An Algorithm for Balancing Sofas

Jordan Fields

December 28th, 2019

1 Objective

Posed by Leo Moser in 1966, the moving sofa problem remains unsolved. In 1992, Joseph Gerver constructed a sofa which he conjectured to be the optimal solution to Moser's problem. However, Gerver provided no proof of this conjecture and to this date it remains unproven. Making use of Gerver's observations, we develop an algorithm which provides computational evidence for Gerver's conjecture.

This document aims to highlight the mathematics which inspired *sofa_gui.py* and provide a brief description of the the inner-workings of the program.

2 A Brief History

In 1966, Leo Moser asked the question

> "what is the shape of largest area in the plane that can be moved around a right-angled corner in a two-dimensional hallway of width 1?"

Despite the problem's simple statement, it remains unsolved. Notable points of progess in the problem's history begin in 1968 with the work of John Hammersley who conjectured that his construction (*Figure 1*) of area $2/\pi + \pi/2 = 2.2074...$ was optimal.

Figure 1: Hammersley's Sofa

In 1992 came the work of Joseph Gerver who found a valid construction (*Figure 2*) of area 2.2195... which was a modest improvement over Hammersley's construction.

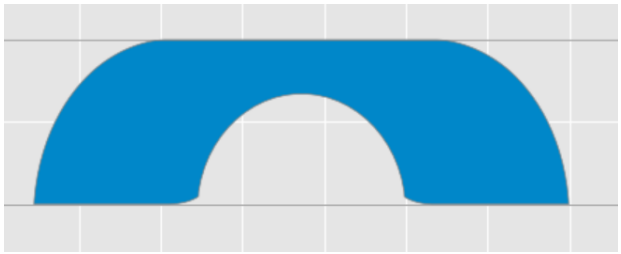

Figure 2: Gerver's Sofa

However, Gerver's results were interesting for reasons other than an increase in area over Hammersley's sofa. Derived from considerations of local optimality, Gerver conjectured that his construction was indeed the solution to Moser's 1966 question. Despite the fact that Gerver's construction remains the largest discovered to date, his conjecture of global optimality remains unproven.

3 Discrete Approximations

3.1 Notation

We begin by defining the sofa in terms of a rotating coordinate frame. Consider the L-shaped hallway that a candidate sofa moves through as given by

$$\begin{aligned} L_{\text{horiz}} &= \{(x,y) \in \mathbb{R}^2 : x \leq 1, 0 \leq y \leq 1\} \\ L_{\text{vert}} &= \{(x,y) \in \mathbb{R}^2 : y \leq 1, 0 \leq x \leq 1\} \\ L &= L_{\text{horiz}} \cup L_{\text{vert}} \end{aligned}$$

Let R_t denote the rotation matrix

$$R_t = \begin{pmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{pmatrix}$$

Then we let S be the planar shape which satisfies

$$S = L_{\text{horiz}} \cap \bigcap_{0 \leq t \leq \pi/2} (x(t) + R_t(L)) \cap \big(x(\pi/2) + R_{\pi/2}(L_{\text{vert}})\big)$$

Where $x(t) : [0, \pi/2] \to \mathbb{R}^2$ is a continuous path satisfying $x(0) = (0,0)$ which defines the movement of the inner corner of L. As in [Romik, 2016], we call $x(t)$ the *rotation path*. Only $x(t)$ which produce connected shapes will be considered. All S defined in this way will be able to traverse the L-shaped hallway. For simplicity's sake, we can enforce the same restrictions on S by combining L_{horiz} with L_{vert} to create one long horizontal hallway. Let

$$L_{\text{strip}} = \{(x,y) \in \mathbb{R}^2 : 0 \leq y \leq 1\}$$

Now define

$$S = L_{\text{strip}} \cap \bigcap_{0 \leq t \leq \pi/2} (x(t) + R_t(L))$$

Consider then, for any given $x(t)$, a sampling of $N+1$ equidistant nodes. More specifically, call $\{X_i\}_{0 \leq i \leq N}$ the set of *anchor points* which satisfy $X_i = x(\pi/2 \cdot i/N)$. For the sake of simplifying notation, let $R_{\pi/2 \cdot i/N} = R_{i^*}$.

