Scaffolding Minimizing Construction Algorithms for Masonry Structures

Sue Anne Davis

Research Mentor: Dr. Etienne Vouga

2019
Abstract

The stability of masonry structures has applications in verifying the safety and feasibility of buildings new and old. The construction of masonry buildings can be optimized for reduced scaffolding when we utilize the stability of intermediate structures. I show how stability algorithms from previous work can help identify these intermediate structures, a novel algorithm for doing so, some explanation of my implementation, and the results.
Contents

Abstract i

Table of Contents iii

1 Introduction 1
   1.1 Analyzing Stability ........................................ 1
   1.2 Theory ..................................................... 4

2 Reformulation and Novel Algorithm 7
   2.1 Shared Theory and Implementation .......................... 7
   2.2 Reformulation of Vouga et. al. .............................. 8
   2.3 Novel Algorithm ........................................... 12

3 Results 13
   3.1 Disc ....................................................... 13
   3.2 Annulus ................................................... 14
   3.3 Square .................................................... 14

4 Conclusion 15
   4.1 Limitations ................................................ 15
   4.2 Future Work ............................................... 15

Bibliography 16
Chapter 1

Introduction

The completion of Santa Mario del Fiore is said to be the beginning of the Renaissance. The great dome, which harkened back to the Classical period, had been a technological feat lost to the Middle Ages. So, a competition was held to see who could come up with a construction plan for the structure. Plans submitted often consisted of huge amounts of scaffolding, most of which required more timber than that which was in all of Tuscany\(^1\). One of the proposals was to create a mold of the dome out of dirt, and then to simply lay the brick on top of it, such that the labor would be inexpensive and the materials cheap as... dirt. The task of removing the mountain of dirt out of the Cathedral once it was completed was to be done by burying coins within the dirt during construction, so that the poor people would carry it out for the builders. Since this method was frowned upon even back then, inventing a new method to create the dome was required. Ribbing served both as the means used to carry bricks to different parts, and as the form-work to which the rest of the building was shaped around. Thus, no scaffolding was required.

The question that both this contest and my project sought to answer is how to find intermediate states that reduce the total amount of scaffolding needed. While we oftentimes have steel and other technologies that allow us to create whatever form-work we like now, the amount of temporary form-work required has helped put the amount of construction waste at 42% of landfills\(^2\). Thus, sustainability (and cost effectiveness) drives this area of work. Since then, work has been done in the area of construction design for scaffolding minimization: however, the algorithms used to find construction states previously were heavily tied to the validation algorithm of said state\(^3\). However, in my project, the algorithm chosen for testing for stability of construction states is totally independent of the algorithm chosen for deciding which blocks to remove from the current state, allowing for more flexibility in algorithm choice.

1.1 Analyzing Stability

Strength Analysis

Analysis of the stability of a masonry building typically comes in two different flavors: strength analysis and stability methods. Strength analysis relies on the plastic and elastic characteristics of the material, as well as the geometric structure made out of it. Strength analysis will then predict how the nodes will deform and fracture under loads. This allows for flexibility in material when testing for critical displacements in a structure.
Stability Methods

Geometric stability methods, on the other hand, assume incompressibility of the building and only takes into account the geometry of the structure. Geometric stability have been used to design compressive architecture since the 1500’s, when Robert Hooke stated,

As it hangs in a continuous flexible form, so it will stand contiguously rigid when inverted.

This helped define the usage of thrust lines, which can be defined as a geometric representation of compressive forces through a 2D structure. To help describe this, bricks can be thought of as either anchored, which means that any forces will be supported (think of it being in the ground); or free, which have to be supported through forces from other bricks. It follows that a thrust line in 2D should terminate at anchored bricks and travel within all free bricks, and that we need not worry about any force, like weight, acting on anchored bricks.

Thrust lines can be thought of as a graph dual of the blocks, such that each block has a corresponding vertex $v_i$ in the thrust line, as well as an $F_i$, an external load (parallel to gravity). Usually this $F_i$ is the dead-weight of the brick itself. Each edge can be thought of as having a weight $w_{i,j}$, which corresponds to the force density transmitted between free blocks. The sum of all contact forces on block $i$ should therefore equal the external force $F_i$:

$$
\begin{pmatrix}
0 \\
0 \\
F(i)
\end{pmatrix} = \sum_{j \sim i} w_{i,j} (v_i - v_j) \tag{1.1}
$$

where the $\sim$ designates all vertices $j$ that are adjacent to vertex $i$.

