Billiards Everything Instruction

Contents

Chapter 1

Installation

1.1 Prerequisite

The program is working on Windows, Linux Ubuntu 22.04 and Mac with an intel chip.

1.2 Installing instruction

First, download the respective version of Billiards Everything for your OS system here.

1.2.1 For Windows

For windows, there is no need to set things up.

Step 1. Unzip the zip file

Step 2. Double click the billiard-viewer.bat file to run it.

Step 3. There should be a blue window pops up the first time you run it. Click on "more info" and then "run anyway".

1.2.2 For Linux

a. Setup

Step 1. Install java 8u66 development kit from this here. If you do not know how to follow the instructions here.

Note: remember to change all the 251 to 66 as they used java 8u251 as an example in the link

Step 2. Open a terminal by pressing right click on your desktop or Ctrl+Alt+T, then type in:
$ sudo apt install -y libsqlite3-dev libgmp-dev libmpfr-dev libmpfi-dev libboost-all-dev libeigen3-dev libtbb-dev libjemalloc-dev openjfx libcanberra-gtk-module libcanberra-gtk3-module

b. Run

Open a terminal in the unzipped folder, then type in:
$ java -jar billiard-viewer.jar

1.2.3 For Mac with an intel chip

a. Setup

Step 1. Install Java (if you already have java, just check the version as below) in the terminal, enter "java -version" The output should be similar to this:
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b17, mixed mode)
If not, try this: /usr/libexec/java_home -v 1.8.0_66
If it works, add the following to ∼/.zshrc
export JAVA_HOME=$(/usr/libexec/java_home -v 1.8.0_66)
Otherwise you need to install the correct version of Java here (version 8u66). You need to make an account to download it.

Step 2. Use homebrew to install dependencies (enter "brew" in the terminal to see if you already have brew). If don't then you can download it from their website.
Enter these commands in th terminal after you have brew
$ brew install git gmp mpfr boost tbb@2020 jemalloc
$ brew link tbb@2020

Step 3. Create a symbolic link to tbb@2020 (use the command below) by typing the command in the terminal
$ ln -s /usr/local/opt/tbb@2020 /usr/local/opt/tbb

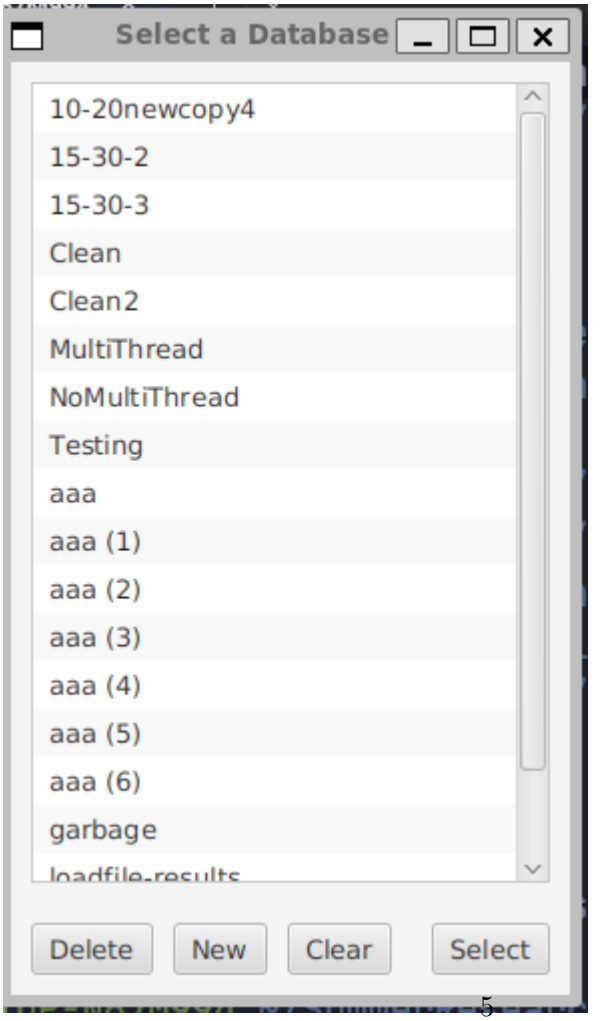
b. Run

Open a terminal, enter "java -jar " and then drag the jar inside the downloaded unzipped folder from sourceforge into the terminal and press enter.

