# Support Graph Preconditioners for Off-Lattice Cell-Based Models

Justin Steinman and Andreas Buttenschön

Department of Mathematics and Statistics, University of Massachusetts Amherst, Amherst 01003, United States of America

October 8, 2024

#### Abstract

Off-lattice agent-based models (or cell-based models) of multicellular systems are increasingly used to create in-silico models of in-vitro and in-vivo experimental setups of cells and tissues, such as cancer spheroids, neural crest cell migration, and liver lobules. These applications, which simulate thousands to millions of cells, require robust and efficient numerical methods. At their core, these models necessitate the solution of a large friction-dominated equation of motion, resulting in a sparse, symmetric, and positive definite matrix equation. The conjugate gradient method is employed to solve this problem, but this requires a good preconditioner for optimal performance.

In this study, we develop a graph-based preconditioning strategy that can be easily implemented in such agent-based models. Our approach centers on extending support graph preconditioners to block-structured matrices. We prove asymptotic bounds on the condition number of these preconditioned friction matrices. We then benchmark the conjugate gradient method with our support graph preconditioners and compare its performance to other common preconditioning strategies.

## 1 Introduction

Agent-based models that simulate individual entities such as humans, animals, or biological cells are an indispensable tool for studying emergent behaviors in complex systems. Over the last few decades, biomedical research has adopted agent-based models to develop digital-twins of in-vitro and in-vivo experiments on cell cultures and tissues [5]. To capture a wide variety of applications and research questions, many different agent-based models have been developed. Broadly, we categorize them into lattice-based (e.g. Cellular Automata [18] or Cellular Potts models [15]) and off-lattice models [25]. These different models each have advantages and disadvantages. For an overview, we refer the reader to the review by [38]. Here, we focus on off-lattice models closely related to colloidal physics [10]. In these models, cells are approximated by elastic spheroids [11, 14], capsules [6], ellipsoids [31], or surfaces of triangulated meshes [25]. The applications of these models are highly varied, including slime-mold aggregation [31], cancer growth and migration [23, 27], cancer monolayers and spheroids [11, 39], liver lobules [20], and neural crest cells [29].

A typical cell configuration of spherical cells with radii  $R_i$  is shown in Fig. 1. Advancing the simulation from  $t \to t + \Delta t$  requires solving the overdamped equation of motion  $\Gamma \mathbf{v} = \mathbf{F}$ , where  $\Gamma$  is the friction matrix,  $\mathbf{v}$  the cells' velocities, and  $\mathbf{F}$  the forces. The matrix  $\Gamma$  is block-structured, symmetric, and positive definite because the individual  $3 \times 3$  friction matrices are symmetric and positive definite [25]. There is a nonzero off-diagonal block in  $\Gamma$  at position (i,j) when cells i and j are in contact. We ultimately solve this large linear system using the conjugate gradient method.

Profiling our simulation software used in [6, 25, 39] shows that solving the equation of motion is often the most timeintensive step. We hypothesize that this is because no good preconditioners are available. From an implementation point of view, it is convenient to implement the linear algebra methods in a matrix-free manner. This means that "off the shelf" preconditioning techniques are difficult to use or adapt. Further, the condition number of the friction matrix  $\Gamma$  is determined by the cells' free surface area. This means that the condition number of the friction matrix varies during a given simulation. Additionally, the sparsity structure of  $\Gamma$  is dynamic because it encodes interacting cell pairs. This makes selecting a preconditioner difficult. Here, we solve this by using the *collision graph* constructed during the collision detection phase as our central data structure instead of the usual sparse matrix implementations.

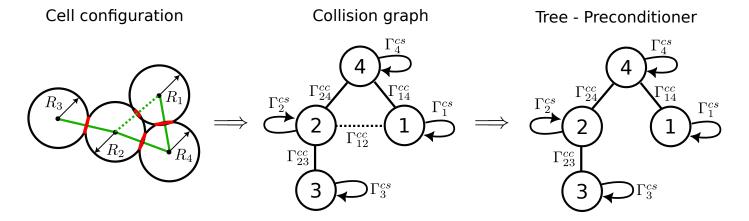


Figure 1: Graphical overview of our preconditioner construction. The figure illustrates the step-by-step construction of our proposed preconditioner, proceeding from left to right. 1. Agent configuration. The initial setup showing individual agents (cells) in a spatial arrangement. 2. The collision graph. Using collision detection algorithms, we construct a graph where nodes represent agents and edges represent collisions or friction interactions between them. The friction matrix is the graph laplacian of the collision graph. 3. The maximum spanning tree. Using Prim's algorithm, we construct a maximum spanning tree from the collision graph. The graph laplacian of this tree, is used as the preconditioner.

These observations, together with the increasing role agent-based models play in biomedical research, motivate our work. Our goal is to develop and benchmark a preconditioning strategy for the friction matrix  $\Gamma$  that is easily implemented in a matrix-free manner, and reduces the required computational time for solving the linear system.

## 1.1 Support Graph Preconditioners

The computational cost per iteration of the conjugate gradient method is dominated by the matrix-vector product. Thus, we aim to reduce the number of required iterations. Let  $\mathbf{e}_k$  denote the true error i.e. the difference between the computed approximation at the k-th iteration and the true solution. The well-known error estimate for the conjugate gradient method is given by [34]:

$$\|\mathbf{e}_k\|_{\Gamma} \le 2\left(\frac{\sqrt{\kappa(\Gamma)}-1}{\sqrt{\kappa(\Gamma)}+1}\right)^k \|\mathbf{e}_0\|_{\Gamma},$$

where  $\kappa(\Gamma)$  is the spectral condition number of  $\Gamma$ , which is a function of the contact areas and the ratio of the friction coefficients.

The error estimate suggests that a matrix with higher condition number requires more iterations. However, in practice, conjugate gradient convergence is more complex and often faster than this estimate suggests [26]. While this estimate has limited practical use, it does motivate the reduction of the condition number through preconditioning.

To precondition the system, we choose a symmetric matrix  $H = EE^T$  such that  $\Gamma^{-1} \sim H$ . We then solve the modified system  $E^T\Gamma E\hat{\mathbf{x}} = E^T\mathbf{F}$  and  $\mathbf{x} = E\hat{\mathbf{x}}$ . This approach reduces the iteration count if  $\kappa(E^T\Gamma E) < \kappa(\Gamma)$ , and reduces computational time if we choose H so that solving  $H\mathbf{x} = \mathbf{b}$  is computationally inexpensive.

Preconditioning is crucial in many problems, particularly those arising from the discretization of partial differential equations. While no unifying theory exists, it is a well-developed field [8, 16]. Our focus on a matrix-free implementation limits the direct application of many existing preconditioning techniques to our problem.

In off-lattice agent-based models, we can interpret the off-diagonal sparsity pattern of  $\Gamma$  as a graph  $\mathfrak{C}$  (see Fig. 1). This graph representation works as follows:

- Each cell at position  $\mathbf{r}_i$  is represented as the *i*-th vertex.
- We draw an edge e = (i, j) between vertices i and j when the cells' contact area  $A_{ij}$  is nonzero.
- The edges are weighted by the cell-cell friction matrices  $\Gamma_{ij}^{cc}$ .
- Cell-substrate matrices  $\Gamma_i^{cs}$  are represented with weighted self-loops.

The result is an undirected, matrix-weighted, and labeled (by cell id) graph that represents the friction matrix.

The relationship between matrices and graphs is well-established [33] and underpins many algorithms for sparse matrices. Typically, the matrix is constructed first, and its underlying graph is derived subsequently. Our approach reverses this process: we start with the collision graph constructed during the collision detection phase, and derive the friction matrix from it. Specifically, the friction matrix is the block Laplacian of the collision graph.

With employ a technique pioneered by Vaidya [37] that uses subgraphs of  $\mathfrak{C}$  as preconditioners. Subgraphs are advantageous because they are sparse, yet capture much of the relevant information from  $\mathfrak{C}$ . This characteristic allows them to effectively balance between approximating  $\Gamma$  and computational efficiency. In our implementation, we specifically use a maximum spanning tree. This tree can be constructed in linearithmic time, and its associated matrix can be factored in linear time. The broader study of using subgraphs as preconditioners is known as support graph theory.

Vaidya's original manuscript lacked many proofs, which were later provided by [3]. In our work, we extend these proofs to apply to block-structured matrices. This extension allows us to obtain estimates for the smallest and largest eigenvalues of such preconditioned systems. These eigenvalue bounds serve two important purposes: (1) They provide worst-case convergence estimates; and (2) They are valuable in implementing the robust conjugate gradient stopping criteria, proposed by [1, 30].