Methodologies utilizing this theory include Gaudi’s hanging nets, which created a thrust network, a 3D representation composed of thrust lines, where a 3D structure has been “sliced” into many 2D forms.

Contemporary stability methods, make a few simplifying assumptions from Heyman’s The Stone Skeleton:

1. There are infinite frictional forces, so sliding failures do not occur.
2. Masonry is capable of infinite compression.
1.1. ANALYZING STABILITY

Figure 1.2: The original figure (orange) projected into the xy-plane (green)

3. Masonry has no tensile strength.

With these assumptions, we no longer need the material properties of the structure; instead, the masonry design problem is transformed into a geometry problem. Thus, we only need the volumes and placement of the blocks, as well as the forces on said blocks. Note that assumption 2 constrains equation 1.1, as if we are only using compressive forces, then \( w_{i,j} \geq 0 \). Proposing the above assumptions, Heyman\(^6\) introduced The Safe Theorem: If a system of forces that result in an equilibrium can be found within the masonry envelope, then the structure is stable, which can be applied in either the 2D or 3D scenario. Oftentimes, many different systems of forces can be found, which leads to a phenomenon called static indeterminancy. Static indeterminancy implies that without extensive testing, the true forces through the masonry is not determinate. As an example, take a four legged chair: a number of different combinations of forces could be transmitted through each leg with the final result still being stable.

The work of this particular thesis utilizes contemporary stability methods to verify stability.

However, I’m not the first! Whiting\(^5\) in her thesis created an implementation that utilized an energy function of infeasibility from the tensile and compressive forces between interfaces of rigid blocks, thus loosening the constraint of absolutely no tensile force. Instead, the tensile force could be viewed as the amount of “glue” required to hold the blocks together.

Block, in his thesis, put forth the idea of Thrust Network Analysis (TNA), which allowed for the analysis of 3D convex masonry through duality theory. His method attributed the dead-weight of lumps of building to each node, and the edges of the graph represented the interfaces. His work will be explained in greater detail in the theory section.

This idea was further worked upon by Vouga et. al.\(^7\), where the iterative updating and “modifying” was simplified into two steps: fix nodes and solve for weights of primal edges, then fix weights of edges and move nodes, both through quadratic programming.
1.2 Theory

Thrust Network Analysis Analyzed

TNA grows off of the base of Thrust Line Analysis. It utilizes the idea that all of the forces $F$ on every block $i$ within the $xy$ plane, $z$ as the vertical direction, must sum to 0. To write this more akin to the above equations, all contact forces $F$ on $i$, or $F \sim i$, should equal 0 in the plane, or $0 = \sum_{F \sim i} F_x$, and $0 = \sum_{F \sim i} F_y$. Since we utilize blocks as vertices $v_i$ and forces on said vertices as the product of length $v_i - v_j$ and weight $w_{i,j}$ of the block, then we can rewrite as

$$0 = \sum_{j \sim i} w_{i,j} (v_i.x - v_j.x)$$
$$0 = \sum_{j \sim i} w_{i,j} (v_i.y - v_j.y)$$

(1.2)

If we project the surface $\Omega$ in question into the $xy$ plane, the resulting surface, $\Omega$, will still have these invariants hold. TNA utilizes this understanding by noting that rotating every edge left by 90° will again not impact the invariant. Thus, if the surface $\Omega$ is in equilibrium, then a reciprocal dual of $\Omega$, $\Omega^*$ can be found.

![Figure 1.3: The original figure (yellow) with a reciprocal dual drawn within it](image)

For a more geometric representation, if all of the force vectors add to 0, then so will the vectors, if glued. There are many different weights that can create a closed face, they will just produce slightly differently shaped ones.