Chapter 2

User's guide

2.1 Database



When you start the program, a small database menu will pop up to allow you to pick a database to store your code sequences using sqlite.

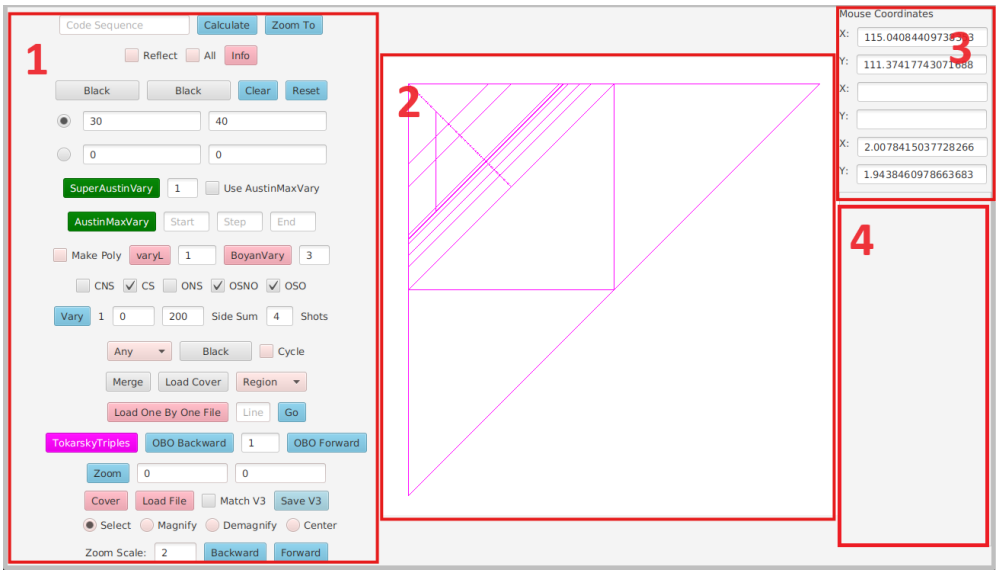
In the image above, the "test" database is already created. If a database has not been created yet, then click on "New" to create a new one.

After creation, to open it, click on that database and then "Select" to use that database.

"Delete" will simply create a database completely, while "Clear" will clear all the contents stored in that database.

