Johann Radon Institute for
Computational and Applied Mathematics
Austrian Academy of Sciences (ÖAW)

Condition number study of graph theory based preconditioners for isogeometric discretization of Poisson equation

K. Gahalaut, S. Tomar

RICAM-Report 2012-14

Condition number study of graph theory based preconditioners for isogeometric discretization of Poisson equation

K.P.S. Gahalaut and S.K. Tomar
RICAM, Austrian Academy of Sciences, Linz, Austria

May 16, 2012

Abstract

We study the preconditioning of the stiffness matrix which arises from the discretization of the Poisson equation using IsoGeometric Method (IGM). A study of condition number of stiffness matrices, resulting from NURBS based IGM, suggests that novel preconditioning techniques are needed for fast and efficient iterative solvers for the resulting linear system. As a first step towards preconditioning the resulting stiffness matrix, we use graph theory based preconditioners, namely, Vaidya's preconditioners (maximum weight spanning tree) and Gremban and Miller's preconditioners (support tree). Numerical results show that these preconditioners do not perform satisfactorily for the matrices arising in IGM.

1 Introduction

IsoGeometric Analysis (IGA), introduced in [14], aims to bridge the gap between Finite Element Analysis (FEA) and Computer Aided Design (CAD). The main idea of IGA is to directly use the geometry provided by CAD system, and to approximate the unknown solution of differential equation by the same functions which are used in CAD. IGA offers several advantages when compared to FEM. For example, complicated geometries are represented more accurately, and some geometries commonly arising in engineering and applied sciences, such as circles or ellipses, are represented exactly. In particular, the description of the geometry, taken directly from the CAD system, is incorporated exactly at the coarsest mesh level. This eliminates the necessity of further communication with the CAD when mesh refinement is carried out, thereby mesh refinement does not modify the geometry. There are several computational geometry technologies that could serve as a basis for IGA. However, NURBS are the most widely used and well established computational technique in CAD, and are used in our study. For other advantages see early papers on IGA, e.g. [12, 2, 3, 4, 5, 10]. To avoid proliferation of notations, we omit the description related to B-splines/NURBS basis and their properties, and refer the reader to [15, 16, 11].

As a model problem we consider the Poisson problem in an open, bounded and connected Lipschitz domain Ω with Dirichlet boundary $\partial\Omega$.

$$-\Delta u = f \quad \text{in } \Omega, \qquad u = u^D \quad \text{on } \partial\Omega. \tag{1}$$

Let $V^0 = H_0^1(\Omega)$ denote the space of test functions which vanishes on $\partial\Omega$, and $V^D = V^0 + u^D$ denote the set which contains the functions fulfilling the Dirichlet boundary condition on $\partial\Omega$. By $V_h^0 \subset V^0$ and $V_h^D \subset V^D$ we denote the finite-dimensional spaces of the B-spline (NURBS) basis functions. Introducing the bilinear form $a(\cdot,\cdot)$ and the linear form $f(\cdot)$ as

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v \; dx, \quad f(v) = \int_\Omega f \; v \; dx, \tag{2}$$

the Galerkin formulation of this problem reads:
Find $u_h \in V_h^D$ such that

$$a(u_h, v_h) = f(v_h) \quad \text{for all } v_h \in V_h^0. \tag{3}$$

It is well known that (3) is a well-posed problem and has a unique solution. By approximating u_h and v_h using B-spline (NURBS) basis functions, the weak formulation (3) is transformed in to a set of linear algebraic equations

$$A_h u_h = f_h, \tag{4}$$

where $A_h \in \mathbb{R}^{n \times n}$ denotes the stiffness matrix obtained from the bilinear form $a(\cdot, \cdot)$, $u_h \in \mathbb{R}^n$ denotes the vector of unknown degrees of freedom (DOF), and $f_h \in \mathbb{R}^n$ denotes the vector of right hand side (RHS) from the known data of the problem. Hereinafter, for simplicity reasons we denote A_h by A.

For the solution of symmetric and positive definite (SPD) linear system of equations such as 4, Conjugate Gradient Method (CGM), and its preconditioned version, are one of the most promising techniques. The number of iterations of the CGM depends on the condition number $\kappa(A)$, where $\kappa = \lambda_{\max}/\lambda_{\min}$ (ratio of largest to smallest eigenvalues). Since the condition number of the stiffness matrix A arising from IGA based discretization of Poisson equation is high, it becomes necessary to develop fast and efficient preconditioners for iterative solution of 4. The purpose of this report is to document our study on graph theory based preconditioners as a first step towards preconditioning the linear system 4.

A brief outline of this report is as follows. Section 2 briefly describes the basic idea of support graph theory based preconditioners, namely, maximum weight spanning tree preconditioners by Vaidya [17, 9], and support tree preconditioners by Gremban and Miller [13], see also related papers [6, 7, 8]. Section 3 describes various numerical techniques for preconditioning. Local preconditioning is discussed in Section 4 and Section 5 contains the resulting numerical results. Conclusions are summarized in Section 6. All the algorithms from the techniques presented in Section 3, and referred throughout the text, are listed in Appendix.

2 Preliminaries of Support Graph Theory

Support graph theory gives us a new class of preconditioners, called support graph preconditioners. The main idea is to use a subgraph of the graph of the coefficient matrix A as a preconditioner. These graphs are connected graphs, and the preconditioner graph edges are the subset of the actual graph edges. Moreover, the preconditioner graph edges support the actual graph edges.

We first present some definitions which are related to the graph theory.

Walk: A walk is an alternating sequence of vertices and edges that begins and end with a vertex, such that any edge in the sequence connects the vertex preceding it to the vertex following it.

Path: A path is a walk in which all the vertices are distinct.