#### 1.2 Outline

The remainder of this paper is organized as follows: In Section 2, we introduce the linear system arising from agent-based models and its natural graph structure. This section provides the foundation for understanding the mathematical framework of our approach. Section 3 focuses on Vaidya's preconditioners. We explain how to construct these preconditioners for block-structured matrices and solve the resulting systems in near-linear time. This section bridges the gap between graph theory and numerical linear algebra. Section 4 presents our main theoretical contribution. Here, we extend support graph theory to block-structured matrices and derive eigenvalue bounds for the preconditioned linear system  $\Gamma$ . This extension is crucial for applying support graph theory to the matrices arising in agent-based models. In Section 5, we present our numerical results. We demonstrate the effectiveness of our preconditioner using a series of numerical benchmarks. These experiments validate our theoretical findings and showcase the practical benefits of our approach. Finally, Section 6 concludes the paper with a discussion of our findings, and its implications for real-world use cases. We situate our contributions in the broader theoretical landscape and discuss connections to the existing literature on this problem. Additionally, we provide potential directions for future research in this area.

## 2 Preliminaries

This section introduces the basic graph and matrix structure of our problem. From the collision detection phase of an offlattice simulation, we derive a matrix-vector equation whose solution represents the velocities of all the cells. We then show that the matrix we are solving is the block Laplacian of the collision graph, and we use this to prove positive definiteness. To set the stage for our subsequent discussion, we briefly outline the steps of an off-lattice model, considering a simulation of n spherical cells.

- Step 1: Broad-phase collision detection. Identify possible cell contact pairs (i, j). Efficient divide-and-conquer algorithms such as axis-aligned bounding boxes, are commonly used [36].
- Step 2: Compute forces between cells. Examine the possible cell pairs identified in the previous step, and identify interacting cells. Then, compute their contact area and contact force using a physical model. We denote the contact area between cells i and j by  $A_{ij}$ . For spherical cells, Hertz or JKR contact mechanics are commonly used [25].
- Step 3: Assemble the friction matrices. Construct the cell-cell and cell-substrate friction matrices. The cell-cell friction matrix between cells i and j is given by

$$\Gamma_{ij}^{cc} = A_{ij} \left( \gamma_{\parallel} \mathbf{u}_{ij} \otimes \mathbf{u}_{ij} + \gamma_{\perp} \left( I - \mathbf{u}_{ij} \otimes \mathbf{u}_{ij} \right) \right), \tag{2.1}$$

where  $\gamma_{\parallel}$  and  $\gamma_{\perp}$  are the parallel and perpendicular coefficients of friction, respectively (both of which are positive), and  $\mathbf{u}_{ij}$  is the unit contact vector [25]. If  $\mathbf{r}_i$  is the position of cell i, then

$$\mathbf{u}_{ij} = \frac{\mathbf{r}_j - \mathbf{r}_i}{\|\mathbf{r}_j - \mathbf{r}_i\|}.$$

The cell-substrate friction matrix has several possible forms depending on the cell shape. In the isotropic case of spherical cells, we can write  $\Gamma_i^{cs} = \lambda_{\text{med}} I$ , where  $\lambda_{\text{med}}$  is the coefficient of friction between the cell and the medium. For ellipsoidal cells, the cell-substrate friction matrix takes on a similar form to the cell-cell friction matrix in terms of directional friction coefficients and the unit direction vector. More complicated cell shapes may not be as easily expressible, but we only require that all the friction matrices are symmetric positive definite.

**Step 4:** Solve the equation of motion. The equation of motion for cell i is

$$\Gamma_i^{cs} \mathbf{v}_i + \sum_{A_{ij} > 0} \Gamma_{ij}^{cc} (\mathbf{v}_i - \mathbf{v}_j) = \mathbf{F}_i, \tag{2.2}$$

where  $\mathbf{v}_i$  is the velocity vector and  $\mathbf{F}_i$  is the total nonfrictional force acting on the cell. We interpret Equation (2.2) as one row of a large linear system  $\Gamma \mathbf{v} = \mathbf{F}$ . We show that the friction matrix  $\Gamma$  is symmetric positive definite. Hence, the conjugate gradient method is an efficient choice for obtaining an accurate solution.

**Step 5:** Update cell positions. Frequently, a forward Euler method is used:

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_i(t) + \Delta t \, \mathbf{v}_i,$$

where the step-size  $\Delta t$  is chosen according to the Euler's method stability criterion. Higher-order integration methods are rarely used. Two exceptions are PhysiCell [14], which employs a second-order Adam's-Bashford method with a fixed time-step, and [6], where an embedded Runga-Kutta-23 method with an adaptive time-step is employed.

Since the friction matrix  $\Gamma$  is comprised of  $3 \times 3$  blocks, we establish a few simple properties of these smaller friction matrices. Note that each cell-cell friction matrix (and the cell-substrate friction matrix of an ellipsoidal cell) is the sum of two orthogonal projectors. Let  $\gamma_{\text{max}}$  and  $\gamma_{\text{min}}$  be the maximum and minimum elements of  $\{\gamma_{\parallel}, \gamma_{\perp}\}$ , respectively.

**Lemma 2.1** Let  $\mathbf{u} \in \mathbb{R}^n$  be a unit vector, then the matrix

$$\Upsilon = \gamma_{\parallel} \mathbf{u} \otimes \mathbf{u} + \gamma_{\perp} (I - \mathbf{u} \otimes \mathbf{u}),$$

- 1. is symmetric positive definite;
- 2. its eigenvalues are  $\gamma_{\parallel}$  and  $\gamma_{\perp}$  with multiplicities 1 and 2 respectively;
- 3. its eigenspaces are  $E_{\gamma_{\parallel}} = \operatorname{span}(\mathbf{u})$  and  $E_{\gamma_{\perp}} = E_{\gamma_{\parallel}}^{\perp}$ ;
- 4. its operator norm (with respect to the 2-norm) is  $\|\Upsilon\| = \gamma_{\max}$ ; and
- 5. its condition number is

$$\kappa(\Upsilon) = \frac{\gamma_{\max}}{\gamma_{\min}}.$$

<u>Proof:</u> The projection matrices  $\mathbf{u} \otimes \mathbf{u}$  and  $I - \mathbf{u} \otimes \mathbf{u}$  are clearly symmetric. Note that  $(\mathbf{u} \otimes \mathbf{u})\mathbf{u} = \mathbf{u}$ , so  $\Upsilon \mathbf{u} = \gamma_{\parallel} \mathbf{u}$ . Since the eigenvectors of a real symmetric matrix are orthogonal, take  $\mathbf{w}$  such that  $\mathbf{w}^T \mathbf{u} = 0$ . Then  $(\mathbf{u} \otimes \mathbf{u})\mathbf{w} = 0$ , so  $\Upsilon \mathbf{w} = \gamma_{\perp} \mathbf{w}$ . The vector  $\mathbf{w}$  lies in the orthogonal complement of  $\mathbf{u}$ , which is two-dimensional. Both of the eigenvalues are positive, which gives positive definiteness. The operator norm of a symmetric positive definite matrix is the maximum eigenvalue.

Observe that the off-diagonal sparsity pattern of  $\Gamma$  is determined by the interacting cell pairs. We interpret this as a matrix-weighted graph.

**Definition 2.2** A matrix-weighted graph G = (V, E, w) is an undirected graph with a matrix-valued weight function  $w: V \times V \to \mathbb{R}^{d \times d}$ . We require that w(e) is positive definite for all  $e \in E$ , that w(v, v) is positive semidefinite for all  $v \in V$ , and that w(u, v) = 0 otherwise.

We allow nonzero weights on pairs (v,v) for convenience when representing cell-substrate friction. To refer to the number of edges |E| and the number of vertices |V|, we use m and n respectively. We reserve the Fraktur font for objects relating to the collision graph. Let  $\mathfrak{D}$  be the set of cells,  $\mathfrak{E}$  the set of interacting cell pairs, and  $\mathfrak{w}: \mathfrak{D} \times \mathfrak{D} \to \mathbb{R}^{3\times 3}$  the weight function with  $\mathfrak{w}(i,j) = \Gamma^{cc}_{ij}$  for all  $(i,j) \in \mathfrak{E}$  and  $\mathfrak{w}(i,i) = \Gamma^{cs}_i$  for all  $i \in \mathfrak{D}$ . These form the collision graph  $\mathfrak{C} = (\mathfrak{D}, \mathfrak{E}, \mathfrak{w})$ . This graph is typically very sparse with  $m = \mathcal{O}(n)$ .

Most of the matrices in this paper have a block structure. If A is an  $m \times n$  block matrix, then we mean that A is m blocks tall by n blocks wide. The precise size of the blocks is not important, but we assume they are square. To be explicit when indexing block matrices, we use an underline to index over blocks. For example, define a  $2 \times 2$  block matrix

$$A = \begin{pmatrix} W & X \\ Y & Z \end{pmatrix}.$$

Then  $A_{11} = W_{11}$  and  $\underline{A_{11}} = W$ . We also use this notation for vectors. If  $\mathbf{a} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$ , then  $\mathbf{a}_1 = \mathbf{x}_1$  and  $\underline{\mathbf{a}_1} = \mathbf{x}$ .

To formalize the relation between the friction matrix  $\Gamma$  and the collision graph  $\mathfrak{C}$ , we introduce the block Laplacian.