Furthermore, if a reciprocal dual can be found, then it can be scaled to support the vertical weights on the blocks. From both 1.2 and 1.1, we can see that when a node is in vertical equilibrium, then

$$F = \sum_{j \sim i} w_{i,j} (v_i.z - v_j.z)$$

(1.3)

Block went about this by first utilizing linear optimization on the magnitude of the dual’s edges, constrained by the equilibrium equations and direction required for a reciprocal dual, and then two more linear optimizations on actually defining where the vertices should go. After this reciprocal dual was found, the magnitudes of the horizontal edges
would be used to solve for nodal displacement $z$ to satisfy 1.3. However, this equilibrium surface, $G$ might not lie within the volume of the structure. Block then explored modifying the dual to attempt to bring $G$ closer inside of the volume.

**TNA Contributions**

Vouga et. al. found a more flexible way to construct the duals by thinking of them in the continuous context: if every block was infinitesimally small and there were infinitely many blocks. In the continuous setting, the 3D equilibrium can be characterized as a Airy Stress Function, which, when discretized as is necessary for the masonry case, follows that a reciprocal dual can always be produced from a flood-fill algorithm, rather than needing to rely on linear optimization. Vouga et. al. also put forth the algorithm where weights on the primal edges were calculated through quadratic programming, which more directly follows the invariants described above. The primal is then directly updated, again through quadratic programming, to new $z$ coordinates if satisfactory weights could not be found. My work builds off of this last two-step algorithm by finding weights, and if the weights have little enough error respective to the forces they need to support, then I return that the current structure is stable.
Chapter 2
Reformulation and Novel Algorithm

2.1 Shared Theory and Implementation

For this thesis, there were two implementations done: one was a reformulation of Vouga et. al. and the other is a novel algorithm put forth in this thesis, which utilizes a portion of this reformulation of Vouga et. al. Since there are shared portions of implementations, both have to solve some of the same issues, namely indexing, which is discussed here in 2.1. Implementation of the algorithms above can be accessed on GitHub\textsuperscript{8}.

Relation To Theory

First, my project includes code that identifies free and anchored nodes, and then utilizes an indexing data structure to correctly construct optimization matrices with only free nodes.

Second, my project rests on the understanding that if a set of weights can be found such that the forces $F_i$ on each vertex are in equilibrium with the three equilibrium equations above (1.1 and 1.2), then the structure is stable. If we can remove some subset of nodes and then calculate a set of weights that puts us in equilibrium, then we have found a stable substructure.

Free and Anchored Blocks

Extracting Anchored Blocks

As described in 1.1, anchored blocks are immobile, and can withstand infinite forces, while free blocks require 1.2 and 1.3 to be true to be stable. All free blocks \textit{must} be stable for the entire structure to be stable. In my implementation, I assumed that all exterior vertices are anchored, while all interior vertices are free. When given a mesh in .obj format, there is no distinction between interior and exterior vertices, so a half-edge data structure was used. Half-edge data structures create two directed “half” edges for each edge in the mesh, which then allows for graph traversal. Therefore, I used a half-edge data structure to crawl through the graph and simply mark which vertices belong to an edge that is on the border. This crawling was done with libigl's\textsuperscript{9} \texttt{HalfEdgeIterator}.

Once all of the border vertices are in an “anchored” node set, then any vertex not in said set are free vertices, and are placed into a “free” node set, and the anchored set is
no longer needed. This set can be modified to suit the needs of the actual architecture, i.e. an arch on the ground floor or a supporting column within the structure, by simply adding or removing the corresponding vertices from the free node set.

**Indexing**

As previously described in 1.1, free nodes are the only nodes that we must analyze forces for, and the only nodes that I am allowing to be moved. We can think of the set of free nodes \( U \) as being ordered, so that we can index into them. Let us call the number of free vertices \( ||v_\omega|| \) This will become useful when constructing matrices.

As will be discussed later in 2.2, the edges that connect to free nodes, and therefore require weights, must also be indexable. Let us call the number of all the edges that connect to free nodes \( ||e_0|| \). I enumerated all edges in the graph by going through each face and placing the two vertices that define each edge into an unordered set. I then go through all the edges, and assign a unique integer between 0 and \( ||e_0|| - 1 \) to each unordered pair of vertices that contained a free vertex, which I shall now refer to as a “free edge”. This gives us one-to-one mapping, \( R_E \), between the numbers 0 through \( ||e_0|| - 1 \) and all free edges, which we can use to index into them.

These two indexing functions, \( U \) and \( R_E \), are going to be necessary to describe how I index into my matrices, and are used in the following discussions.