We call S_N the discrete approximation of S given by

$$S_N = L_{\text{strip}} \cap \bigcap_{i=1,2,\ldots,N-1} (X_i + R_{i^*}(L))$$

3.2 Discrete Approximations

One would hope that $\lim_{N\to\infty} S_N$ would tend towards S in area and shape for a fixed rotation path $x(t)$. In absence of a proof, we will show computationally that this appears to be the case for Hammersley's sofa. Hammersley's sofa was chosen for computational simplicity, but the argument should extend to other sofa shapes.

To illustrate this principle, consider the rotation path $x(t) = 2/\pi(\cos t - 1, \sin t)$ which gives Hammersley's sofa. For $N = 2, 3, ..., 256$ we generated the corresponding set of anchor points and then built S_N. This was done using the function hallway_set() which builds a list of elements where the i^{th} element is given by $X_i + R_{i^*}(L)$. Then, the set was passed to the function *set_to_poly()* which intersected every element in the set and returned the resulting polygon. Both of these functions are found in *sofa_gui.py*.

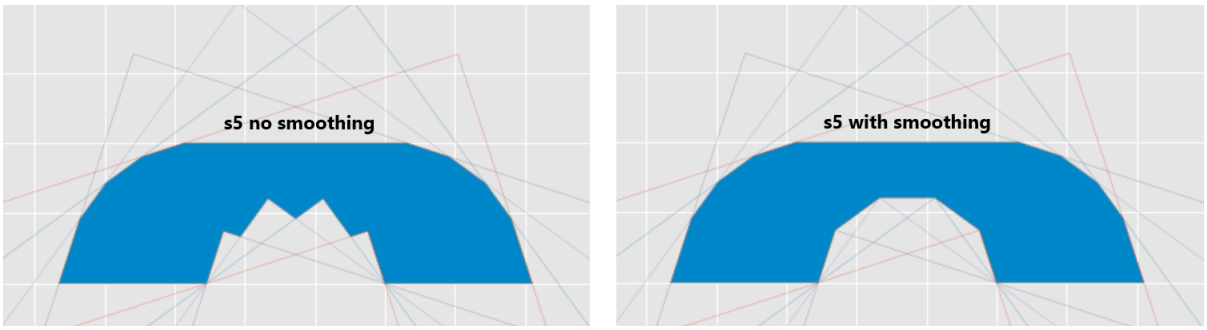

Figure 3: Discrete Approx of Hammersley's Sofa for N=5

Then, we calculated the difference between the area of S_N and Hammerley's sofa then plotted the results. In addition, we created 'smoothed' versions of S_N, denoted by S_N^*, by doing piecewise linear interpolation between all anchor points to redefine the boundary. The following figures demonstrate the convergence of S_N and S_N^* to Hammersley's sofa. If we let $\lambda(H)$ be the area of Hammersley's sofa, then the figure suggests the following

$$|\lambda(S_N) - \lambda(H)| = \mathcal{O}(1/N) \quad \text{and} \quad |\lambda(S_N^*) - \lambda(H)| = \mathcal{O}(1/N^2)$$