Graph embedding: Let G and H be two graphs. An embedding of H into G is a mapping of vertices of H onto vertices of G and edges of H onto paths in G.

Dilation: The dilation of an edge e of H is the number of edges in its support path (length of support path) in G. The dilation of the embedding is the length of the longest path in G onto which an edge of H is mapped.

Congestion: The congestion of an edge e in G is the number of paths of the embedding that contain e. The congestion of the embedding is the maximum congestion of the edges in G.

Support: The support $\sigma(A, B)$ of matrix B for matrix A is

$$\min\{\tau : \tau B - A \text{ is positive semi-definite (PSD)}\}.$$

The following results [13] are of fundamental importance for graph-theory based preconditioners.

Lemma 2.1 (Support lemma) *Let A and B be SPD matrices. If $\tau B - A$ is PSD, where τ is a positive number, then $\lambda_{\max}(B^{-1}A) \leq \tau$.*

The support lemma shows that the support of a matrix pair (A, B) bounds its eigenvalue, i.e.,

$$\lambda_{\max}(B^{-1}A) \leq \sigma(A, B).$$

Note that

$$1/\lambda_{\min}(B^{-1}A) = \lambda_{\max}(A^{-1}B) \leq \sigma(B, A),$$

which implies

$$\kappa(B^{-1}A) \leq \sigma(A, B).\sigma(B, A).$$

Lemma 2.2 (Splitting lemma) *If $A = A_1 + A_2 + ... + A_m$, where $A_1, A_2, ..., A_m$ are all PSD matrices, then A is a PSD matrix.*

Furthermore, it can be proved that

$$\sigma(A, B) \leq \max\{\sigma(A_i, B_i)\},$$

where

$$A = \sum A_i, \quad \text{and} \quad B = \sum B_i,$$

satisfies the splitting lemma.

In other words, let $A = A_1 + A_2 + \ldots + A_m$, and $B = B_1 + B_2 + \ldots + B_m$. Assume that we have a set $\{\tau_1, \tau_2, \ldots, \tau_m\}$, such that $\tau_i B_i - A_i$ is PSD for all i. Let $\tau^* = \max\{\tau_1, \tau_2, \ldots, \tau_m\}$ then $\tau^* B_i - A_i$ is PSD for all i.

3 Graph Preconditioning Techniques

3.1 Maximum Weight Spanning Tree Preconditioners

Graph preconditioners, introduced by Vaidya [17] in early nineties, use maximum weight spanning tree (MWST) preconditioners to bound the condition number of a preconditioned system. Vaidya's method constructs a preconditioner V whose underlying graph G_V is a subgraph of G_A (graph of A). The graph G_V of the preconditioner has the same set of vertices as G_A, and a subset of the edges of G_A. The methodology of Vaidya's preconditioners is described in Algorithm 1.

Vaidya proposed two classes of preconditioners. The first class, MWST preconditioners, guarantees a condition-number bound of $O(n^2)$ for any $n \times n$ sparse diagonally dominant symmetric matrix [6]. The second class of preconditioners is based on MWST augmented with a few extra edges. The cost of factoring this second class of preconditioners depends on how many edges are added to the tree. Vaidya proposed that the factorization cost can be balanced with the iteration costs, and he provided balancing guidelines for some classes of matrices. 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, and by $O(n^{1.2})$ for sparse diagonally dominant matrices whose underlying graphs are planar.

We construct MWST by simply dropping some off-diagonal non-zeros from A and modifying the diagonal elements to maintain certain row-sum property. We construct the preconditioners in the following ways.

1. Algorithm 2 constructs MWST with maximum value (not absolute maximum) and makes the matrix of MWST diagonally dominant. Note that the preconditioner matrix is diagonally dominant but not an M-matrix.

2. Algorithm 3 first transforms the A matrix into an M-matrix, and then finds the MWST. So the resulting preconditioner is diagonally dominant as well as M-matrix.

3. Algorithm 4 first drops all the positive off-diagonal entries, and then constructs MWST of modified stiffness matrix. To maintain the row-sum property it performs a weighted distribution of the sum of dropped positive off-diagonal entries and dropped negative off-diagonal entries (while forming MWST) on the negative off-diagonal entries of MWST. In a variant of Algorithm 4 we make the following changes. If the sum of dropped positive off-diagonal entries and dropped negative off-diagonal entries (while forming MWST) is positive then it is added to the diagonal entry, otherwise the algorithm performs a weighted distribution of the sum on the negative off-diagonal entries of MWST. Note that in the variant of Algorithm 4 the preconditioned matrix is not symmetric.

4. Algorithm 5 first constructs MWST with maximum value (not absolute maximum), and calculates the sum of the dropped positive and negative off-diagonal entries. If this sum is positive it performs a weighted distribution of this sum on the negative off-diagonal entries of the tree, and if this sum is negative then it adds the sum to the diagonal entry of MWST.

Some numerical results using Vaidya's preconditioners and support tree preconditioners (see Section 3.2) are given in Table 1 using linear basis functions. Here M_w denotes the MWST preconditioner matrix, P_V denotes the preconditioner from Vaidya's approach, k denotes the number of added extra edges, and P_T denotes the preconditioner from support tree approach.

Table 1: Vaidya's preconditioning

mesh	4×4	8×8	16×16	32×32	64×64
$\kappa(A)$	3.15	12.82	51.71	207.34	829.85
$\kappa(M_w^{-1}A)$	2.75	10.73	45.80	184.08	737.79
$\kappa(P_V^{-1}A), k = (n/log_{10}n)$	1.00	5.54	13.43	53.15	158.17
$\kappa(P_V^{-1}A), k = (n/2)$	1.91	5.54	11.71	20.47	46.23
$\kappa(P_T^{-1}A)$	3.04	12.77	51.70	207.33	829.85

3.2 Support Tree Preconditioners

Gremban and Miller extended Vaidya's work, and introduced support tree preconditioners. A support tree preconditioner is a preconditioning matrix whose associated graph is a tree on a superset of the vertices. The tree is constructed to provide good support for coefficient matrix. Each vertex in the graph of the matrix becomes a leaf of the support tree, and the internal vertices of the tree are added vertices. The algorithmic way of constructing support tree preconditioners is given in Algorithm 6.