### 2.2 Reformulation of Vouga et. al.

**Overview of Quadratic Programming**

Quadratic problems finding optimal solution \( x \) given matrices \( G \) and vector \( a \), such that

\[
f(x) = \frac{1}{2} a^T x + \frac{1}{2} x^T G x
\]

(2.1)

is minimized. \( x \) is then optionally constrained with \( E^T x = e \) and \( C^T x \geq c \).

**Optimizing For Weights**

The energy function I am using is an interpretation of Vouga et. al.

First, we calculate a force on each vertex \( i \), \( F(i) \), that is the product of both \( i \)'s force density and \( i \)'s Voronoi area. Once this is computed, the energy function becomes minimizing the difference between vertical force \( F(i) \) with the support we attribute to the current geometry, which is summarized in the formula 1.3. For a single vertex \( i \), the function would be

\[
||F(i) - \sum_{j \sim i} (v_i.z - v_j.z)w_{i,j}||^2
\]

where \( j \) is all vertices adjacent to \( i \) and the weight \( w_{i,j} \) corresponding to each edge \( \{i, j\} \) adjacent to \( i \).

To create a system of all the equilibrium equations, we first pack all of the edge weights into a vector \( w \) of size \( ||e_0|| \) where \( w_{i,j} = w(R_E(b)) \) is the weight of interior edge \( \{i, j\} \) and \( b \) is some unique integer between 0 and \( ||e_0|| \), as described in 2.1. The matrix \( D \) to represent \( v_i.z - v_j.z \), or the \( z \) difference between all adjacent nodes, would be of size
2.2. **REFORMULATION OF VOUGA ET. AL.**

$||v_ω||$ by $||e_0||$. $D$ will represent an altered adjacency matrix, and is laid out such that for any row $i$ has an affiliated internal node $k$, $k = U(i)$, as previously discussed in 2.1. Every column has an affiliated free edge as well, described by the indexing function $R_E$. Thus, in row $i$, with associated free vertex $k$, any column $b$ associated with free edge $R_E(b) = \{k, j\}$, where $k$ is an endpoint, will have an entry at

$$D(i, b) = v_k.z - v_j.z$$

All other entries, where the column’s affiliated edge does not contain the row’s affiliated vertex as an endpoint, will be 0. Note that any column will have two entries in the entire matrix, one for each endpoint, and they will be of equal and opposite magnitude. Thus, when multiplying $ Dw $, we should get as a result vector, for each row

$$\sum_{j\sim i} (v_i - v_j)w_{i,j}$$

The force function $ F(i) $, can be written as vector $ F $, which is of the size $ ||v_ω|| $, by indexing with $ U $. The energy function is to minimize is $ ||F - Dw||^2 $$, so a little structuring had to be done, much to the tune of a constrained linear least squares problem. From the energy formula, we can rewrite

$$||F - Dw||^2 = (F - Dw)^T (F - Dw)$$

$$= (F^T - w^T D^T) (F - Dw)$$

$$= F^T F - F^T Dw - w^T D^T F + w^T D^T Dw$$

Now, a slight rewriting of a component

$$w^T D^T F = (F^T Dw)^T$$

Note: The transpose is equal, in this case, as the product is a 1 by 1 matrix. Thus, we can achieve:

$$F^T F - 2F^T Dw + w^T D^T Dw$$

Since we are finding a minimum, the constant $ F^T F $ is not necessary for the calculation, so the linear component will be $ 2F^T D $ while the quadratic component is $ D^T D $.

We do still however, need to make sure our results of the minimization represent valid weights and states. As mentioned in the theory section, all forces in the $ xy $ plane are represented and constrained by these two equations:

$$\sum_{j\sim i} w_{i,j} (v_i.x - v_j.x) = 0$$

$$\sum_{j\sim i} w_{i,j} (v_i.y - v_j.y) = 0$$

(1.2)

We can write this, to conform with the requirements of the quadratic programming setup, as two matrices: $ X $ and $ Y $.
CHAPTER 2. REFORMULATION AND NOVEL ALGORITHM

$X$ is a free node by free edge matrix, which will attempt to satisfy the equation in the $x$ dimension, and it is much like $D$, just in the $x$ dimension instead of $z$. Each row $i$ corresponds to a free node $j$, where $U(i) = j$. Each column $l$ corresponds to some free edge $R_E(l) = \{m, n\}$. We create an altered adjacency matrix, so that in row $i$ with associated $j$, all columns $t$ where $R_E(t) = \{r, j\}$ with $j$ is an endpoint, will contain the value $v_j.x - v_r.x$, and all other columns will contain zero, much like the $D$ matrix.

