST_3$, respectively), and the robust incomplete factorization with drop tolerance equal to 0.1 (RIF). The CPU time for computing the preconditioner (P-time) and the total time for computing an approximate solution (T-time) are given in Table 3. Table 2 reveals that the preconditioner $MWST_1$ is faster (in terms It-time) than the other preconditioners (except for $MWST_3$ with $A =$NOS6 and $B = ST_5, ST_{50}$) and it requires a lower number of iterations than AINV and IC(0) preconditioners. Table 3 shows that the preconditioners $MWST_1$ is faster (in terms T-time) than the other preconditioners (except for $MWST_3$ with $A =$nos6 and $B = ST_{50}$, and RIF for NOS6 and ST_5). In addition, for large matrices (NOS6 with ST_{40}, and ST_{50}), $MWST_1$ preconditioner is better (in terms of P-time) than the other preconditioners. For small matrices (NOS6 with ST_5, ST_{10}, ST_{20}, and ST_{30}), we observe that the time of constructing the preconditioner RIF is smaller than that of the other preconditioners. For the second set of examples, we define matrices $STM_n = ST_n + DI_n$, where DI_n is a diagonal matrix such that the matrix STM_n has zero row weights (except for one row, where we increase the row sums to obtain a nonsingular matrix). Table 4 represents the properties of these matrices.

Table 4: Second set of test problems information

Test matrix	n	nnz	cond
STM_5	25	105	2.0002×10^6
STM_{10}	100	460	8.0012×10^6
STM_{20}	400	1920	3.2006×10^7
STM_{30}	900	4380	7.2016×10^7
STM_{40}	1600	7840	1.2803×10^8
STM_{50}	2500	12300	2.0005×10^8

The results obtained for these matrices (which have large condition number) are presented in Tables 5 and 6. The results of Table 5 show that $MWST_1$ is the best in terms of iteration time (except for $MWST_3$ with $A = STM_5$, $B = STM_{30}$ and RIF with $A = STM_{20}$, $B = STM_{20}$). From the results of Table 6, we observe that, for large matrices, $MWST_1$ is better (in terms of total time) than the other preconditioners (except for RIF with $A = STM_{10}$, $B = STM_{10}$ and $A = STM_{20}$, $B = STM_{20}$). Finally, we consider the results obtained for the preconditioner $MWST_1$ in terms of CPU time, GPU time, and the number of iterations. We mention that the preconditioner was computed on the CPU. All numerical experiments in this section were computed in double precision with a MATLAB code. We used a Geforce GTX 1070 GPU with 8 GBytes VRAM memory. The results are listed in Tables 7 and 8. The notations CIT (GIT) and CIT-time (GIT-time) represent the number of iterations and CPU iteration time (GPU iteration time) on the CPU (GPU) required for convergence, respectively. These tables show that the number of iterations for the CPU and the GPU are close

Table 5: Number of iterations and CPU times to converge for the second set of examples

Matrices		$MWST_1$		$MWST_2$		$MWST_3$		AINV		IC(0)		RIF	
A	B	It	It-time	It	It-time	It	It-time	It	It-time	It	It-time	It	It-time
STM_5	STM_5	101	0.01	85	0.01	84	0.01	195	0.02	165	0.01	83	0.01
STM_5	STM_{10}	390	0.03	360	0.06	357	0.05	1135	0.11	669	0.10	447	0.08
STM_5	STM_{20}	932	1.06	882	1.52	881	1.09	4745	5.87	2152	3.31	1585	2.28
STM_5	STM_{30}	1491	8.86	1437	9.67	1435	8.50	†	†	4299	25.19	3327	24.73
STM_{10}	STM_{10}	831	0.18	768	0.49	762	0.29	1084	0.27	662	0.31	431	0.18
STM_{10}	STM_{20}	2264	4.72	2187	8.63	2186	5.66	7074	14.41	3149	9.62	2322	6.30
STM_{10}	STM_{30}	3596	34.00	3520	52.34	3514	34.80	†	†	6589	66.48	5141	60.66
STM_{20}	STM_{20}	4568	36.58	4420	89.29	4419	44.67	9185	69.22	3721	41.84	2667	29.07
STM_{20}	STM_{30}	7599	216.78	7517	461.86	7508	253.27	†	†	†	†	8309	333.13