**Definition 2.3** A block Laplacian is a symmetric block matrix whose off-diagonal blocks are either zero or negative definite, and whose row and column sums are positive semidefinite. For a matrix-weighted graph G = (V, E, w), the block Laplacian of G is the matrix L where

$$\underline{L_{ij}} = \begin{cases} -w(i,j) & i \neq j, \\ w(i,i) + \sum_{k \neq i} w(i,k) & i = j. \end{cases}$$

It is easy to see that  $\Gamma$  is the block Laplacian of  $\mathfrak{C}$ . The following lemma establishes the definiteness needed to apply the conjugate gradient method.

**Lemma 2.4** All block Laplacians are positive semidefinite, and they are positive definite when their block row sums are positive definite.

**Proof:** Let A be a block Laplacian. For all  $\mathbf{x}$ ,

$$\mathbf{x}^{T} A \mathbf{x} = \sum_{i} \underline{\mathbf{x}}_{i}^{T} A_{ii} \underline{\mathbf{x}}_{i} + \sum_{i \neq j} \underline{\mathbf{x}}_{i}^{T} A_{ij} \underline{\mathbf{x}}_{j}$$

$$\geq \sum_{i \neq j} |\underline{\mathbf{x}}_{i}^{T} A_{ij} \underline{\mathbf{x}}_{i}| + \sum_{i \neq j} \underline{\mathbf{x}}_{i}^{T} A_{ij} \underline{\mathbf{x}}_{j}$$

$$= \sum_{i > j} \left( |\underline{\mathbf{x}}_{i}^{T} A_{ij} \underline{\mathbf{x}}_{i}| + |\underline{\mathbf{x}}_{j}^{T} A_{ij} \underline{\mathbf{x}}_{j}| + 2(\underline{\mathbf{x}}_{i}^{T} A_{ij} \underline{\mathbf{x}}_{j}) \right).$$

This inequality is strict when the block row sums are positive definite. By the AM-GM and Cauchy-Schwarz inequalities respectively,

$$|\mathbf{x}_i^T A_{ij} \mathbf{x}_i| + |\mathbf{x}_j^T A_{ij} \mathbf{x}_j| \ge 2\sqrt{|\mathbf{x}_i^T A_{ij} \mathbf{x}_i| \cdot |\mathbf{x}_j^T A_{ij} \mathbf{x}_j|} \ge 2|\mathbf{x}_i^T A_{ij} \mathbf{x}_j|.$$

This implies that each term in the summation above is nonnegative, which yields the desired result.

# 3 Support Graph Preconditioners

To effectively precondition  $\Gamma$ , we need to find an easily factorable matrix that closely approximates it. This section introduces support graph theory and Vaidya's preconditioners as tools to do this, along with efficient graph algorithms for their implementation. Typically, an underlying graph is derived from a given matrix. This graph is manipulated (e.g., by taking a subgraph) and its Laplacian is used as a preconditioner. However, since the friction matrix is derived from the collision detection phase, it is more appropriate for us to view  $\Gamma$  as the underlying matrix of  $\mathfrak{C}$ , and we can derive preconditioners from manipulations (e.g., subgraphs) of the collision graph. This is why support graph preconditioners are a natural choice for simulations. In fact, we can entirely avoid assembling matrices by working with the collision graph. All we need are the contact areas, normal vectors, and friction coefficients.

#### 3.1 Vaidva's Preconditioners

The first of Vaidya's preconditioners is the maximum spanning tree (MST) preconditioner. The idea is to precondition the Laplacian of a graph with the Laplacian of an MST. We work with trees because their Laplacians can be factored in linear time, and MSTs in particular because they capture a lot relevant information about the graph. In other words, an MST "supports" its parent graph well.

Since we are working with matrix-weighted graphs, we define an MST with respect to the minimum eigenvalues of the weights, a choice that is justified in the next section. In the case of  $\mathfrak{C}$ , this is equivalent to weighting by contact area. We also include the self-loops (i.e., the cell-substrate friction) in the MST. Let  $\mathfrak{T}$  be an MST of  $\mathfrak{C}$  and let P be its block Laplacian. We call  $\mathfrak{T}$  a support graph of  $\mathfrak{C}$  and P an MST preconditioner of  $\Gamma$ .

Vaidya's second class of preconditioners builds on the first by adding edges back to an MST. Given a parameter t, we split T into t disjoint subtrees of roughly the same size where each subtree has at most m/t vertices. Then we add the maximum weight edge in  $\mathfrak{C}$  between each pair of subtrees if they are connected in  $\mathfrak{C}$  but not in T. Let  $\mathfrak{T}'$  be the augmented support graph and P' its block Laplacian. We call P' an augmented MST preconditioner of  $\Gamma$ . The theoretically optimal value of t is approximately  $n^{1/4}$  [7].

The only step left to define is how we generate the support graph. Prim's algorithm is the best choice for finding an MST because it tells us, at no extra cost, how to permute the rows and columns of the block Laplacian to generate zero fill during factorization. We prove this in the next subsection. The time complexity of Prim's algorithm is  $\mathcal{O}(m \log n)$ . Augmenting an MST is straightforward once it has been decomposed into subtrees. A simple partitioning algorithm is presented in [7, TreePartition] and a more sophisticated one in [35].

In practice, it is often beneficial to give the preconditioner the same diagonal blocks as the original matrix. This has the intuitive benefit of capturing all the information contained in a Jacobi preconditioner. Doing so invalidates our analysis in the next section of the minimum eigenvalue of the preconditioned system, but it also decreases the maximum eigenvalue. While this choice is theoretically unfounded, it can be helpful in the reality of finite precision arithmetic (see Section 5).

#### 3.2 The Elimination Game

When factoring or performing Gaussian elimination on a matrix, new nonzero entries may be created, changing the sparsity pattern of the matrix. These new entries, called fill, require more memory and slow down computations. However, permuting the rows and columns of the matrix can change the amount of fill. There is a graphical interpretation of Gaussian elimination that shows how fill is created, called the elimination game [32].

In the game, all the vertices of a graph are eliminated in some order (e.g., see Fig. 2 where the elimination order is according to the vertex labels). When a vertex v is eliminated, fill edges are constructed so that every uneliminated neighbor of v (connected by either an original or fill edge) becomes pairwise adjacent. In other words, the uneliminated neighbors of v become a clique. The set of fill edges is in bijection with the set of fill entries that would be created during Gaussian elimination or decomposition. Note that the elimination game is never explicitly implemented but rather implicitly performed. Using this game, we can easily analyze the amount of fill created by a given permutation. The general problem of minimizing fill is NP-hard [40], but trees can easily be ordered to produce no fill.

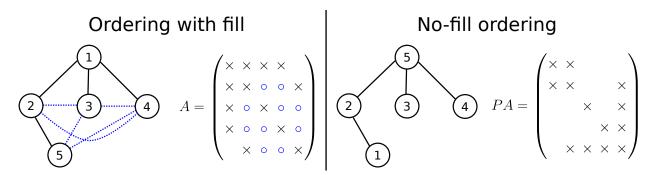


Figure 2: An example of the elimination game being played on the same graph with different orderings. Graph edges and matrix entries are in solid black and denoted by crosses respectively. Fill edges and entries are in dotted blue and blue circles respectively.

**Lemma 3.1** An ordering of the vertices of a rooted tree where no vertex occurs before any of its children produces no fill in the elimination game.

**Proof:** The first vertex eliminated must be a leaf. This produces no fill and yields another rooted tree. Inductively, the hypothesis demands that each vertex must be a leaf when it is eliminated, otherwise, it would have children to eliminate first. Since eliminating leaves produces no fill, we get the desired result.

Corollary 3.2 The reverse order in which vertices are added to the MST in Prim's algorithm produces no fill in the elimination game.

For augmented trees, reducing fill is far less simple. Reversing the traversal order of Prim's algorithm can yield very poor results. For example, if the added edges are between vertices that occur late in the traversal order, then lots of unnecessary fill is created. Of course, vertices that are not the ancestor of (we say a vertex is its own ancestor) an endpoint of an added edge can be eliminated as before without producing any fill. The rest of the vertices can be eliminated in the order specified by a fill-reducing algorithm. Both GENMMD and METIS are tested in [7], but it is worth mentioning that Chen and Toledo only work with matrices whose underlying graphs are regular meshes. Cell-based models tend to produce more irregularities in their graph structure. A variety of algorithms like reverse Cuthill-McKee may also perform well [12].

#### 3.3 Solving Preconditioners

Matrix inverses are almost never explicitly computed. Instead, large matrices are decomposed into the product of diagonal and triangular matrices in which solving systems is easy. For a symmetric positive definite matrix M, the Cholesky decomposition finds a lower triangular matrix L such that  $M = LL^T$ . We instead opt for the similar  $LDL^T$  decomposition in which D is a diagonal matrix and L has only ones on its diagonal. This is preferable for block matrices because it avoids computing matrix square roots. This section provides a general decomposition algorithm for symmetric positive definite block matrices and adapts it to the special case of nonsingular block Laplacians of trees.