The construction of support tree is based on hierarchical decomposition of the graph G_A. The algorithm removes from G_A a set of edges, known as a separator, that breaks it into a small number of subgraphs $G_1, G_2, \ldots, G_k$. The algorithm then recursively partitions each G_i until the graph is decomposed into single vertices. The resulting support tree T of the graph G_A can not be used as

preconditioner because number of vertices in T are more than the number of vertices in G_A. Now we perform Gaussian elimination on

$$T = \begin{bmatrix} B_{11} & B_{12} \\ B_{12}^T & B_{22} \end{bmatrix}$$

to eliminate all the internal vertices, where block B_{11} is a square matrix of the same order as A, say n, block B_{22} is a square matrix whose order is the number of extra added vertices in the support tree, say m, and block B_{12} is a rectangular matrix of order $n \times m$. Eliminating the internal vertices of T produces a matrix T' defined as

$$T' = \begin{bmatrix} B & 0 \\ 0 & D \end{bmatrix},$$

where B is the Schur compliment $B_{11} - B_{12}B_{22}^{-1}B_{12}^T$, and D is a diagonal matrix. Gremban and Miller used this matrix B as a preconditioner of A. The condition number of support tree preconditioned matrix is of $O(n \log n)$. Unfortunately, involvement of matrix B_{22}^{-1} seriously limits the advantages of these preconditioners, and does not offer a significant gain in the condition number of preconditioned system ($B^{-1}A$), see last row of Table 1.

4 Local Preconditioning

To study the effect of local preconditioning (element level), we now consider the symmetric positive semi-definite (SPSD) stiffness matrix A_e of one element (without imposing the boundary condition) but with increasing p. We study the effective condition number κ_{eff} of the generalized eigenvalue problem

$$\tilde{A}_e x = \lambda \tilde{B}_e x \tag{5}$$

where $\tilde{A}_e = P_e^T A_e P_e$, $\tilde{B}_e = P_e^T B_e P_e$, B_e is some matrix obtained from A_e, and the matrix $P_e \in \mathbb{R}^{n \times (n-1)}$, defined as

$$P_e = \begin{pmatrix} 1 & & & & & & \\ -1 & 1 & & & & & \\ & -1 & 1 & & & & \\ & & & \cdots & \cdots & \cdots & \\ & & & & & -1 & 1 \\ & & & & & & -1 \end{pmatrix}, \tag{6}$$

is the transformation matrix to eliminate the zero eigenvalue.

4.1 M-matrix Approach

We approximate the SPSD matrix A_e by an M-matrix B_e which is obtained from A_e by the following approaches.

1. Algorithm 7 drops all positive off-diagonal entries. Note that the resulting preconditioned matrix in this approach is not positive definite.

2. Algorithm 8 adds all positive off-diagonal entries to the diagonal and replaces them by zero.

3. Algorithm 9 drops all positive off-diagonal entries and performs a weighted distribution of the sum of dropped positive off-diagonal entries on the negative off-diagonal entries (row-wise). Note that the resulting preconditioned matrix is not symmetric.

4.2 Two-Colors Approach

Given a mesh, multi-coloring consists of assigning a color to each point so that the couplings between two points of the same color are eliminated in the discretization matrix. We color the graph of stiffness matrix by two colors such that each color forms a disjoint set. This can also be done by reordering the matrix in such a way that the diagonal blocks of the matrix form two disjoint sets of nodes. Suppose we divide the nodes of the graph of stiffness matrix into two disjoint sets, say S_1 and S_2. Then

$$A_e = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

where block A_{11} is the matrix of connections within S_1, block A_{22} is the matrix of connections within S_2, block A_{12} is the matrix of connections of S_1 to S_2, and block A_{21} is the matrix of connections of S_2 to S_1. We denote the modified matrix by B_e such that

$$B_e = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$

where B_{11}, B_{12}, B_{21}, and B_{22} are the modified versions of A_{11}, A_{12}, A_{21}, and A_{22}, respectively.

The following three algorithms modify A_{11} to a diagonal matrix. The other three blocks A_{12}, A_{21} and A_{22} are returned unchanged.

1. Algorithm 10 modifies A_{11} by simply dropping all off-diagonal entries.
2. Algorithm 11 drops all off-diagonal entries and the diagonal entries are the row-wise sum of A_{11}.
3. Algorithm 12 drops all off-diagonal entries and the diagonal entries are the row-wise absolute sum of A_{11}.

In the following three approaches the algorithms modify the blocks A_{11} and A_{12}, and return the remaining two blocks A_{21} and A_{22} unchanged.

1. Algorithm 13 drops all off-diagonal entries of A_{11}, and performs a row-wise weighted distribution of the sum of dropped entries of A_{11} in A_{12}.
2. Algorithm 14 drops all off-diagonal entries of A_{11}, adds the row-wise sum of positive off-diagonal entries of A_{11} to its diagonal, and performs a row-wise weighted distribution of the sum of the remaining negative entries of A_{11} in A_{12}. We make the following changes in a variant of Algorithm 14, drop all off-diagonal entries of A_{11}, adds the row-wise sum of negative off-diagonal entries of A_{11} to its diagonal, and performs a row-wise weighted distribution of the sum of the remaining positive entries of A_{11} in A_{12}.