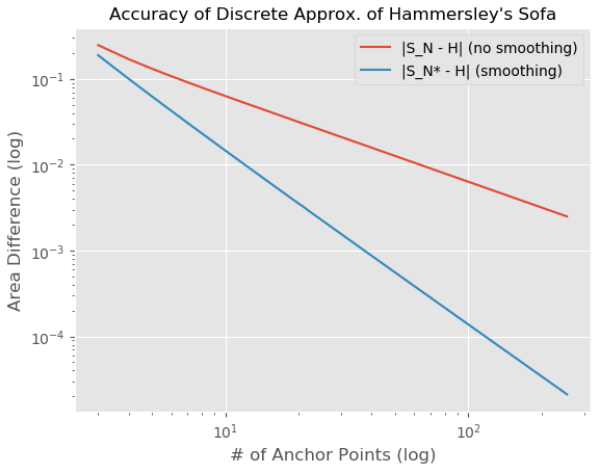

Figure 4: Convergence of S_N to Hammersley's Sofa with and without smoothing

3.3 Balancing Polygons

Gerver began his 1992 paper by giving the following definition

> Definition: A polygon is *balanced* if, given any side, that side and every other side parallel to it lies on one of two lines, where the distance between the lines is 1 and the total length of the sides lying on each of the two lines is equal.

The fundamental function of the algorithm is to use Gerver's balanced condition to continually grow the size of an unbalanced discrete sofa. We will now describe mathematically how the discrete sofa is balanced.

Theorem 3.1. *Let*

$$S_N = L_{horiz} \cap \bigcap_{i=1,2,\dots,N-1} (X_i + R_{i^*}(L)) \cap (X_N + R_{N^*}(L_{vert}))$$

such that S_N is not a balanced polygon. Then, there exists a linear function $\delta : (0,\infty) \to \mathbb{R}^2$ and an $\epsilon > 0$ such that we can define

$$S'_N = S_N / \{X_k + R_{k^*}(L)\} + \{X_k + \delta(\epsilon) + R_{k^*}(L)\}$$

for some $k \in \{0, ..., N\}$ such that

$$\lambda(S'_N) > \lambda(S_N)$$

Proof. Let l_1, l_2 be two parallel lines unbounded in length and one unit apart. Let $\delta(t) : (0,\infty) \to \mathbb{R}^2$ be a linear function that is perpendicular to both l_1 and l_2 such that for $\epsilon > 0$

$$||l_1 - \delta(t-\epsilon)|| > ||l_1 - \delta(t+\epsilon)||$$

Let S_N be a discrete sofa. Let $(l_1 \cap S_N) \subset \partial S_N$ and $(l_2 \cap S_N) \subset \partial S_N$. Let $(X_k + R_{k^*}(L))$ be called the k^{th} *hallway*. Note that both $(l_1 \cap S_N)$ and $(l_2 \cap S_N)$ are connected and, by the construction of S_N, it holds that $(l_1 \cap S_N), (l_2 \cap S_N) \subset h_k$. for some $k \in \{0, ..., N\}$.

Without loss of generality assume that $||l_1 \cap S_N|| > ||l_2 \cap S_N||$. By Theorem 1 in [Gerver,1992], we can increase the area of S_N by a perpendicular shift of the k^{th} hallway towards $(l_1 \cap S_N)$. For some $\epsilon > 0$, set

$$S'_N = S_N / \{h_k\} + \{X_k + \delta(\epsilon) + R_{k^*}(L)\}$$

Then it follows that

$$\lambda(S'_N) > \lambda(S_N)$$

□

In the next section a suitable choice for δ becomes clear.