#### Algorithm 1

It is worth restating this algorithm because we cannot take commutativity for granted as is often done.

```
1: function LDLT(A: n \times n symmetric positive definite block matrix)
 2:
           L, D \leftarrow n \times n block matrices
 3:
           for i \leftarrow 1, n do
                                           ▷ Current column
                X \leftarrow 0
 4:
                for j \leftarrow 1, \ldots, i-1 do
 5:
                     X \leftarrow X + \underline{L_{ij}D_{jj}L_{ij}}
 6:
                end for
 7:
                \underbrace{\frac{D_{ii}}{L_{ii}} \leftarrow \overset{-}{A}_{ii}}_{} - X
 8:
 9:
                \overline{\mathbf{for}} \ j \leftarrow i+1,\ldots,n \ \mathbf{do}
                                                              ▷ Current row
10:
                     Y \leftarrow 0
11:
                     for k \leftarrow 1, \dots, i-1 do
12:
                           Y \leftarrow Y + L_{ik}D_{kk}L_{jk}
                                                                     13:
                end for \underbrace{L_{ji} \leftarrow (\underline{A_{ji}} - Y)\underline{D_{ii}^{-1}}}_{\text{end for}} end for
14:
15:
16:
           end for
17:
18:
           return (L,D)
    end function
```

The general  $LDL^T$  algorithm takes  $\mathcal{O}(n^3)$  time, but we show that the preconditioners we want to factor are sparse and only take linear or near-linear time. Another perk of this algorithm is that L can be calculated in-place because previous entries in A are not reused.

**Theorem 3.3** Algorithm 2 returns the  $LDL^T$  factorization of the nonsingular block Laplacian of a matrix-weighted tree in  $\mathcal{O}(n)$  time. The sparsity pattern of L is the same as the lower triangle of the original matrix.

**Proof:** We assume the validity of Algorithm 1 and prove its equivalence to Algorithm 2 in this special case. Say that the rows and columns of A are ordered as specified on line 3 of Algorithm 2, so i < j means that vertex i is at least as far

#### Algorithm 2

```
1: function TREELDLT(A: nonsingular block Laplacian of a rooted tree)
             L, D \leftarrow n \times n block matrices
 2:
             for all vertices i in decreasing order of distance to the root do
 3:
                   X \leftarrow 0
 4:
                   for all children i of i do
 5:
                         X \leftarrow X + L_{ij}D_{jj}L_{ij}
 6:
                   end for
 7:
                   \underline{\frac{D_{ii}}{L_{ii}}} \leftarrow \underline{\frac{A_{ii}}{1}} - X
 8:
 9:

\frac{\underline{\underline{i}}_{i}}{\underline{j}} \leftarrow \text{parent of } i \\
\underline{\underline{L}_{ji}} \leftarrow \underline{\underline{A}_{ji}D_{ii}^{-1}} \\
\text{end for}

10:
11:
12:
             return (L, D)
13:
14: end function
```

from the root as vertex j. In this case, we say that i is younger than j and that j is older than i. Note that vertices i and j being connected is equivalent to  $\underline{A}_{ij}$  being nonzero. Lines 5–7 in both algorithms behave identically because the only vertices younger than i that are also connected to i are its children.

We want to show the equivalence of lines 10–16 in Algorithm 1 to lines 10–11 in Algorithm 2. We prove by induction on i that, in Algorithm 1,  $\underline{L}_{ji}$  is nonzero only if i is connected to j for all j > i. The hypothesis holds for the first vertex because Y is guaranteed to be zero as there are no younger vertices. Now assume that the hypothesis holds for all vertices younger than some i. If Y is nonzero, then there exists an older vertex j and a younger vertex k such that k is connected to both i and j. However, the only vertex older than k that it is connected to is its unique parent. This implies that i = j, which contradicts line 10. Therefore, Y is zero and the hypothesis is true, meaning the only j we need to consider is the parent of i.

When there is zero fill, no additional memory allocation is required for L even when using a matrix-free or sparse matrix implementation. However, this is not the case for augmented MST preconditioners. Algorithm 2 is no longer valid and extra fill entries need to be stored. Another benefit of the standard MST preconditioner is that solving systems in the decomposed block Laplacian takes linear time. We provide algorithms for this in Section B.1.

This begs the question of whether there exists an effective augmentation strategy that does not create fill. This restricts us to only add back edges between siblings in the MST. However, we suspect that this would not yield promising results for reasons we formalize in the next section. Intuitively, we want to add back edges that drastically reduce the distance between pairs of vertices, but sibling edges do a poor job of this.

Finally, it is known that block LU factorization for symmetric positive definite matrices is stable as long as the matrix is well-conditioned [9, 19].

# 4 Block-Structured Support Graph Theory

The existing literature on support graphs focuses entirely on symmetric diagonally dominant matrices. Here, we generalize a sequence of lemmas from [3] and [17] to work with block Laplacians as well, and we refer the reader to Appendix A for further generalizations to matrices with positive definite off-diagonal blocks. We provide proofs where they differ from the non-block case. The goal is to show that both of Vaidya's preconditioners achieve a minimum eigenvalue of at least 1, and that the condition number is  $\mathcal{O}(\kappa mn)$  with an MST preconditioner and  $\mathcal{O}(\kappa n^2/t^2)$  with an augmented MST preconditioner, where  $\kappa$  is the maximum condition number of all the edge weights and we assume sufficient sparsity.

We use  $\succeq$  to represent the Loewner order. For matrices A and B, we say  $A \succeq B$  if and only if A - B is positive semidefinite. A fact that we use without proof is that the Loewner order is a partial order on symmetric matrices. We denote the set of finite generalized eigenvalues of a pair of matrices (A, B) by  $\lambda(A, B)$ . That is to say,  $\lambda(A, B)$  is the set of numbers  $\lambda$  such that there exists a vector  $\mathbf{x}$  where  $A\mathbf{x} = \lambda B\mathbf{x}$ . If B is a preconditioner for A, then the condition number of the preconditioned system  $B^{-1}A$  is precisely the ratio of the extremal finite generalized eigenvalues  $\lambda_{\max}(A, B)/\lambda_{\min}(A, B)$ .

We start by defining the support of a pair of matrices, which bounds the maximum eigenvalue of the pair.

**Definition 4.1** The support of a pair of matrices (A, B) is

$$\sigma(A, B) = \min\{\tau \mid \tau B \succeq A\}.$$

If no such  $\tau$  exists, then we say  $\sigma(A, B) = \infty$ .

**Lemma 4.2** Suppose A and B are positive semidefinite matrices. Then

$$\lambda_{\max}(A, B) \le \sigma(A, B),$$

and equality holds when the support is finite.

**Proof:** See [17, Lemma 4.4]

Since  $\lambda_{\max}(B,A) = \lambda_{\min}(A,B)^{-1}$ , the supports  $\sigma(A,B)$  and  $\sigma(B,A)$  are all we need to bound the condition number  $\kappa(B^{-1}A)$ . In fact, we already have the necessary tools to bound the minimum eigenvalue.

**Theorem 4.3** Let G = (V, E, w) be a matrix-weighted graph with subgraph H = (V, F, w') and block Laplacians  $L_G$  and  $L_H$ . Then  $\lambda_{\min}(L_G, L_H) \geq 1$ .

**Proof:** Observe that  $L_G = L_H + L_K$  where  $L_K$  is the block Laplacian of the graph  $K = (V, E \setminus F, w - w')$ . Since  $L_G - L_H = L_K$  is a block Laplacian, it is positive semidefinite. This implies that  $\sigma(L_H, L_G) \leq 1$  and that  $\lambda_{\min}(L_G, L_H) \geq 1$ .

Bounding the maximum eigenvalue requires more effort. To simplify computing the support of a matrix and preconditioner, we use the following lemma to decompose the matrices into sums of positive semidefinite matrices.

**Lemma 4.4** Let  $A = A_1 + \cdots + A_k$  and  $B = B_1 + \cdots + B_k$  where each  $A_i$  and  $B_i$  is positive semidefinite. Then

$$\sigma(A, B) \le \max_{i} {\{\sigma(A_i, B_i)\}}.$$

**Proof:** See [17, Lemma 4.7]

A further simplification we can make is to only consider block Laplacians with zero block row sums. This means that we can ignore cell-substrate friction in the rest of the analysis using the following lemma.

**Lemma 4.5** Let A be a block Laplacian and define A' to be the matrix with the same off-diagonal blocks as A and zero block row sums. Let B' be a block matrix and B = B' + A - A'. If  $\beta B' \succeq A'$  for some  $\beta \geq 1$ , then  $\beta B \succeq A$ . Similarly, if  $\alpha A' \succeq B'$  for some  $\alpha \geq 1$ , then  $\alpha A \succeq B$ .

**Proof:** See [3, Lemma 2.5]

The lower bound on  $\alpha$  and  $\beta$  is not important because the pairs of matrices with which we are concerned have supports of at least 1. Next, we prove a small lemma that is helpful in the rest of the section.

Lemma 4.6 Let A be a symmetric positive semidefinite matrix. Then

$$\lambda_{\max}(A)I \succeq A \succeq \lambda_{\min}(A)I$$
.