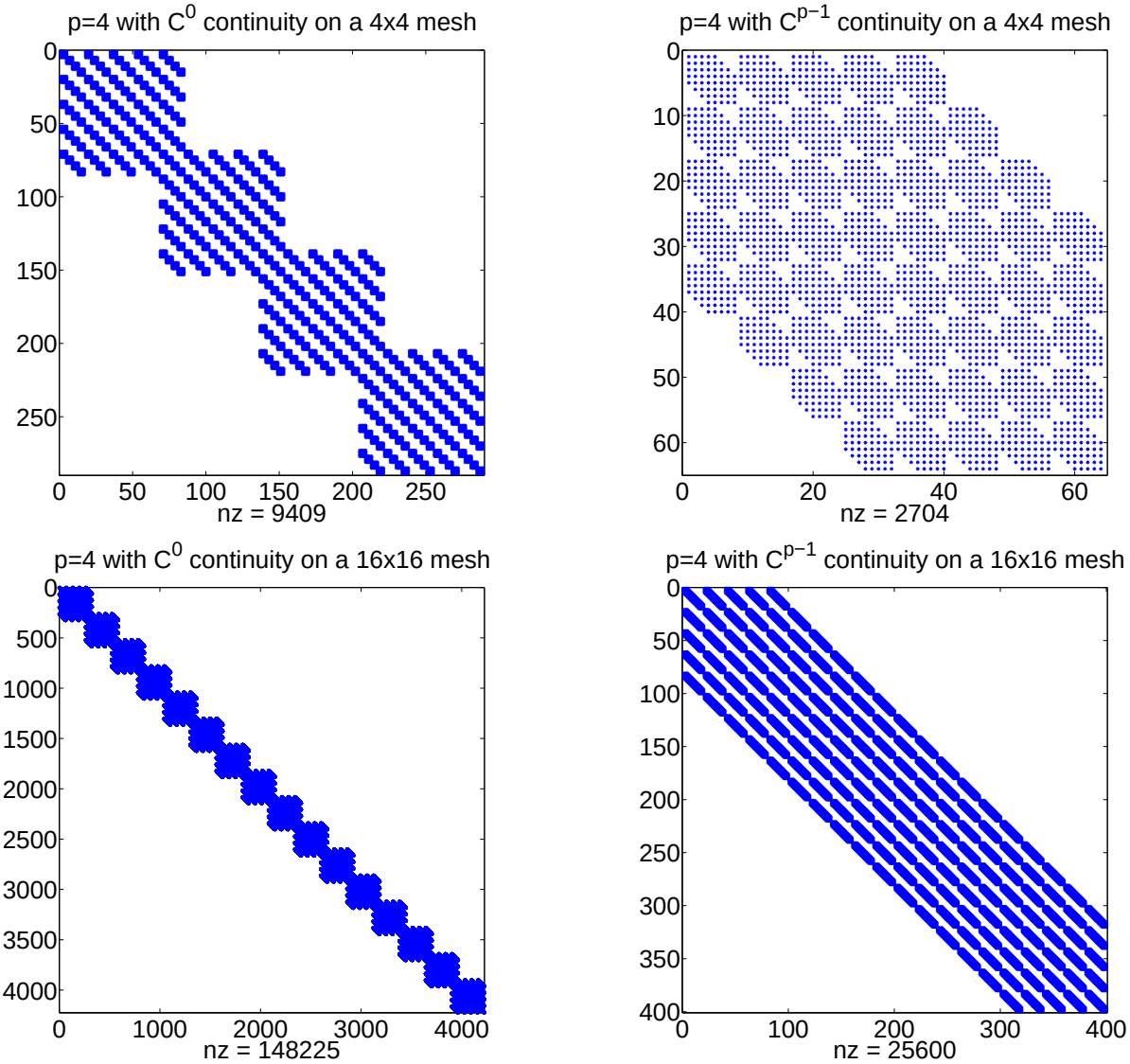
The last two approaches are as follows.

- In Algorithms 10–14 the block B_{11} is a diagonal matrix. Algorithm 15 returns a non-diagonal block B_{11} by adding the sum of positive entries to the diagonal for each row in A_{11}, and keeping the negative entries untouched. The resulting matrix B_{11} is positive definite.
- In Algorithms 13–15 the preconditioning matrix B is non-symmetric. Algorithm 16 returns a symmetric matrix by performing the following steps:
  - The absolute row-sum of A_{11} forms B_{11}.
  - To maintain row-sum property, the added quantity is distributed (in a weighted sense) to the block A_{12} to get B_{12}
  - Transpose of B_{12} forms B_{21}. Again, to maintain row-sum property, the added quantity in B_{21} is added to the diagonal entry of A_{22} to get B_{22}.

5 Numerical Results

For our numerical tests we consider the problem (1) with domain $\Omega = (0,1)^2$, and $u^D = 0$. We know that the sparsity of the coefficient matrix can be exploited to reduce the amount of necessary work in building and solving the algebraic system. Therefore, in Figure 1, we first depict the sparsity pattern of the coefficient matrices using fourth degree NURBS polynomials, with C^0 and C^{p-1} continuity, with uniform knot vectors in a single patch of 4×4 mesh and 16×16 mesh. Clearly,

Figure 1: Sparsity pattern of A



the block structure is reduced for higher continuity and the band is denser, however, the nonzero size is reduced due to the reduced problem size.

The condition number of the coefficient matrix is calculated with NURBS basis functions of degrees $p = 2, \ldots, 6$ and with different mesh sizes, and are shown in Table 2 for C^0- and C^{p-1}-continuity. Note that, in the following tables $\times$ signifies that the particular case could not be computed (due to limitations of the corresponding algorithm), and $-$ signifies that the corresponding entry is not computed because the preceeding results did not show any improvement.