Table 6: Preconditioning times and total times for the second set of examples

Matrices		$MWST_1$		$MWST_2$		$MWST_3$		AINV		IC(0)		RIF	
A	B	P-time	T-time	P-time	T-time	P-time	T-time	P-time	T-time	P-time	T-time	P-time	T-time
STM_5	STM_5	0.04	0.05	0.04	0.05	0.02	0.03	0.02	0.04	0.02	0.03	0.02	0.03
STM_5	STM_{10}	0.07	0.10	0.13	0.19	0.05	0.10	0.02	0.13	0.04	0.14	0.02	0.10
STM_5	STM_{20}	0.22	1.28	2.80	4.32	0.27	1.36	0.10	5.97	0.38	3.69	0.10	2.38
STM_5	STM_{30}	0.47	9.33	20.29	29.96	1.55	10.05	0.82	†	1.88	27.07	0.61	25.34
STM_{10}	STM_{10}	0.10	0.28	0.22	0.71	0.08	0.37	0.02	0.29	0.06	0.37	0.02	0.20
STM_{10}	STM_{20}	0.25	4.97	2.89	11.52	0.30	5.96	0.10	14.51	0.40	10.02	0.10	6.4
STM_{10}	STM_{30}	0.50	34.50	20.38	72.72	1.58	36.38	0.82	†	1.90	68.38	0.61	61.27
STM_{20}	STM_{20}	0.40	36.98	5.56	94.85	0.52	45.19	0.18	70.30	0.74	42.58	0.19	29.26
STM_{20}	STM_{30}	0.65	217.43	23.05	484.91	1.8	255.07	0.90	†	2.24	†	0.69	333.82

together and that the GPU time is very smaller than the CPU time for large matrices. So, we can conclude that $MWST_1$ preconditioner in the GL-CG method offers a great potential in a parallel processing environment.

Table 7: The performance of the preconditioner $MWST_1$ on CPU and GPU for the first set of examples

Matrices		$MWST_1$			
A	B	CIt	CIt-time	GIt	GIt-time
NOS6	ST_5	207	0.16	208	0.56
NOS6	ST_{10}	499	0.97	500	1.39
NOS6	ST_{20}	1131	16.78	1131	11.95
NOS6	ST_{30}	1817	84.99	1815	57.91
NOS6	ST_{40}	2489	308.60	2487	200.15
NOS6	ST_{50}	3167	884.18	3162	591.95
NOS6	NOS6	725	22.90	729	18.10

Table 8: The performance of the preconditioner $MWST_1$ on CPU and GPU for the second set of examples

Matrices		$MWST_1$			
A	B	CIt	CIt-time	GIt	GIt-time
STM_{10}	STM_{10}	831	0.18	833	1.09
STM_{10}	STM_{20}	2264	4.72	2267	4.73
STM_{10}	STM_{30}	3596	34.00	3604	20.35
STM_{10}	STM_{40}	4926	152.78	4931	71.82
STM_{10}	STM_{50}	6274	488.41	6287	197.01
STM_{20}	STM_{20}	4568	36.58	4578	24.91
STM_{20}	STM_{30}	7599	216.78	7611	143.72
STM_{20}	STM_{40}	10343	840.34	10354	520.89
STM_{20}	STM_{50}	13010	2234.6	13027	1500.6
STM_{30}	STM_{30}	11171	712.07	11199	491.93
STM_{30}	STM_{40}	15489	2563.8	15504	1731.5
STM_{30}	STM_{50}	19497	6225.7	19521	4999.5
STM_{40}	STM_{40}	20170	6165.1	20207	4156.8
STM_{40}	STM_{50}	25848	15348.00	25873	12075.00
STM_{50}	STM_{50}	31525	30637.00	31573	24410.00

5 Conclusion

We have proposed an approach for computing an approximate solution of matrix equation $AXB = C$, where A and B are Stieltjes matrices. In this approach, by using the BFS algorithm, we presented a new algorithm for obtaining the inverse of Vaidya's maximum spanning tree preconditioner as an approximate inverse preconditioner. This preconditioner does not require solving any linear systems and is highly parallelizable. We observed that this algorithm furnishes an efficient preconditioner for the matrix equations. The numerical experiments showed that, for large matrices, this preconditioner is better than the other preconditioner in terms of iteration time and total time and the new algorithm is very efficient on the GPU.

Acknowledgements

Authors are grateful to the anonymous referees and editor for their constructive comments.

References

1. Bai, Z.Z. *On Hermitian and skew-Hermitian splitting iteration methods for continuous Sylvester equations*, J. Comput. Math. 29(2) (2011), 185–198.