For the $Y$ matrix, the same structuring is done as in both $D$ and $X$, but this time in the dimension $Y$. We can then combine these two matrices by interlacing the $x$ difference rows and $y$ difference rows between each other, like so:

$$L = \begin{bmatrix} X.row(0) \\ Y.row(0) \\ X.row(1) \\ Y.row(1) \\ \vdots \end{bmatrix}$$

To configure these matrices into the quadratic form 2.1, I used $G = 2D^T D$, $x = w$, $a = 2F^T Dw$, and with $E = [L]$, $e = \{0\}$, $C =$ Identity, $c = \{0\}$.

If $||F - Dw||^2 = 0$, then the result of the minimized quadratic program should be $-F^T F$, which can be utilized to help find when to halt the optimization.

Optimizing for Vertex Positions

For updating the vertices, we must still minimize $F_i - \sum_{j\sim i} w_{i,j} (v_i - v_j)$. To do this, I flattened all free vertices into a vector $v$, which is of size $||v_\omega||$ by dimension (3, in my implementation). However, since we are now using $v$ as a means to optimize rather than $w$, we can no longer use the previous quadratic formulation.

Since we are not going to update the placements of the anchored nodes, the forces given by those need to be otherwise accounted for. Instead of using $F(i)$ as my left side of the equation, I used $B(i) = F(i) - \sum_{j\sim i\in\text{anchored}} w_{j,i} v_j.z$, so that the stability constraint would then be

$$B(i) = \sum_{j\sim i\in\text{free}} w_{i,j} (v_i.z - v_j.z) + \sum_{j\sim i\in\text{anchored}} w_{i,j} v_i.z. \quad (2.3)$$

This is equivalent to subtracting both sides of 1.3 by $\sum_{j\sim i\in\text{anchored}} w_{j,i} v_j.z$.

The new minimization is now $||B - Mv||$, where again $v$ is free. To create $M$, we now construct $||v_\omega||$ by $3||v_\omega||$ sized matrix. For every row $i$, there is an affiliated internal node $k$, $U(i) = k$. The columns, which have 3 columns for every free vertex, can be thought of as having the three different dimensions for each free node, so $U(i/3) = k$, and $i.x, i.y, i.z$ are the three columns associated with $k.x, k.y$, and $k.z$. For every row $i$ (and internal node $v_k$), we insert entry

$$M(i, 3i + 2) = \sum_{j\sim k} w_{k,j}$$

Since indexing function $U$ is one-to-one from indices to free nodes, it has an inverse, $U^{-1}$ allowing us to find a corresponding index when given a free node. We can therefore add
for every node \( \{ j \sim k \mid j \in \text{free} \} \) at indices \((i, 3(U^{-1}(j)) + 2)\) a value of \(-w_{k,j}\), so that the following entries are made

\[
\forall j \sim i | j \in \text{free} \quad M(i, 3(U^{-1}(j)) + 2) = -w_{k,j}
\]

Every other entry is 0 in \( M \). Thus, the product of \( Mv \) is a vector to which every row equals

\[
\sum_{j \sim i | j \in \text{free}} w_{i,j}v_i.z + (-w_{i,j})v_j.z + \sum_{j \sim i | j \in \text{anchored}} w_{i,j}v_i.z
\]

Which can be simplified to

\[
\sum_{j \sim i | j \in \text{free}} w_{i,j}(v_i.z - v_j.z) + \sum_{j \sim i | j \in \text{anchored}} w_{i,j}v_i.z
\]

While we now have some of the matrices necessary, we still need to formulate our constraint matrices. For our 1.2 constraints, we again need to change the equation to encompass the idea that the anchor nodes are now constants.

\[
\sum_{j \sim i | j \in \text{free}} w_{i,j}(v_i.x - v_j.x) = 0
\]

\[
\sum_{j \sim i | j \in \text{free}} w_{i,j}(v_i.x - v_j.x) + \sum_{j \sim i | j \in \text{anchored}} w_{i,j}v_i.x = -\sum_{j \sim i | j \in \text{anchored}} w_{i,j}v_j.x
\]