Table 2: Condition number of A

p \ mesh	2×2	4×4	8×8	16×16	32×32
C^0-continuity					
2	7	13	36	140	555
3	75	107	120	270	1075
4	881	1100	1189	1215	1762
5	11094	12951	13680	13886	13940
6	146296	162488	168707	170451	170900
C^{p-1}-continuity					
2	4	4	5	20	78
3	31	29	29	29	82
4	340	269	240	223	215
5	4177	3220	2148	1812	1700
6	54385	46606	20819	15133	–

5.1 Preconditioning of the Stiffness Matrix

Since the qualitative behavior of the numerical results does not essentially change for $p \geq 5$, to avoid proliferation we omit the preconditioning results for $p \geq 5$ in all cases. In Table 3 we present the results for the condition number of the preconditioned system $P^{-1}A$, where P is the preconditioning matrix of A. The preconditioner P is obtained from various approaches listed in the algorithms, and the Jacobi preconditioner D (i.e. the diagonal of the matrix A). Since the preconditioned system of Algorithm 5 is singular, its results are not shown here. In Table 4 the condition number of the preconditioned system $P^{-1}A$ are shown for C^{p-1} continuity.

Table 3: Condition number of $P^{-1}A$ for C^0-continuity

p \ mesh	2×2	4×4	8×8	16×16	32×32
$P = D$					
2	2	6	22	87	346
3	20	30	48	192	766
4	181	249	275	331	1323
$P = B_1$, see Algorithm 2					
2	2	6	23	92	366
3	5	12	48	190	761
4	50	60	127	524	–
$P = B_2$, see Algorithm 3					
2	×	4	16	66	280
3	20	28	32	98	–
4	220	299	325	336	–
$P = B_3$, see Algorithm 4					
2	×	16	155	979	9409
3	18	388	4525	22757	–
4	171	3259	41763	–	–

Table 4: Condition number of $P^{-1}A$ for C^{p-1}-continuity

p \ mesh	2×2	4×4	8×8	16×16	32×32
$P = B_1$, see Algorithm 2					
2	2	3	6	21	84
3	9	9	13	19	80
4	89	45	105	257	1056
$P = B_2$, see Algorithm 3					
2	×	4	4	17	68
3	×	22	20	20	53
4	228	185	123	116	113

5.2 Local Preconditioning

We now present the effective condition number of the generalized eigenvalue problems (5) in Table 5, where B_e is computed from Algorithm 8. Recall that the preconditioned matrix in Algorithm 7 is not positive definite, and in Algorithm 9 it is not symmetric, therefore, corresponding results are not shown here.

Table 5: Effective condition number of (A_e, B_e) for $p = 2, \ldots, 6$

B_e \ p	2	3	4	5	6
Alg. 8	2.4	25.8	246.5	2336	23999

The observation from the two-colors approach discussed in Section 4.2 are as follows.

- The resulting preconditioned system from Algorithm 11 is indefinite because in the matrix A_{11} the negative off-diagonal entries are dominating the diagonal entry.
- The resulting preconditioned system from Algorithms 12 and 15 are positive definite but the effective condition number of the preconditioned system is very large.
- The Algorithms 10, 13 and 14 give indefinite preconditioned systems.
- The resulting preconditioning system in Algorithm 16 is SPD but the effective condition number of the preconditioned system is very large.

5.3 Preconditioning of the Schur Complement Matrix

In this section we present the condition number of Schur complement S (obtained by eliminating all degrees of freedom inside the elements) and the preconditioned Schur complement system.

The condition number of the Schur Complement for C^0-continuity is shown in Table 6. In Table 7, results are shown for the condition number of the preconditioned system $P_S^{-1}S$, where P_S is the preconditioning matrix obtained from various approaches listed in the algorithms, and the diagonal matrix obtained from S.

The eigenvalues of the generalized eigenvalue problems $\tilde{S_e}x = \lambda \tilde{Bs_e}x$ are calculated, and the effective condition number κ_{eff} of (S_e, Bs_e) is shown in Table 8, where Bs_e is computed from Algorithm 8. Algorithms 4 and 5 have also been tried but results are not favorable and not shown here.

Table 6: Condition number of S

p \ mesh	2×2	4×4	8×8	16×16	32×32
2	3	9	28	106	417
3	9	14	37	147	585
4	29	40	48	178	706
5	100	123	130	205	815
6	355	410	430	436	951

Table 7: Condition number of $P_S^{-1}S$

p \ mesh	2×2	4×4	8×8	16×16	32×32
$P_S = D_S$					
2	2	5	19	73	291
3	7	10	33	132	529
4	20	29	47	181	720
$P_S = B_{S_1}$, see Algorithm 2					
2	2	5	20	77	307
3	3	8	29	115	457
4	5	12	45	176	703
$P_S = B_{S_2}$, see Algorithm 3					
2	×	4	14	59	241
3	5	6	19	76	303
4	18	24	26	99	398

Table 8: Effective condition number of (S_e, B_{S_e}) for $p = 2, \ldots, 6$

B_{S_e} \ p	2	3	4	5	6
Alg. 8	1.3	5.8	20.1	64.2	203.2

6 Conclusion

We have numerically studied the behavior of condition numbers of graph theory based preconditioners for NURBS-based IGM discretization of Poisson equation. It can be seen that the condition number of preconditioned system, based on both the types of Vaidya's preconditioners, as well as support tree preconditioners of Gremban and Miller, is not independent of mesh-size. It is important to note that the support graph preconditioners are sensitive to the nonzero structure of the coefficient matrix, and not to the values of its entries. However, these methods are limited to the class of Symmetric Diagonally Dominant (SDD) matrices. Unfortunately, except for $p = 1$ when we get the stiffness matrix as an M-matrix, the matrices obtained from IGM for $p \geq 2$ can not be guaranteed to be SDD. We tried a few approaches (discussed in earlier sections) for higher p using support graph preconditioners but in the current form these approaches do not give condition numbers independent of mesh-size. Therefore, it is not possible to develop optimal order solver with these preconditioners.