**Proof:** Since A is symmetric positive semidefinite, it is diagonalizable and we can write  $A = PDP^{-1}$  where D is the diagonal matrix whose entries are the eigenvalues of A. For concision, let  $\lambda = \lambda_{\max}(A)$ . Observe

$$\lambda I - A = \lambda I - PDP^{-1} = P(\lambda I - D)P^{-1}.$$

By the definition of  $\lambda$ , the middle factor of  $\lambda I - D$  has all nonnegative entries, so the whole difference is positive semidefinite. Similar logic applies for the minimum eigenvalue.

Now we are ready to prove the main result. We decompose a matrix and support graph preconditioner into sums of the block Laplacians of individual edges and paths. Then we analyze their pairwise supports with the following three lemmas.

**Lemma 4.7** Suppose A and B are symmetric positive definite matrices. Let

$$\hat{A} = \begin{pmatrix} A & 0 & \cdots & 0 & -A \\ 0 & 0 & & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & 0 & 0 \\ -A & 0 & \cdots & 0 & A \end{pmatrix}, \quad and \quad \hat{B} = \begin{pmatrix} A & -A \\ -A & 2A & -A \\ & & \ddots & \\ & & -A & 2A & -A \\ & & & -A & A \end{pmatrix},$$

be  $(k+1) \times (k+1)$  block matrices. Then  $k\hat{B} \succeq \hat{A}$ .

**Proof:** Let  $C = k\hat{B} - A$  and define  $\hat{C} = \mathrm{diag}(A^{-1})C$  which is a block-structured matrix in which the blocks are either nonzero or a multiple of the identity matrix. Since  $\mathrm{diag}(A^{-1})$  is positive definite, it follows that C is semi-positive definite if  $\hat{C}$  is positive semidefinite. We prove that  $\hat{C}$  is positive semidefinite using induction as in the non-block-structured case in [3, Lemma 2.7]. The only difference is that we use a block-structured symmetric Gaussian elimination, meaning the i-th elimination step is

$$\hat{C}_i = E_i \hat{C}_{i-1} E_i^T$$

where  $E_i$  is block-structured with identity blocks along its diagonal and two nonzero off-diagonal blocks:

$$\underline{E_{1i}} = \frac{1}{1+i}I, \ \underline{E_{(i+1)(i)}} = \frac{1}{i+1}I.$$

At completion of this process we obtain the matrix

$$\hat{C} = \operatorname{diag}\left(0, 2kI, \frac{3k}{2}I, \dots, \left(\frac{i+1}{i}\right)kI, \dots, 0\right).$$

Since the matrix  $\hat{C}$  has nonnegative values on its diagonal, this shows that  $\hat{C}$  is positive semidefinite.

Lemma 4.8 Suppose A and B are symmetric positive definite matrices. Let

$$\hat{A} = \begin{pmatrix} A & 0 & \cdots & 0 & -A \\ 0 & 0 & & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & & 0 & 0 \\ -A & 0 & \cdots & 0 & A \end{pmatrix}, \quad and \quad \hat{B} = \begin{pmatrix} B & -B \\ -B & 2B & -B \\ & & \ddots & \\ & & -B & 2B & -B \\ & & & -B & B \end{pmatrix},$$

be  $(k+1) \times (k+1)$  block matrices. Then  $k \cdot \lambda_{\max}(AB^{-1})\hat{B} \succeq \hat{A}$ .

<u>Proof:</u> Let  $\lambda = \lambda_{\max}(AB^{-1})$ . When A = B, we have that  $\lambda = 1$  and the statement is equivalent to Lemma 4.7. When  $A \neq B$ , we can reduce to the case of equality by multiplying  $\hat{B}$  by diag $(AB^{-1})$ . This yields

$$k \cdot \operatorname{diag}(AB^{-1})\hat{B} \succeq \hat{A}.$$

Next we show that  $k \cdot \lambda \hat{B} \succeq k \cdot \text{diag}(AB^{-1})\hat{B}$ . This is equivalent to showing that  $\lambda I \succeq AB^{-1}$ , which is true by Lemma 4.6. The rest follows from the transitivity of the Loewner order.

Note that, up to permutation, the block Laplacian of a single edge looks like  $\hat{A}$  and that of a simple path with uniform edge weights looks like  $\hat{B}$ . The following congestion-dilation lemma generalizes the previous one so that  $\hat{B}$  can have varied edge weights.

#### Lemma 4.9 (congestion-dilation lemma) Let

$$\hat{A} = \begin{pmatrix} A & 0 & \cdots & 0 & -A \\ 0 & 0 & & & 0 \\ \vdots & & \ddots & & \vdots \\ 0 & & & 0 & 0 \\ -A & 0 & \cdots & 0 & A \end{pmatrix}$$

and

$$\hat{B} = \begin{pmatrix} C_1 & -B_1 \\ -B_1 & C_2 & -B_2 \\ & \ddots & \\ & -B_{k-1} & C_k & -B_k \\ & & -B_k & C_{k+1} \end{pmatrix}$$

be  $(k+1) \times (k+1)$  block matrices where A,  $B_i$ , and  $C_i$  are symmetric positive definite for all i and  $\hat{B}$  has zero block row sums. Then

$$k \cdot \max_{i} \{\lambda_{\max}(AB_i^{-1})\} \cdot \hat{B} \succeq \hat{A}.$$

**<u>Proof:</u>** Let  $D = \min_i \{\lambda_{\min}(B_i)\} \cdot I$ . Decompose

$$\hat{B} = \hat{B}_1 + \hat{B}_2 = \begin{pmatrix} D & -D & & & \\ -D & 2D & -D & & & \\ & \ddots & & & \\ & -D & 2D & -D & \\ & & -D & D \end{pmatrix} + \begin{pmatrix} C_1 - D & -B_1 + D & & \\ -B_1 + D & C_2 - 2D & -B_2 + D & & \\ & & \ddots & & \\ & & -B_{k-1} + D & C_k - 2D & -B_k + D \\ & & -B_k + D & C_{k+1} - D \end{pmatrix}.$$

Let  $\lambda = \max_i \{\lambda_{\max}(AB_i^{-1})\}$  and write

$$k \cdot \lambda \hat{B} - \hat{A} = (k \cdot \lambda \hat{B}_1 - \hat{A}) + (k \cdot \lambda \hat{B}_2).$$

The first summand is positive semidefinite by Lemma 4.8. The diagonal blocks of  $\hat{B}_2$  are positive semidefinite and the nonzero off-diagonal blocks are negative semidefinite by Lemma 4.6. This and the fact that the block row sums are zero means the second summand is a block Laplacian, so it is positive semidefinite.

Under the support of the path represented by  $\hat{B}$ , we call  $\lambda$  the *congestion* of the edge represented by  $\hat{A}$ , and k is its dilation. A more concise statement of the lemma is that  $\sigma(\hat{A}, \hat{B})$  is bounded above by the product of the congestion and dilation. We use the this lemma to prove an upper bound on the maximum eigenvalue of a block Laplacian with an MST preconditioner. The following proofs make use of the fact that

$$\frac{\lambda_{\max}(A)}{\min_i \{\lambda_{\min}(B_i)\}} \ge \max_i \{\lambda_{\max}(AB_i^{-1})\}.$$

**Theorem 4.10** Let G = (V, E, w) be a matrix-weighted graph with MST T weighted by minimum eigenvalues. Let  $L_G$  and  $L_T$  be their block Laplacians and let  $\kappa$  be the maximum condition number of all the edge weights in G. Then

$$\lambda_{\max}(L_G, L_T) \le \kappa m(n-1).$$

**Proof:** For every edge e = (u, v), let p(e) be the path in T from u to v that uses at least 1/m fraction of each edge weight. We can write

$$L_G = \sum_{e \in E} L_e$$
 and  $L_T = \sum_{e \in E} L_{p(e)}$ 

where  $L_e$  and  $L_{p(e)}$  are the block Laplacians of the edges and paths respectively. By Lemma 4.4,

$$\sigma(L_G, L_T) \le \max_{e \in E} \{ \sigma(L_e, L_{p(e)}) \}.$$

The maximum possible length (edge count) of each p(e) is n-1. This is the dilation. Since T is an MST, the minimum eigenvalue of each edge weight in p(e) is at least that of w(e). This means that the congestion is at most  $\kappa m$ . The congestion-dilation lemma yields the desired result.

Augmented MST preconditioners have a better upper bound that can be proven similarly.

**Theorem 4.11** Let G = (V, E, w) be a matrix-weighted graph with augmented MST T' as described in Section 3.1. Let  $L_G$  and  $L_{T'}$  be their block Laplacians and let  $\kappa$  be the maximum condition number of all the edge weights in G. If every vertex has at most d neighbors, then

$$\lambda_{\max}(L_G, L_{T'}) \le \frac{2\kappa d^3 n^2}{t^2}.$$

**Proof:** We perform a similar decomposition to the previous proof. Let  $S_1, \ldots, S_t$  be the subtrees of T'. For each edge e = (u, v), a path from u to v in T' is not necessarily unique. If both u and v are in the same  $S_i$ , let p(e) be the unique path from u to v contained in  $S_i$ . If u is in  $S_i$  and v is in  $S_j$  with  $i \neq j$ , let p(e) be the concatenation of the following paths: the unique path in  $S_i$  from u to the endpoint of the edge that connects  $S_i$  and  $S_j$ , that edge itself, and the unique path in  $S_j$  from the other endpoint to v.