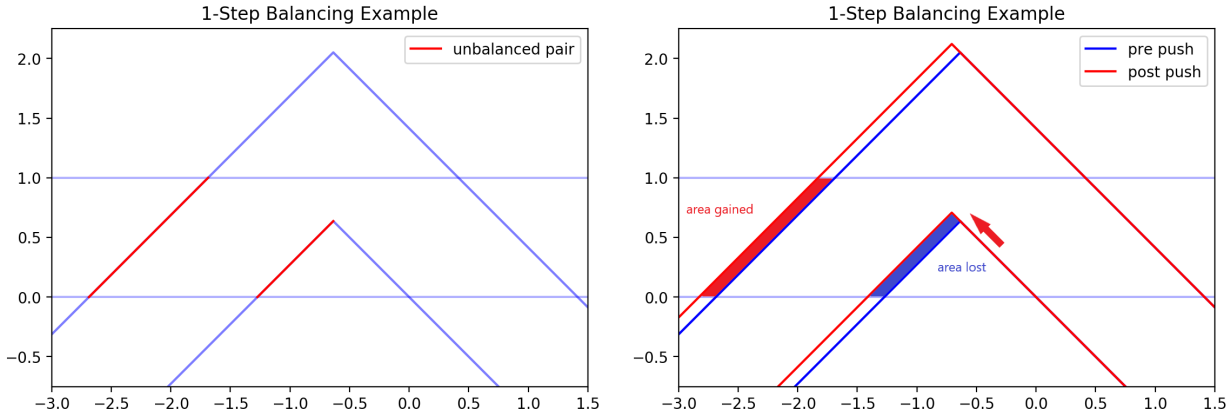

Figure 5: An example illustrating how the balanced condition can be used to increase polygon area

As illustrated in Figure 5, when we translate one of the hallways by a small amount in the direction towards and perpendicular to the largest boundary line in the 'unbalanced pair' the area we gain is larger than the area lost. The algorithm works by continually finding unbalanced pairs and then performing a balancing step to increase the area of the polygon. As the algorithm continues, the polygon will become "more balanced" in that the length difference between any two line pairs will begin to be very small.

3.4 Measuring Angles and Finding Delta

Using the function *get_features()*, details omitted for the sake of brevity, the program chooses an initial line, l_1, on the boundary of S_N. The line is defined by a set of two points $(x_1, y_1), (x_2, y_2)$. The program then measures the angle of the line in the following manner:

$$\theta = \begin{cases} \text{atan2}(y_2 - y_1, x_2 - x_1) & y_2 > y_1 \\ \text{atan2}(y_1 - y_2, x_1, x_2) & y_1 > y_2 \\ \pi/2 & y_1 = y_2 \\ 0 & x_1 = x_2 \end{cases}$$

The benefit of measuring θ in such a way is that it allows for easy identification of the corresponding h_k and makes a suitable choice for δ clear.

First, we recover the index for the k^{th} hallway by

$$k = \begin{cases} N\theta 2/\pi & \theta > 0 \\ N\theta 2/\pi + N & \theta < 0 \end{cases}$$

Before we define δ, we need to clarify how lines are chosen. The program chooses l_1 to be the line with the most negative x-coordinate. If the x-coordinates are equal, then the line with the largest y-coordinate is chosen as l_1. Then, we define the sgn variable as such

$$sgn = \begin{cases} -1 & l_1 > l_2 \\ 1 & l_1 < l_2 \end{cases}$$

Now we are ready to define δ.

$$\delta(\epsilon) = (sgn \cdot \epsilon \cdot \cos(\theta), sgn \cdot \epsilon \cdot \sin(\theta))$$

4 The Algorithm

4.1 Shapely, Discrete Sofas, and Hallway Sets

The Shapely library provides much of functionality for handling polygons and, performing intersections and transformations throughout the program. Important for understanding the algorithm is understanding the relationship between the discrete sofa S_N and the *hallway_set* variables. In short, the program makes its measurements on the sofa (such as line length, line angles etc) and then manipulates the hallway_set according to the previous measurements. Then, the program generates a new S_N by intersecting the elements of the hallway_set.

4.2 Pseudocode

Let S_N be a discrete sofa. Let *hallway_set* be the set of $(X_i + R_{i^*}(L))$ for $i = 0, 1, ..., N-1$ where the 0^{th} element is L_{strip}. The following pseudocode describes a single iteration of the main function which performs a *balancing operation* which increases the size of S_N. The pseudocode assumes that ϵ was chosen correctly. In the actual implementation of the algorithm, if ϵ is chosen to be too large the program throws away the results and tries a slightly smaller ϵ.