Acknowledgments

The authors thank Dr. Johannes Kraus (RICAM, Linz) for helpful discussions. This research was supported by Austrian Science Fund (FWF) Project **P21516-N18**.

References

[1] F. Auricchio, L. Beirão da Veiga, A. Buffa, C. Lovadina, A. Reali, and G. Sangalli. A fully "locking-free" isogeometric approach for plane linear elasticity problems: A stream function formulation. *Comput. Methods Appl. Mech. Engrg.* 197 (2007), 160–172.

[2] Y. Bazilevs, L. Beirão Da Veiga, J.A. Cottrell, T.J.R. Hughes, and G. Sangalli. Isogeometric analysis: Approximation, Stability and error estimates for h- refined meshes. *Math. Models Methods Appl. Sci.* 16, #7 (2006), 1031–1090.

[3] Y. Bazilevs, V.M. Calo, Y. Zhang, and T.J.R. Hughes. Isogeometric fluid-structure interaction analysis with applications to arterial blood flow. *Comput. Mech.* 38 (2006), 310–322.

[4] Y. Bazilevs, and T.J.R. Hughes. Weak imposition of Dirichlet boundary conditions in fluid mechanics. *Computers and Fluids* 36 (2007), 12–26.

[5] Y. Bazilevs, C. Michler, V.M. Calo, and T.J.R. Hughes. Weak Dirichlet boundary conditions for wall-bounded turbulent flows. *Comput. Methods Appl. Mech. Engrg.* 196 (2007), 4853–4862.

[6] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo. Support Graph Preconditioners. *SIAM J. Matrix Anal. Appl.* 27 (2006), 930–951.

[7] E. Boman, D. Chen, B. Hendrickson and S. Toledo. Maximum-weight-basis preconditioners. *Numer. Linear Algebra Appl.* 11 (2004), 695–721.

[8] E. Boman and B. Hendrickson. Support theory for preconditioning. *SIAM J. Matrix Anal. Appl.* 25 (2003), 694–717.

[9] D. Chen and S. Toledo. Vaidya's Preconditioners: Implementation and Experimental Study. *ETNA* 16 (2003), 30–49.

[10] J.A. Cottrell, T.J.R. Hughes, and A. Reali. Studies of refinement and continuity in isogeometric structural analysis. *Comput. Methods Appl. Mech. Engrg.* 196 (2007), 4160–4183.

[11] J.A. Cottrell, T.J.R. Hughes, Y. Bazilevs. *Isogeometric Analysis: Toward Integration of CAD and FEA*. Wiley, 2009.

[12] J.A. Cottrell, A. Reali, Y. Bazilevs, and T.J.R. Hughes. Isogeometric analysis of structural vibrations. *Comput. Methods Appl. Mech. Engrg.* 195 (2006), 5257–5296.

[13] K. Gremban. Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems. PhD thesis, School of Computer Science, Cargenei Mellon University, Pittsburgh, (1996).

[14] T.J.R. Hughes, J.A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Comput. Methods Appl. Mech. Engrg.* 194 (2005), 4135–4195.

[15] L. Piegl, W. Tiller. *The NURBS Book (Monographs in Visual Communication)*, Second ed., Springer-Verlag, New York, 1997.

[16] D.F. Rogers. *An Introduction to NURBS With Historical Perspective.* Academic Press, San Diego, CA, 2001.

[17] P. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. An unpublished manuscript. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computations, Minneapolis, (1991).

Appendix

The algorithms resulting from all the techniques discussed in this report are listed here.

Algorithm 1 Vaidya's Preconditioner

Require: G_A (Graph of A)
Ensure: Preconditioner V

1. *Find a maximum weight spanning tree T in G_A.*
2. *Decompose it into a set of k connected subgraphs $T_1, ..., T_k$, such that each T_i has roughly same number of vertices.*
3. *Form G_V by adding to T the heaviest edge between T_i and T_j for all i and j.*
4. *Add nothing if there are no edges between T_i and T_j or the heaviest edge between T_i and T_j is already in T.*
5. *The preconditioner V is the matrix whose graph is G_V.*

Algorithm 2 MWST preconditioner (Diagonally Dominant)

Require: Stiffness Matrix A
Ensure: Preconditioner B_1

1. *Find maximum weight spanning tree B_{mwst} with maximum value (NOT absolute maximum).*
2. ***if** B_{mwst} is diagonally dominant **then***
3. $B_1 = B_{mwst}$ *(output matrix).*
4. ***else***
5. *find* $s_i = \sum_{i \neq j} b_{ij}$ *for each row i of B_{mwst}.*
6. ***if** $b_{ii} < s_i$ **then***
7. $b_{ii} = s_i$.
8. *The resulting matrix is B_1.*

Algorithm 3 MWST preconditioner (Diagonally dominant M-matrix)

Require: Stiffness Matrix A
Ensure: Preconditioner B_2

1. ***for** $i = 1$ to # of rows in A **do***
2. ***for** $j = 1$ to # of columns in A **do***
3. ***if** $a_{ij} > 0$ **then***
4. $a_{ii} = a_{ii} + a_{ij}$
5. $a_{ij} = 0$
6. *From step (1), we get an M-matrix, say A_m.*
7. *Find maximum weight spanning tree B_{mwst} with absolute maximum value of A_m.*
8. *The resulting matrix B_{mwst} is our preconditioner matrix B_2.*