Now we must decide what fraction of each edge weight to use. Each edge e in  $S_i$  can be in a support path from any of the dn/t vertices in  $S_i$  to any of their d neighbors. This is  $d^2n/t$  total paths, so we use  $t/(d^2n)$  fraction of each edge weight. Following the same logic as in the previous proof, we get that the congestion is at most  $\kappa(d^2n/t)$ .

In the worst case, one of these paths may go across all dn/t - 1 edges in one subtree, the edge connecting it to another subtree, and all dn/t - 1 edges in the other subtree. Therefore, the dilation is less than 2(dn/t) and the rest follows from the congestion-dilation.

In the context of collision graphs,  $\kappa$  is simply  $\max_{i,j} \{A_{ij}\} \cdot \gamma_{\max}/\gamma_{\min}$ . Since the minimum eigenvalue is at least 1, the preceding bounds on the maximum eigenvalue are also bounds on the condition number of the preconditioned system.

## 5 Numerical Benchmarks

In the previous sections, we extended support graph theory to block-structured matrices, introduced the collision graph, and argued for constructing preconditioners directly from the collision graph. Although we derived theoretical bounds on the eigenvalues of the preconditioned system, past experience with using the conjugate gradient method in finite precision arithmetic demonstrates that theoretical estimates often do not accurately predict performance in practice [26]. For this reason, we conduct benchmarking experiments to compare five preconditioning strategies: (1) no preconditioner, (2) the Jacobi preconditioner (i.e., the diagonal of  $\Gamma$ ), (3) the block Jacobi preconditioner (i.e., the block diagonals of  $\Gamma$ ), (4) the support graph MST preconditioner (as introduced in Section 3), and (5) a modified support graph MST preconditioner which uses the same block diagonals as the original matrix  $\Gamma$ .

#### 5.1 Experimental Setup

To test the preconditioning strategies, we generate two types of simulation-like test cases. Our experiments simulate of elastic spherical cells of radius r=0.5 (units are in 10s of  $\mu m$ ). The Hertz contact model is used to resolve cell-cell contact areas and forces. We assume that the cell-substrate friction matrix is diagonal (i.e.,  $\Gamma^{cs} = \gamma_{\rm med} I$ ), and the cell-cell friction matrix is given by equation (2.1). We consider both cases in which  $\gamma_{\parallel} = \gamma_{\perp}$  and  $\gamma_{\parallel} \neq \gamma_{\perp}$ . We find that cases in which  $\gamma_{\parallel} \neq \gamma_{\perp}$  require more iterations for convergence. Typically the cell-cell friction coefficients are between 1–3 orders of magnitude greater than the cell-substrate friction coefficients [?]. We consider two types of spatial configurations for the cells:

- 1. Random placement: Cells are randomly placed in a 3D-sphere to simulate noisy cell configurations.
- 2. Hexagonal close packing: Each cell is surrounded by 12 other cells. Additionally, we apply normally distributed noise with mean zero and fixed standard deviation to vary the edge-vertex ratio.

The random configurations are created by randomly sampling locations in a 3D-sphere of given radius and only accepting locations that are a minimum distance from previously placed cells.

#### 5.2 Benchmarking Results

In Fig. 3, we report the observed mean convergence behavior of the conjugate gradient method for the friction matrix of N = 10000 randomly placed cells in a 3D-sphere. The relative error is computed by comparing the conjugate gradient solution to the solution obtained from a sparse direct method.

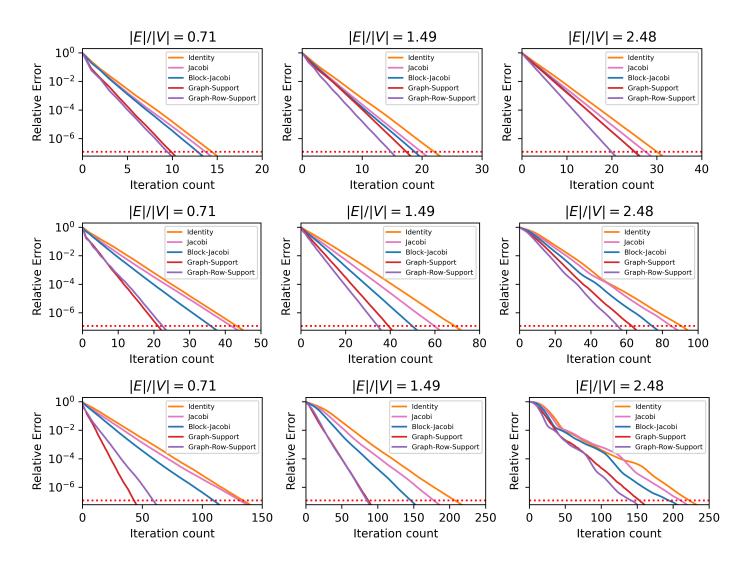


Figure 3: The average conjugate gradient convergence of 25 experiments, each of N=10000 randomly placed cells in a large 3D-sphere. The resulting block-structured linear systems are solved using the conjugate gradient method with five different preconditioning strategies: (1) the identity; (2) Jacobi preconditioner; (3) block Jacobi preconditioner; (4) the support-graph preconditioner; and (5) the row-support graph preconditioner. In the top row  $\gamma_{\rm med}=3\times10^5$ ,  $\gamma_{\parallel}=2\times10^6$  and  $\gamma_{\perp}=8\times10^6$ . In the middle row  $\gamma_{\rm med}=3\times10^4$ ,  $\gamma_{\parallel}=2\times10^6$  and  $\gamma_{\perp}=8\times10^6$ . In the bottom row  $\gamma_{\rm med}=3\times10^4$ ,  $\gamma_{\parallel}=2\times10^7$  and  $\gamma_{\perp}=8\times10^7$ .

Rows: The difference between the cell-substrate and cell-cell friction increases from single order of magnitude to three orders of magnitude across rows. We observe that more iterations are required for larger order of magnitude differences.

**Columns:** The ratio of edges to vertices in the collision graph is increasing from left to right. We observe that the performance advantage of the support graph preconditioners becomes less pronounced as the edge-vertex ratio increases.

For low edge-vertex ratios, the support graph preconditioners result in approximately a 50% iteration count reduction. To explore the degradation in convergence rate of the support graph preconditioner as the edge-vertex ratio increases, we construct examples with high edge-vertex ratios. We generate cell configurations on three dimensional hexagonal lattices, so that each cell has 12 neighbors (equivalent to an edge-vertex ratio of six). The convergence results for these cases are reported in the last column of Fig. 4. In the first two rows of Fig. 4, there is a two order of magnitude difference between the cell-substrate friction and the cell-cell friction, while in the last row there is a three order of magnitude difference. We observe that the performance of the different preconditioners is comparable with the support graph preconditioner performing the worst. In the last row, for larger order of magnitude differences between cell-substrate and cell-cell friction,

this difference is more pronounced.

To see whether we recover the support graph preconditioner performance as the edge-vertex ratio decreases, we add a normally distributed random variable with mean zero and standard deviation  $\sigma$  to each cell location. As the standard deviation increases, the edge-vertex ratio decreases, and the support graph preconditioners perform best once more. In the cases with high edge-vertex ratios, the block Jacobi preconditioner outperforms the support graph preconditioner. This observation originally motivated us to consider the row-support graph preconditioners in which we use the block diagonals of the friction matrix together with the off-diagonals of the MST preconditioner.

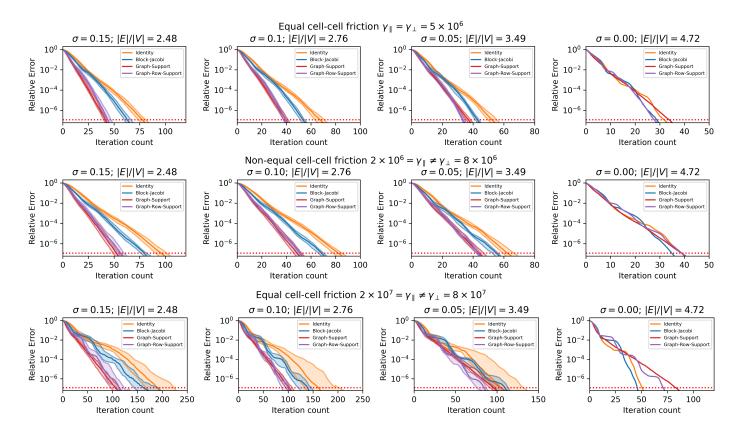


Figure 4: Cells arranged on a three-dimensional hexagonal lattice with normal noise with mean zero and standard deviation  $\sigma$  applied to each hexagonal cell position. In these examples we have N=309 cells. For each configuration we report the number of edges per vertex. As the edge to vertex ratio increases, the rate of convergence of both support graph preconditioners approaches that of the block-Jacobi preconditioners. For the hexagonal configuration without noise, where each cell has 12 neighbours the rate of convergence for all preconditioners is comparable. Here  $\gamma_{\rm med}=3\times10^4$ .