Algorithm 1 sofa_gui.py skeleton - a single iteration

Assume *hallway_set* provided as a set of each $(X_i + R_i(L))$ for $i = 1, ..., N$
Begin
Set S_N = the intersection of all elements in *hallway_set*
Take Measurements
→ Initialize list $\{P_n\}$ of points (x_n, y_n) that make up the corners of S_N
→ Initialize list $\{L_n\}$ of tuples $((x_{n-1}, y_{n-1}), (x_n, y_n))$ which define the lines making the edges of S_N
→ Initialize list $\{\theta_n\}$ of the angles of each element of $\{L_n\}$ as described in *3.4*
Find Where S_N is Unbalanced
→ Pick any $L_r \in \{L_n\}$ and the corresponding $\theta_r \in \{\theta_n\}$
→ Cycle through $\{L_n\}$ until we find $L_{r'}$ such that $\theta_r == \theta_{r'}$
Balance S_N
→ Initialize k and calculate value as described in *3.4*
→ Initialize sgn and calculate value as described in *3.4*
Update *hallway_set*
→ *hallway_set*[k] = *hallway_set*[k] $+\delta(\epsilon)$
GOTO Begin

5 Results

The following two figures show the results of short runs (1000 iterations) of the algorithm for low values of N as implemented in *sofa_gui.py*.

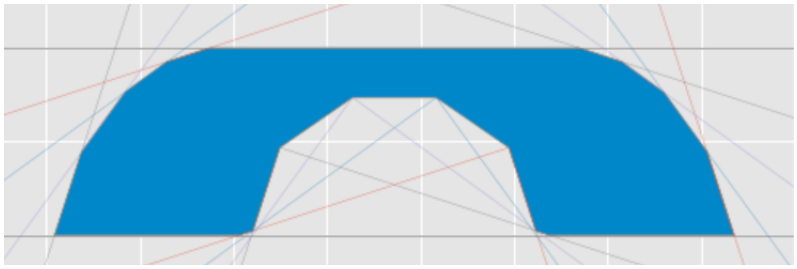

Figure 6: N=5 with smoothing

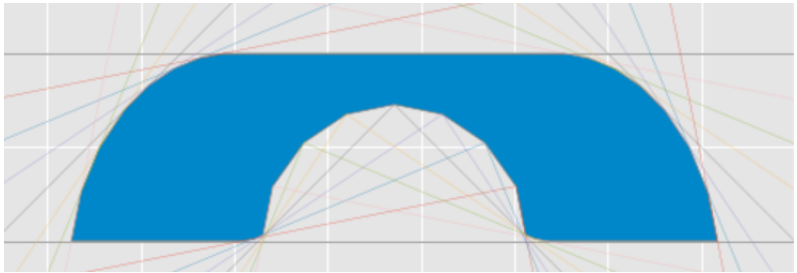

Figure 7: N=8 with smoothing

The next two figures show the results of slightly longer runs (10-15k iterations) on larger values of N. The shapes quickly become indistinguishable from Gerver's sofa and appear to converge in area.