Algorithm 4 Approximation by M-matrices (a), note the comments for the variant

Require: Stiffness matrix A

Ensure: Matrix B_3

1. ***for*** $i = 1$ *to* # *of rows in* A ***do***
2. $s_p(i) = 0$
3. ***for*** $j = 1$ *to* # *of columns in* A ***do***
4. ***if*** $i \neq j$ ***then***
5. ***if*** $a_{ij} > 0$ ***then***
6. $s_p(i) = s_p(i) + a_{ij}$
7. $a_{ij} = 0$
8. *The resulting matrix is M-matrix, say* A_m.
9. *Find MWST of* A_m.
10. ***for*** $i = 1$ *to* # *of rows in* A_m ***do***
11. $s_n(i) =$ *sum of row-wise dropped negative entries while finding MWST* A_m
12. $s_{n(A_m)}(i) =$ *sum of off-diagonal entries of* i^{th} *row of* A_m
13. $S(i) = s_p(i) + s_n(i)$
14. ***if*** $S(i) > 0$ $\{< 0$ *for the variant*$\}$ ***then***
15. ***for*** $k = 1$ *to* # *of rows in* A_m ***do***
16. ***if*** $i \neq k$ ***then***
17. $a^m_{ik} = a^m_{ik} + \{S(i) \times |\frac{a^m_{ik}}{s_{n(A_m)}(i)}|\}$
18. ***else***
19. $a^m_{ii} = a^m_{ii} + S(i)$
20. *The output matrix is* B_3 $\{B_3^v$ *for the variant*$\}$

Algorithm 5 Approximation by M-matrices (b)

Require: Stiffness matrix A

Ensure: Matrix B_4

1. *Find MWST of* A, *say* A_T.
2. ***for*** $i = 1$ *to* # *of rows in* A_T ***do***
3. $s_p(i) =$ *sum of row-wise dropped positive entries while finding MWST* A_T
4. $s_n(i) =$ *sum of row-wise dropped negative entries while finding MWST* A_T
5. $s_{n(A_T)}(i) =$ *sum of off-diagonal entries of* i^{th} *row of* A_T
6. $S(i) = s_p(i) + s_n(i)$
7. ***if*** $S(i) > 0$ ***then***
8. ***for*** $k = 1$ *to* # *of rows in* A_T ***do***
9. ***if*** $i \neq k$ ***then***
10. $a^T_{ik} = a^T_{ik} + \{S(i) \times |\frac{a^T_{ik}}{s_{n(A_T)}(i)}|\}$
11. ***else***
12. $a^T_{ii} = a^T_{ii} + S(i)$
13. *The output matrix is* B_4

Algorithm 6 Support Tree Preconditioner

Require: G_A (Graph of A)

Ensure: Preconditioner V

1. *Find a small edge separator that gives a balanced cut.*
2. *Assign the root of the tree to this separator.*
3. *Recursively build support trees for the components left after removing the separator.*
4. *Add edges from the root of the subtrees of the components to the root of the tree.*

Algorithm 7 Approximation by M-matrices (c)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

for $i = 1$ *to* # *of rows in* A_e ***do***
  for $j = 1$ *to* # *of columns in* A_e ***do***
    if $i \neq j$ ***then***
      if $a_{ij}^e > 0$ ***then***
        $a_{ij}^e = 0$
The output matrix is B_e.

Algorithm 8 Approximation by M-matrices (d)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

for $i = 1$ *to* # *of rows in* A_e ***do***
  for $j = 1$ *to* # *of columns in* A_e ***do***
    if $a_{ij}^e > 0$ ***then***
      $a_{ii}^e = a_{ii}^e + a_{ij}^e$
      $a_{ij}^e = 0$
The output matrix is B_e.

Algorithm 9 Approximation by M-matrices (e)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

for $i = 1$ *to* # *of rows in* A_e ***do***
  $s_p(i) = 0, s_n(i) = 0$
  for $j = 1$ *to* # *of columns in* A_e ***do***
    if $i \neq j$ ***then***
      if $a_{ij}^e > 0$ ***then***
        $s_p(i) = s_p(i) + a_{ij}^e$
        $a_{ij}^e = 0$
      if $a_{ij}^e < 0$ ***then***
        $s_n(i) = s_n(i) + a_{ij}^e$
  for $k = 1$ *to* # *of rows in* A_e ***do***
    if $i \neq k$ ***then***
      $a_{ik} = a_{ik} + \{s_p(i) \times |\frac{a_{ik}}{s_n(i)}|\}$
The output matrix is B_e.

Algorithm 10 Two Colors Approaches (a)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

for $i = 1$ *to* # *of rows in* A_{11} ***do***
  for $j = 1$ *to* # *of columns in* A_{11} ***do***
    $B_{11}(i, i) = A_{11}(i, i)$
The other blocks of B_e *are same as the blocks of* A_e *respectively*
The output matrix is B_e

Algorithm 11 Two Colors Approaches (b)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

1. ***for*** $i = 1$ *to # of rows in* A_{11} ***do***
2. $\quad s(i) = 0$
3. $\quad$ ***for*** $j = 1$ *to # of columns in* A_{11} ***do***
4. $\quad\quad s(i) = A_{11}(i,j) + s(i)$
5. $\quad B_{11}(i,i) = s(i)$
6. *The other blocks of* B_e *are same as the blocks of* A_e *respectively*
7. *The output matrix is* B_e

Algorithm 12 Two Colors Approaches (c)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

1. ***for*** $i = 1$ *to # of rows in* A_{11} ***do***
2. $\quad s(i) = 0$
3. $\quad$ ***for*** $j = 1$ *to # of columns in* A_{11} ***do***
4. $\quad\quad s(i) = abs(A_{11}(i,j)) + s(i)$
5. $\quad B_{11}(i,i) = s(i)$
6. *The other blocks of* B_e *are same as the blocks of* A_e *respectively*
7. *The output matrix is* B_e