### 6 Discussion

We have proposed efficient preconditioners for linear systems arising from off-lattice agent-based models. Using the notions of matrix-weighted graphs and block Laplacians, we extended support graph theory to this problem. By using maximum spanning trees, we obtained preconditioners that are computationally efficient to compute and factor, while significantly decreasing the condition number and iteration count in the conjugate gradient method. We proved bounds on the condition number that, while overly pessimistic in practice, demonstrate asymptotic stability. Additionally, we proved a bound for augmented spanning trees, but did not implement them because augmented trees lead to fill-in during factorization. In such cases, it may be beneficial to use the augmented MST preconditioner with an incomplete  $LDL^T$  decomposition that completely ignores fill, however further research would be needed to develop a fill-reducing ordering. Finally, while we have not proved any bounds that theoretically justify including the entire block diagonal of the original matrix in the preconditioner, we provided numerical evidence of its potential helpfulness.

We benchmarked our proposed preconditioners and compared them to the identity, Jacobi, and block Jacobi preconditioners. Support graph preconditioners showed significant advantages in scenarios with low edge-vertex ratios, but their effectiveness decreased as this ratio increased. We observed that in situations where the support graph did not perform well, the block Jacobi preconditioner performed well. These findings led to the development of row-support graph preconditioners, combining block diagonal and support graph approaches. This combination resulted in a preconditioner which shares properties with both the block Jacobi and MST preconditioner. Our benchmarking supports this observation, as the row-support graph preconditioner performs well across various scenarios. Finally, our results highlight the importance of benchmarking, as theoretical bounds frequently do not predict practical performance in finite precision arithmetic.

What are the key takeaways for real-world implementations? Since the collision graph is a byproduct of the collision detection algorithm, it is logical to utilize it to define the system's friction matrix. The collision graph allows for elegant implementations of the required linear algebra operations and the construction of an MST preconditioner. While our theoretical results give us asymptotic bounds on the eigenvalues of the preconditioned system, and numerical exploration indeed suggests a significant reduction in condition number, the observed number of required iterations in finite precision arithmetic paints a more complicated picture. The choice of preconditioner is problem-specific. The key considerations are the system's edge-vertex ratio and magnitude difference between cell-substrate and cell-cell friction. The larger the difference between cell-substrate and cell-cell friction, the more useful preconditioning becomes. Similarly, the lower the edge-vertex ratio, the larger the reduction in required iterations when using the support graph preconditioner. Our results suggest that in practice, the edge-vertex ratio could be used to switch between support graph and block Jacobi preconditioning, or we could use the row-support graph preconditioner.

This paper generalizes and applies the earliest results in support graph theory. While Vaidya's preconditioners are very efficient to compute, more sophisticated preconditioners achieve much better condition numbers and near-linear time convergence. The main tool they employ is the low-stretch spanning tree. These are the basis of Spielman's groundbreaking paper [35] that uses augmented low-stretch spanning trees as preconditioners. We can generalize the notion of stretch to matrix-weighted graphs by examining the minimum eigenvalues of the edge weights, similar to our approach for MSTs. Substantial progress has been made in solving symmetric diagonally dominant systems with preconditioners derived from low-stretch spanning trees, most recently in [21] and [13]. Additionally, with a low-stretch spanning tree, we can implement combinatorial algorithms like those described in [22] and [24] that do not use the conjugate gradient method at all. We expect that all results from the existing literature can be generalized to block-structured matrices by adding a factor of  $\kappa$  to the condition number bounds.

# A Positive Definite Off-Diagonal Blocks

An ostensible limitation of the theory presented in this paper is that it only works with block Laplacians, requiring offdiagonal blocks to be zero or negative definite. This section describes methods of working with positive definite off-diagonal blocks as well.

There are two established ways of handling positive off-diagonal entries in the non-block case. The first is a reduction due to Gremban [17, Lemma 7.3] which is also described concisely in [35, Appendix A] and [28, Appendix A.2]. The validity of this reduction immediately transfers to the block case. Let A be a block matrix whose off-diagonal blocks are either zero or definite (positive or negative), and for every row i,

$$\underline{A_{ii}} \succeq \sum_{j \neq i} |\underline{A_{ij}}|$$

where  $|\cdot|$  leaves positive semidefinite matrices unchanged and negates negative semidefinite ones. This matrix is positive semidefinite by the proof from Lemma 2.4. We may call this a generalized block Laplacian, possibly originating from a similarly defined generalized matrix-weighted graph. Then  $A = D + A^{(+)} + A^{(-)}$  where D contains the diagonal blocks,  $A^{(+)}$  the positive definite off-diagonal blocks, and  $A^{(-)}$  the negative definite ones. To solve the system  $A\mathbf{x} = \mathbf{b}$ , we construct a  $2n \times 2n$  block Laplacian system

$$A'\begin{pmatrix}\mathbf{x}_1\\\mathbf{x}_2\end{pmatrix}=\begin{pmatrix}\mathbf{b}\\-\mathbf{b}\end{pmatrix}\quad\text{where}\quad A'=\begin{pmatrix}D+A^{(-)}&-A^{(+)}\\-A^{(+)}&D+A^{(-)}\end{pmatrix}.$$

The desired solution is then  $\mathbf{x} = (\mathbf{x}_1 - \mathbf{x}_2)/2$ . Thanks to the simplicity of this reduction, the condition number analysis from earlier still applies and  $\epsilon$ -approximate solutions to the larger system produce  $\epsilon$ -approximate solutions to the original one.

The other method of solving a generalized Laplacian system is with a maximum weight basis preconditioner [4]. This is a considerably more complicated technique that requires additional analysis of the condition number, but it does not require a reduction to a larger problem. We have not proven any analogues for the block case.

# B Assorted Graph Algorithms

### B.1 Directly Solving Systems from Trees

Once the  $LDL^T$  decomposition of an MST preconditioner P is computed, we need to repeatedly solve systems  $P\mathbf{x} = \mathbf{b}$ . This section describes how to do this in terms of the implicit tree structure of P (i.e., we do not distinguish between the indices of rows and columns and the vertices they represent). We assume the rows and columns of P are ordered as described in Lemma 3.1 so that L has the same sparsity pattern as the lower triangle of P. Let d be the order of the blocks of P.

Step 1: Forward substitution. Define  $\mathbf{z} = DL^T\mathbf{x}$  and solve  $L\mathbf{z} = \mathbf{b}$ , and proceed as in algorithm 3. The cost is n-1 matrix-vector multiplications of  $d \times d$  matrices.

### Algorithm 3

```
\begin{array}{l} \textbf{function} \ \mathsf{FORWARDSOLVE}(L, \mathbf{b}) \\ \textbf{for} \ i \leftarrow 1, \dots, n \ \textbf{do} \\ \underline{\mathbf{z}_i} \leftarrow \underline{\mathbf{b}_i} \\ \textbf{for} \ \textbf{all} \ \textbf{children} \ j \ \textbf{of} \ i \ \textbf{do} \\ \underline{\mathbf{z}_i} \leftarrow \underline{\mathbf{z}_i} - \underline{L_{ji}} \underline{\mathbf{z}_j} \\ \textbf{end} \ \textbf{for} \\ \textbf{end} \ \textbf{for} \\ \textbf{return} \ \mathbf{z} \\ \textbf{end} \ \textbf{function} \end{array}
```

Step 2: Block diagonal solve. Define  $\mathbf{y} = L^T \mathbf{x}$  and solve  $D\mathbf{y} = \mathbf{z}$ . The cost is n solves of  $d \times d$  matrices.

#### Algorithm 4

```
\begin{array}{l} \textbf{function} \ \ \text{DiagSolve}(D, \, \mathbf{z}) \\ \textbf{for} \ i \leftarrow 1, \dots, n \ \textbf{do} \\ \underline{\mathbf{y}_i} \leftarrow \underline{D_{ii}^{-1} \mathbf{z}_i} \\ \textbf{end} \ \textbf{for} \\ \textbf{return} \ \mathbf{y} \\ \textbf{end} \ \textbf{function} \end{array}
```

Step 3: <u>Backward substitution</u>. Solve  $L^T \mathbf{x} = \mathbf{y}$ . The cost is n-1 matrix-vector multiplications of  $d \times d$  matrices.

### Algorithm 5

```
\begin{array}{l} \textbf{function} \ \text{BackwardSolve}(L, \mathbf{y}) \\ \textbf{for} \ i \leftarrow n, \dots, 1 \ \textbf{do} \\ j \leftarrow \text{parent of} \ i \\ \underline{\mathbf{x}_i} \leftarrow \underline{\mathbf{y}_i} - L_{ij}^T \underline{\mathbf{x}_j} \\ \textbf{end for} \\ \textbf{return} \ x \\ \textbf{end function} \end{array}
```

# B.2 Matrix-Vector Product of a graph

Lastly, we give an algorithm for computing matrix-vector products with the block Laplacian of a matrix-weighted graph. This is needed for a matrix-free implementation of the conjugate gradient method.