To represent this right hand size, we create a vector \( x_v \) where

\[
x_v(i) = -\sum_{j \sim U(i) | j \in \text{anchored}} w_{U(i),j}v_j.x
\]

To represent the left, we create a matrix \( X_v \) which is very similar in construction to \( M \). Every row \( i \) still has an associated vertex \( k, U(i) = k \). Since I chose to view the columns as dimensions, we can insert into columns associated with \( x \) instead of the columns associated with \( z \), and the values of \( v.x \) instead of the values of \( v.z \) with the exact same process as \( M \). The product of \( X_v \), will give us for each row

\[
\sum_{j \sim i | j \in \text{free}} w_{i,j}(v_i.x - v_j.x) + \sum_{j \sim i | j \in \text{anchored}} w_{i,j}v_i.x
\]

We can repeat this process with a matrix \( Y_v \) and vector \( y_v \), with respect to \( y \) instead of \( x \). We can then interlace the two, as we did in 2.2, creating a \( 2||v_\omega|| \times 3||v_\omega|| \) matrix \( P_v \), and the accompanying vector \( p_v \) which has rows interlaced in the same way.

We can parameterize a compound energy function with scalar weights \( \alpha_0 \) and \( \alpha_1 \), such that the energy function becomes

\[
\alpha_0 ||B - Mv||^2 + \alpha_1 ||v_0 - v||^2
\]

where \( v_0 \) is the flattened locations at the beginning of the iteration. This allows for the positions of the vertices to move to satisfy the stability constraints while also being penalized for moving too far. If penalizing poor stability is more important, which might be true for earlier iterations, then \( a_0 >> a_1 \). I used \( a_0 = 0.9 \) and \( a_1 = 0.1 \), for instance.
Otherwise, if you would rather slow down the movement of nodes, and search for stable weights more finely, the weights could be more equal, with $a_0 = 0.7$ and $a_1 = 0.3$. I made them add up to one just because it made the arithmetic simpler in my code, but this is unnecessary.

The means of decomposing $\|B - Mv\|$ and $\|v_0 - v\|^2$ into a form compatible with quadratic programming is the exact same process and method as described for the weight optimization step. Thus, the decomposition for $\|B - Mv\|$ should be

$$B^T B - 2B^T M v + v^T M^T M v$$

while the formulation for $\|v_0 - v\|^2$ would instead be

$$v_0^T v_0 - 2v_0^T v + v^T v$$

To combine these two and put them into the quadratic programming layout 2.1, $x = v$, $G = \alpha_0 M^T M + \alpha_1 \{ I \}$, $a = 2(\alpha_0 B^T M + \alpha_1 v_0^T)$, and finally $E = P_v$, and $e = p_v$.

This formulation does not have an inequality constraint. The ”perfect” outcome for this particular minimization would be $-\alpha_0 F^T F - \alpha_1 v_0^T v_0$.

### 2.3 Novel Algorithm

<table>
<thead>
<tr>
<th>Algorithm 1 Intermediate Structure Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>function</strong> DELETEANODE</td>
</tr>
<tr>
<td>cheapestNode ← findLeastForces()</td>
</tr>
<tr>
<td>removeNode(cheapestNode);</td>
</tr>
<tr>
<td><strong>end function</strong></td>
</tr>
<tr>
<td><strong>procedure</strong> MAIN</td>
</tr>
<tr>
<td>do</td>
</tr>
<tr>
<td>deleteANode()</td>
</tr>
<tr>
<td>successValue ← computeWeights()</td>
</tr>
<tr>
<td>while successValue &lt; $\delta$ and more faces exist</td>
</tr>
<tr>
<td><strong>end procedure</strong></td>
</tr>
</tbody>
</table>

This is the novel algorithm put forth in this thesis: find the next intermediate state for scaffolding. Given a structure (not necessarily a stable one), this algorithm will find the cheapestNode, which in my case is the one with the highest $z$ coordinate. Then, we delete that vertex from the graph and test for stability in the remaining structure with the exact same implementation as 2.2. If the success value is within some small $\delta$ of a perfect solution, then we have a stable structure. The perfect solution value can be calculated trivially, as described in 2.2 as being $F^T F$. We can then rinse and repeat, finding the next cheapest node, then testing for stability, until we reach an unstable structure. The next unstable structure represents a state where there is a need for form-work.
Chapter 3

Results

These results come from creating what is assumed to be stable structures. The meshes’ $z$ coordinates are created from constraining the Graph Laplacian, $L$, as

$$Lz = 1$$

(3.1)

such that all edge vertices have $z = 1$ and more interior nodes have a greater $z$ value. The force density on each vertex is assumed to be constant for all vertices (as would be if all blocks were identical in physical weight), and the area designated to each vertex is the Voronoi area of the vertex on the surface.