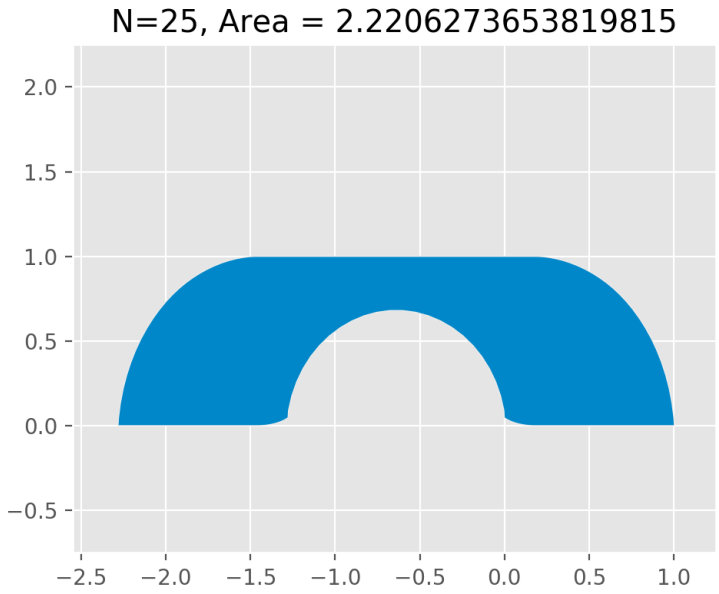

Figure 8: N=25 with smoothing

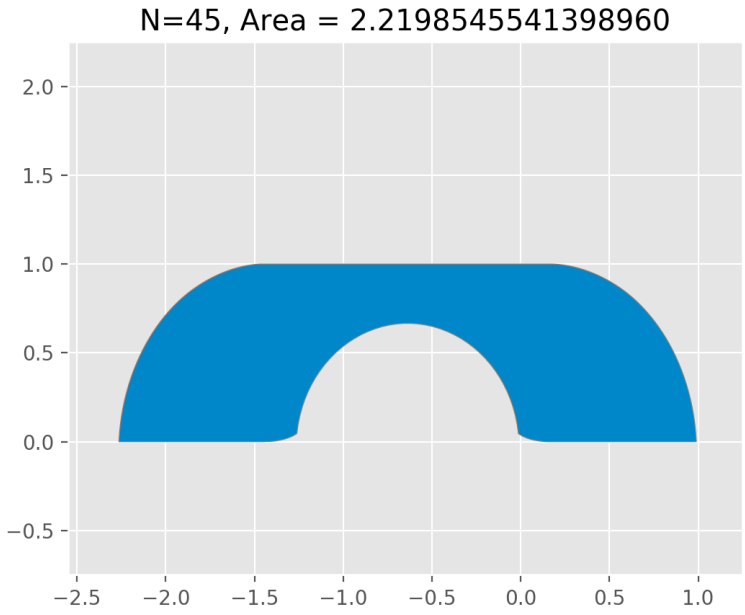

Figure 9: N=45 with smoothing

6 Installation and Dependencies

The software was written in Python 3.6 and should work with any version of Python 3.x. The software depends on a number of packages to run. The packages are *math* (native), *numpy, shapely, descartes, matplotlib, PySimpleGUI, random* and *copy*. The program should be run from the command line.

7 Using the Program

The GUI interface provides the user with a number of ways to alter the behavior of the program. Each option and its function is described here.

1. **Number of Anchor Points**
   (a) (positive integer) Determines the number of interior anchor points placed (that is, X_i for $i = 1, ..., N-1$). This is because the program assumes X_0 and X_N to exist by default.
2. **Number of Iterations**
   (a) (positive integer) Determines the number of times the program should run through the steps detailed in the pseudocode section.
3. **Show Hallway Outlines**
   (a) (true/false) If set to true, displays the outlines of the hallways composing the hallway set while the program runs.
4. **Smooth Sofa Object**
   - (true/false) If set to true, does piecewise linear interpolation between anchor points to redefine the edge of S_N. Note that smoothing just affects what is drawn to the screen.
5. **Hallway Placement (height/width)**
   - The initial placement of the sofa is along the rotation path

     $$x(t) = 2/\pi \cdot (\cos t - 1 + (.5 - a), (.2 - b) \cdot \sin t)$$

     The (height) slider adjusts the value of the b variable while the (width) slider adjusts the value of the a variable.
6. **Preview Sofa**
   - This button allows the user to preview the initial S_N based on the starting conditions set. It's not necessary to preview S_N before running the program.

8 Known Bugs

1. When the number of anchor points is set to 20 or 90 along with the height and width sliders set to their max, shapely is sometimes unable to properly intersect elements of the hallway set resulting in an S_N which does not satisfy the definition given in section *3.1*.