2. Baur, U., Benner, P. *Cross-gramian based model reduction for data-sparse systems*, Electron. Trans. Numer. Anal. 31 (2008), 256–270.

3. Beauwens, R. *Upper eigenvalue bounds for pencils of matrices*, Linear Algebra Appl. 62 (1984), 87–104.

4. Beauwens, R. *Approximate factorizations with modified S/P consistently ordered M-factors*, Numer. Linear Algebra Appl. 1(1) (1994), 3–17.

5. Benzi, M., Meyer, C.D. and Tuma, M. *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput.17(5) (1996), 1135–1149.

6. Benzi, M. and Tuma, M. *A robust incomplete factorization preconditioner for positive definite matrices*, Preconditioning, 2001 (Tahoe City, CA). Numer. Linear Algebra Appl.10(5-6) (2003), 385–400.

7. Bern, M., Gilbert, J.R., Hendrickson, B., Nguyen, N. and Toledo, S. *Support-graph preconditioners*, SIAM J. Matrix Anal. Appl. 27(4) (2006), 930–951.

8. Bouhamidi, A., Jbilou, K., Reichel, L. and Sadok, H. *An extrapolated TSVD method for linear discrete ill-posed problems with kronecker structure*, Linear Algebra Appl. 434(7) (2011), 1677–1688.

9. Calvetti, D. and Reichel, L. *Application of ADI iterative methods to the restoration of noisy images*, SIAM J. Matrix Anal. Appl. 17(1) (1996), 165–186.

10. Datta, B.N. *Numerical methods for linear control systems: design and analysis*, Elsevier, Academic Press, 2004.

11. Deng, Y.B., Bai, Z.Z. and Gao, Y.H. *Iterative orthogonal direction methods for hermitian minimum norm solutions of two consistent matrix equations*, Numer. Linear Algebra Appl. 13(10) (2006), 801–823.

12. George, J.A. *Computer implementation of the finite element method*, Technical report, Department of Computer Science, Stanford University, 1971.

13. George, J.A. and Liu, J.W.H. *Computer Solution of Large Sparse Positive Definite System*, Prentice-Hall, 1981.

14. Gremban, K. D. *Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems*, PhD thesis, Carnegie Mellon University, 1996.

15. Gremban, K.D., Miller, G.L. and Zagha, M. *Performance evaluation of a new parallel preconditioner*, 9th International Parallel Processing Symposium, IEEE, (1995), 65–69.

16. Guennouni, A.E., Jbilou, K. and Riquet, A.J. *Block Krylov subspace methods for solving large Sylvester equations*, Matrix iterative analysis and biorthogonality (Luminy, 2000).Numer. Algorithms, 29(1-3) (2002), 75–96.

17. Horn, R.A., and Johnson, C.R. *Topics in matrix analysis*, Cambridge University Press, 1991.

18. Khojasteh-Salkuyeh, D. *Cg-type algorithms to solve symmetric matrix equations,* Appl. Math. Comput. 172(2) (2006), 985–999.

19. Khojasteh-Salkuyeh, D. and Toutounian, F. *New approaches for solving large Sylvester equations,* Appl. Math. Comput. 173(1) (2006), 9–18.

20. Matrix Market, Available at `http//math.nist.gov/Matrix Market,` May 2007.

21. Notay, Y. *Solving positive (semi) definite linear systems by preconditioned iterative methods*, Preconditioned Conjugate Gradient Methods, Lectures Notes in Mathematics, Springer, 1990.

22. Notay, Y. *Conditioning analysis of modified block incomplete factorizations*, Linear Algebra Appl. 154 (1991), 711–722.

23. Notay, Y. *Conditioning of Stieltjes matrices by S/P consistently ordered approximate factorizations*, Appl. Numer. Math. 10(5) (1992), 381–396.

24. Saad, Y. *Iterative methods for sparse linear systems*, SIAM, 2003.

25. Skiena, S.S. *The algorithm design manual: Text,* Springer Science & Business Media, 1998.

26. Vaidya, P.M. *Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners,* Unpublished manuscript UIUC, A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, 1991.

27. Wang, M. and Feng, Y. *An iterative algorithm for solving a class of matrix equations,* J. Control Theory Appl. 7(1) (2009), 68–72.

28. Xie, L., Ding, J. and Ding, F. *Gradient based iterative solutions for general linear matrix equations*, Comput. Math. Appl. 58(7) (2009), 1441–1448.