3.1 Disc

![Figure 3.1: A dome and the next unstable state after applying our algorithm.](image)

When creating a dome, each layer is stable, as it forms a ring where force distribution in the $xy$ plane can be found. This is empirically backed, and one of the reason why domes are so popular as a building structure. When I created a Laplacian distributed disc, I expected that it would remove quite a few nodes before hitting an unstable configuration. The lack of stability might be due to the slight bumpiness of the shape, which causes a deformation in the ring and the surrounding support for the ring, which causes the 1.2 constraints to be violated.
Figure 3.2: The original annulus (left) and the next unstable state (right)

3.2 Annulus

Because an annulus has a much higher ratio of anchored nodes to free nodes thanks to the hole in the center, more nodes were able to be supported. This is due to significantly fewer free nodes relying solely on other free nodes to support their own weight.

3.3 Square

Figure 3.3: A square mesh and the next unstable state

The square shape with Laplacian distribution seemed to actually replicate the behavior that I was expecting with the dome. Again, the same issue seems to arise, which I attempted to capture in the picture: the ring that should be able to support the oculus is too deformed.
Chapter 4

Conclusion

4.1 Limitations

One of the limitations of the current implementation is that it assumes constant force density of brick-representing nodes. This could be fixed with a richer format for structure meshes, such that each node is four dimensional, with its fourth dimension being the force density.

Another limitation is the choice to use an active-set algorithm for sparse matrices that came out of the box with LIBIGL\textsuperscript{9}. The first choice, Quadprogpp\textsuperscript{10}, would always converge on a viable solution or return an error. Quadprog++ was able to handle a number of vertices, or blocks, on the scale of thousands, which would take about ten minutes to compute on my laptop. I thought that utilizing sparse matrices (as my implementation section 2 explains that I used) would speed things up, and I was also worried about bugs within Quadprogpp’s data structures, as I had to fix one of them to make my code work. The active set algorithm would quietly break instead of giving an error code, which was difficult to debug and or work with, as I can move unstable nodes in the $xy$-plane but I can’t make negative weights valid. It also fails to converge on any valid state for larger meshes, even when the shape given should imply a number of valid states. For instance, a mesh with 118 triangles will converge correctly in 7 seconds, a mesh with 302 triangles will converge correctly in 1 minute and 4 seconds, and a mesh with 1102 triangles will crash within the quadratic optimization at 1 minute and 34 seconds. Another issue with this project is the lack of variety in meshes. Because of the dearth of easily triangulated buildings, I utilized a Triangle\textsuperscript{11} wrapper implemented in LIBILG. This leads to less building shaped shapes than I would have hoped, but can be otherwise expanded.

Finally, the Vouga et al. formulation of updating vertices relies on $F(i)$ being used. However, $F(i)$ is a function of the force density on node $i$ and the Voronoi area of node $i$, which is expected to change during vertex updates. Therefore, the shift of the vertices to a locally stable state (stale $F(i)$) is not necessarily more stable globally in my formulation, and might cause a lack of convergence.

4.2 Future Work

For the future of this problem, it could be beneficial to modify the problem such that allowing for modification of the finished building to optimize for further lack of scaffolding.
The algorithm for deciding nodes to delete could also be bolstered by known architectural heuristics for stable standing structures, to allow for more variance in intermediate structures rather than what the greedy solutions would provide.

Finally, the algorithm presented is agnostic to whether it is called on a stable structure or not. Therefore, it can be used to create entire sequences of unstable partial states (the ones that will require form-work).
Bibliography


[2] AN INVESTIGATION OF CONSTRUCTION WASTE MADE IN MAINLAND CHINA.


[11]