Algorithm 13 Two Colors Approaches (d)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

1. ***for*** $i = 1$ *to # of rows in* A_{11} ***do***
2. $\quad s_1(i) = 0$
3. $\quad$ ***for*** $j = 1$ *to # of columns in* A_{11} ***do***
4. $\quad\quad$ ***if*** $i \neq j$ ***then***
5. $\quad\quad\quad s_1(i) = A_{11}(i,j) + s_1(i)$
6. $\quad\quad B_{11}(i,i) = A_{11}(i,i)$
7. ***for*** $i = 1$ *to # of rows in* A_{12} ***do***
8. $\quad s_2(i) = 0$
9. $\quad$ ***for*** $j = 1$ *to # of columns in* A_{12} ***do***
10. $\quad\quad$ ***if*** $i \neq j$ ***then***
11. $\quad\quad\quad s_2(i) = A_{12}(i,j) + s_2(i)$
12. $\quad$ ***for*** $k = 1$ *to # of columns in* A_{12} ***do***
13. $\quad\quad B_{12}(i,k) = A_{12}(i,k) + \{s_1(i) \times |\frac{A_{12}(i,k)}{s_2(i)}|\}$
14. *The other two blocks* B_{21}, B_{22} *are same as the blocks of* A_e *respectively*
15. *The output matrix is* B_e

Algorithm 14 Two Colors Approaches (e), note the comments for the variant

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

1. ***for*** $i = 1$ *to # of rows in* A_{11} ***do***
2. $s_1(i) = 0, s_2(i) = 0$
3. ***for*** $j = 1$ *to # of columns in* A_{11} ***do***
4. ***if*** $i \neq j$ ***then***
5. ***if*** $A_{11}(i,j) > 0$ ***then***
6. $s_1(i) = A_{11}(i,j) + s_1(i)$
7. ***else***
8. $s_2(i) = A_{11}(i,j) + s_2(i)$
9. $B_{11}(i,i) = A_{11}(i,i) + s_1(i)$ $\{s_2(i)$ *for the variant*$\}$
10. ***for*** $i = 1$ *to # of rows in* A_{12} ***do***
11. $s_3(i) = 0$
12. ***for*** $j = 1$ *to # of columns in* A_{12} ***do***
13. $s_3(i) = A_{12}(i,j) + s_3(i)$
14. ***for*** $k = 1$ *to # of columns in* A_{12} ***do***
15. $B_{12}(i,k) = A_{12}(i,k) + [s_2(i)$ $\{s_1(i)$ *for the variant*$\}$ $\times|\frac{A_{12}(i,k)}{s_3(i)}|]$
16. B_{21} *and* B_{22} *are same as* A_{21} *and* A_{22} *respectively*
17. The output matrix is B_e

Algorithm 15 Two Colors Approaches (f)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

1. ***for*** $i = 1$ *to # of rows in* A_{11} ***do***
2. $s(i) = 0$
3. ***for*** $j = 1$ *to # of columns in* A_{11} ***do***
4. ***if*** $i \neq j$ ***then***
5. ***if*** $A_{11}(i,j) > 0$ ***then***
6. $s(i) = A_{11}(i,j) + s(i)$
7. $A_{11}(i,j) = 0$
8. $B_{11}(i,j) = A_{11}(i,j)$
9. $B_{11}(i,j) = B_{11}(i,j) + s(i)$
10. The other blocks of B_e *are same as the blocks of* A_e *respectively*
11. The output matrix is B_e

Algorithm 16 Two Colors Approaches (g)

Require: Element stiffness matrix A_e
Ensure: Matrix B_e

1. ***for*** $i = 1$ *to* $\#$ *of rows in* A_{11} ***do***
2. $\quad s_1(i) = 0, s_2(i) = 0$
3. $\quad$ ***for*** $j = 1$ *to* $\#$ *of columns in* A_{11} ***do***
4. $\quad\quad s_1(i) = s_1(i) + abs(A_{11}(i,j))$
5. $\quad\quad$ ***if*** $A_{11}(i,j) < 0$ ***then***
6. $\quad\quad\quad s_2(i) = -2 \times abs(A_{11}(i,j)) + s_2(i)$
7. $\quad B_{11}(i,i) = s_1(i)$
8. ***for*** $i = 1$ *to* $\#$ *of rows in* A_{12} ***do***
9. $\quad s_3(i) = 0$
10. $\quad$ ***for*** $j = 1$ *to* $\#$ *of columns in* A_{12} ***do***
11. $\quad\quad s_3(i) = A_{12}(i,j) + s_3(i)$
12. $\quad$ ***for*** $k = 1$ *to* $\#$ *of columns in* A_{12} ***do***
13. $\quad\quad B_{12}(i,k) = A_{12}(i,k) + \{s_2(i) \times |\frac{A_{12}(i,k)}{s_3(i)}|\}$
14. $B_{21} = B_{12}^T$
15. *Let the resulting matrix after modifying the blocks* $A_{11}, A_{12},$ *and* A_{21} *be* E
16. ***for*** $i = (\#$ *of rows in* $B_{11} + 1)$ *to* $\#$ *of rows in* E ***do***
17. $\quad s_4(i) = 0$
18. $\quad$ ***for*** $j = 1$ *to* $\#$ *of columns in* E ***do***
19. $\quad\quad s_4(i) = s_4(i) + E(i,j)$
20. ***for*** $i = 1$ *to* $\#$ *of rows in* A_{22} ***do***
21. $\quad B_{22}(i,i) = A_{22}(i,i) - s_4(i + \#$ *of rows in* $B_{11})$
22. *The output matrix is* B_e