### Algorithm 6

```
\begin{array}{l} \textbf{function} \ \operatorname{MatVec}(G = (V, E, w), \, \mathbf{v}) \\ \textbf{for} \ i \leftarrow 1, \dots n \ \mathbf{do} \\ \underline{\mathbf{x}_i} \leftarrow w(i, i) \underline{\mathbf{v}_i} \\ \textbf{end} \ \textbf{for} \\ \textbf{for} \ (i, j) \in E \ \mathbf{do} \\ \underline{\mathbf{x}_i} \leftarrow \underline{\mathbf{x}_i} + w(i, j) (\underline{\mathbf{v}_i} - \underline{\mathbf{v}_j}) \\ \underline{\mathbf{x}_j} \leftarrow \underline{\mathbf{x}_j} + w(i, j) (\underline{\mathbf{v}_j} - \underline{\mathbf{v}_i}) \\ \textbf{end} \ \textbf{for} \\ \textbf{end} \ \textbf{for} \\ \end{array}
```

# References

- [1] Owe Axelsson and Igor Kaporin. Error norm estimation and stopping criteria in preconditioned conjugate gradient iterations. Numerical Linear Algebra with Applications, 8(4):265–286, 2001.
- [2] Richard Barrett, Michael Berry, Tony F Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk Van der Vorst. <u>Templates for the solution of linear systems: building blocks for iterative methods</u>. SIAM, 1994.
- [3] Marshall Bern, John R Gilbert, Bruce Hendrickson, Nhat Nguyen, and Sivan Toledo. Support-graph preconditioners. SIAM Journal on Matrix Analysis and Applications, 27(4):930–951, 2006.
- [4] Erik Boman, Doron Chen, and Bruce Hendrickson. Maximum-weight-basis preconditioners. <u>Numerical Linear Algebra</u> with Applications, 11:695 721, 10 2004.
- [5] Andreas Buttenschön and Leah Edelstein-Keshet. Bridging from single to collective cell migration: A review of models and links to experiments. PLoS computational biology, 16(12):e1008411, 2020.
- [6] Andreas Buttenschön, Paul van Liedekerke, Margriet Palm, and Dirk Drasdo. Does single cell migration behavior permit prediction of multi-cellular migration patterns: Lessons from a physics-based model., 2025.
- [7] Doron Chen and Sivan Toledo. Vaidya's preconditioners: Implementation and experimental study. <u>Electronic</u> Transactions on Numerical Analysis, 16(9):03, 2003.
- [8] James W Demmel. Applied numerical linear algebra. SIAM, 1997.
- [9] James W Demmel, Nicholas J Higham, and Robert S Schreiber. Stability of block lu factorization. <u>Numerical linear algebra with applications</u>, 2(2):173–190, 1995.
- [10] Dirk Drasdo. Coarse graining in simulated cell populations. Adv. Complex Syst., 08(02n03):319–363, 2005.
- [11] Dirk Drasdo and Stefan Höhme. A single-cell-based model of tumor growth in vitro: monolayers and spheroids. Phys. Biol., 2(3):133–147, 12 July 2005.
- [12] Iain Duff and Gérard Meurant. The effect of ordering on preconditioned conjugate gradient. BIT, 29, 12 1989.
- [13] Yuan Gao, Rasmus Kyng, and Daniel A Spielman. Robust and practical solution of laplacian equations by approximate elimination. arXiv preprint arXiv:2303.00709, 2023.
- [14] Ahmadreza Ghaffarizadeh, Randy Heiland, Samuel H Friedman, Shannon M Mumenthaler, and Paul Macklin. Physicell: An open source physics-based cell simulator for 3-d multicellular systems. PLoS computational biology, 14(2):e1005991, 2018.

- [15] François Graner and James A Glazier. Simulation of biological cell sorting using a two-dimensional extended potts model. Physical review letters, 69(13):2013, 1992.
- [16] Anne Greenbaum. <u>Iterative methods for solving linear systems</u>. SIAM, 1997.
- [17] Keith D Gremban. <u>Combinatorial preconditioners for sparse, symmetric, diagonally dominant linear systems</u>. PhD thesis, Carnegie Mellon University, 1996.
- [18] Karl-Peter Hadeler and Johannes Müller. Cellular automata: analysis and applications. Springer, 2017.
- [19] Nicholas J Higham. Accuracy and stability of numerical algorithms. SIAM, 2002.
- [20] Stefan Hoehme, Marc Brulport, Alexander Bauer, Essam Bedawy, Wiebke Schormann, Matthias Hermes, Verena Puppe, Rolf Gebhardt, Sebastian Zellmer, Michael Schwarz, et al. Prediction and validation of cell alignment along microvessels as order principle to restore tissue architecture in liver regeneration. Proceedings of the National Academy of Sciences, 107(23):10371–10376, 2010.
- [21] Arun Jambulapati and Aaron Sidford. Ultrasparse ultrasparsifiers and faster laplacian system solvers. <u>ACM</u> Transactions on Algorithms, 2021.
- [22] Jonathan A Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A simple, combinatorial algorithm for solving sdd systems in nearly-linear time. In Proceedings of the forty-fifth annual ACM symposium on Theory of computing, pages 911–920, 2013.
- [23] Hildur Knutsdottir, John S Condeelis, and Eirikur Palsson. 3-d individual cell based computational modeling of tumor cell-macrophage paracrine signaling mediated by egf and csf-1 gradients. Integrative Biology, 8(1):104–119, 2016.
- [24] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In 2013 ieee 54th annual symposium on foundations of computer science, pages 147–156. IEEE, 2013.
- [25] P. Van Liedekerke, A. Buttenschön, and D. Drasdo. Chapter 14 off-lattice agent-based models for cell and tumor growth: Numerical methods, implementation, and applications. In Miguel Cerrolaza, Sandra J. Shefelbine, and Diego Garzón-Alvarado, editors, Numerical Methods and Advanced Simulation in Biomechanics and Biological Processes, pages 245 267. Academic Press, 2018.
- [26] Jörg Liesen and Zdenek Strakos. Krylov subspace methods: principles and analysis. Numerical Mathematics and Scie, 2013.
- [27] Paul Macklin, Mary E Edgerton, Alastair M Thompson, and Vittorio Cristini. Patient-calibrated agent-based modelling of ductal carcinoma in situ (dcis): from microscopic measurements to macroscopic predictions of clinical progression. Journal of theoretical biology, 301:122–140, 2012.
- [28] Bruce M Maggs, Gary L Miller, Ojas Parekh, R Ravi, and Shan Leung Maverick Woo. Finding effective support-tree preconditioners. In Proceedings of the seventeenth annual ACM symposium on Parallelism in algorithms and architectures, pages 176–185, 2005.
- [29] Rebecca McLennan, Linus J Schumacher, Jason A Morrison, Jessica M Teddy, Dennis A Ridenour, Andrew C Box, Craig L Semerad, Hua Li, William McDowell, David Kay, et al. Neural crest migration is driven by a few trailblazer cells with a unique molecular signature narrowly confined to the invasive front. Development, 142(11):2014–2025, 2015.
- [30] Gérard Meurant and Petr Tichy. Error Norm Estimation in the Conjugate Gradient Algorithm. SIAM, 2024.
- [31] Eirikur Palsson. A 3-d model used to explore how cell adhesion and stiffness affect cell sorting and movement in multicellular systems. Journal of Theoretical Biology, 254(1):1–13, 2008.
- [32] Alex Pothen and Sivan Toledo. Elimination structures in scientific computing. Handbook of Data Structures and Applications, pages 945–965, 2018.
- [33] Jennifer Scott and Miroslav Tuma. Algorithms for sparse linear systems. Springer Nature, 2023.

- [34] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain. Carnegie-Mellon University. Department of Computer Science Pittsburgh, 1994.
- [35] Daniel A Spielman and Shang-Hua Teng. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. SIAM Journal on Matrix Analysis and Applications, 35(3):835–885, 2014.
- [36] Daniel J Tracy, Samuel R Buss, and Bryan M Woods. Efficient large-scale sweep and prune methods with aabb insertion and removal. In 2009 IEEE Virtual Reality Conference, pages 191–198. IEEE, 2009.
- [37] PM Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners, unpublished manuscript uiuc 1990. In <u>IMA Workshop on Graph Theory and Sparse Matrix Computation</u>, 1991.
- [38] P Van Liedekerke, M M Palm, N Jagiella, and D Drasdo. Simulating tissue mechanics with agent-based models: concepts, perspectives and some novel results. Comp. Part. Mech., 2(4):401–444, 26 November 2015.
- [39] Paul Van Liedekerke, Johannes Neitsch, Tim Johann, Kévin Alessandri, Pierre Nassoy, and Dirk Drasdo. Quantitative cell-based model predicts mechanical stress response of growing tumor spheroids over various growth conditions and cell lines. PLoS computational biology, 15(3):e1006273, 2019.
- [40] Mihalis Yannakakis. Computing the minimum fill-in is np-complete. SIAM Journal on Algebraic and Discrete Methods, 2, 03 1981.