2.2 Main interface

2.2.1 Section categorization



1 - Tools: What you can do in this program

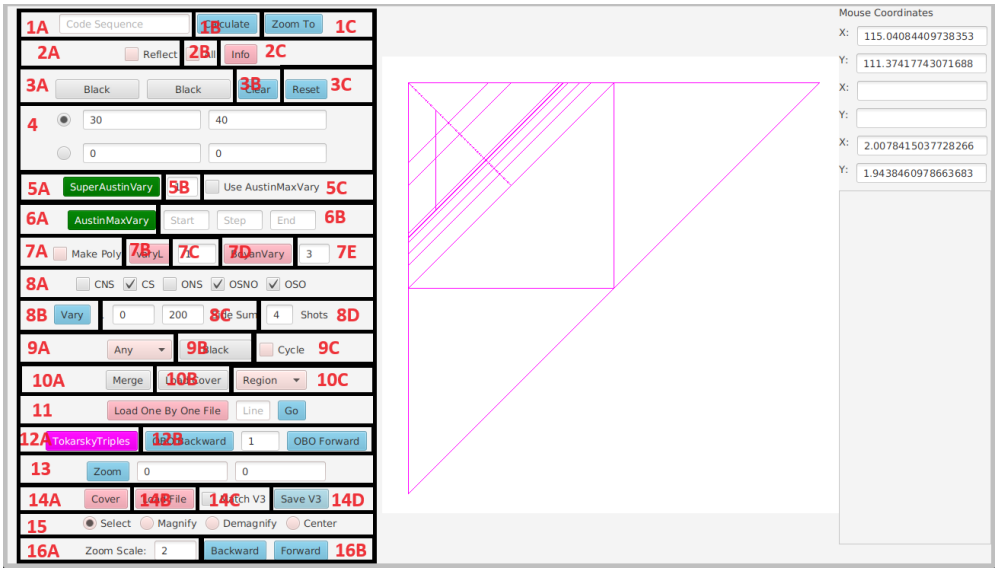
2 - Viewer: Show results

3 - Mouse coordinates: Show your mouse coordinates

4 - Code sequences: Show the code sequences

2.2.2 Tools

In this section, we will focus on what each button do in the tools section.



1. a. Code seqence Box(triples require commas)
   b. Calculate Using Code Sequence, when clicked the Iteration Window will appear
   c. Zoom The Code Sequence
2. a. Reflect the Viewer
   b. Permutes x,y,z coordinates
   c. Information about the Code Inputted
3. a. Colour of the Regions in Viewer
   b. Clears Viewer
   c. Resets to Default
4. X Y Coordinates
5. a. SuperAustinVary
   b. Subdivision step
   c. Use AustinMaxVary with SuperAustinVary
6. a. AustinMaxvary
   b. start end AustinMaxvary
7. a. Creates Polygon Vertices Coordinates
   b. List of Coordinates

c. Maximum Number of Regions Drawn From Coordinates of VaryL
d. Polygon Coordinates
e. Maximum Number of Subdivisions

8. a. Code type
   b. Finds Codes and the number represents the Number of Codes Found after done
   c. Min Max Side Sum
   d. Number of Shots

9. a. Total Number of Selected Polygons
   b. Colour of Cover Squares
   c. Cycles the Colours in the Cover

10. a. Merges Abutting Covers
    b. Loads a Previous Cover
    c. Draws Selected Bounding Polygon

11. Load One By One File

12. a. Triples, when clicked the Triple Window will appear
    b. Direction of One By One

13. Center the Viewer at the Coordinates

14. a. Cover a Polygon with Stables and Triples, when clicked the Cover Window will appear
    b. Draws Codes From a Selected File
    c. Find Intersecting Code From ClickedLocation
    d. Save Code

15. Select and Press on Viewer

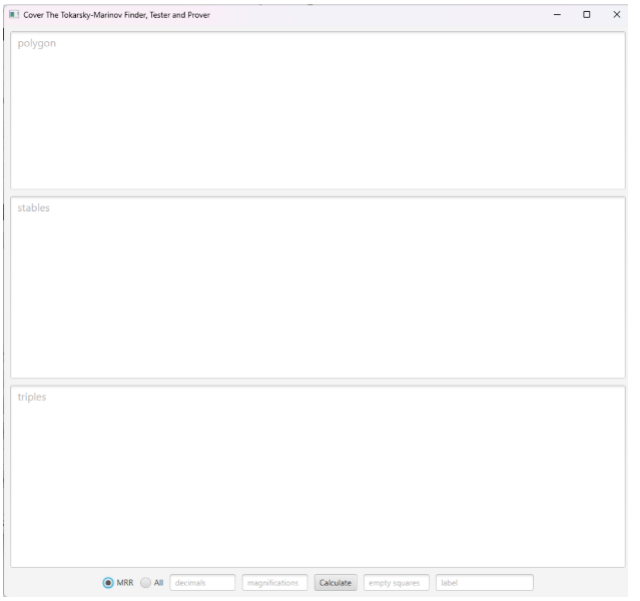
16. a. Zoom Times Size
    b. Undo Redo Viewer

2.3 Iteration Window

1. a. Increase/Decrease code Individually By 2
2. Add 2/Subtract 2 at specified positions (for triples separate by comma)
3. a. Put in specified positions (for triples separate by comma)
   b. Put in specified number of iterations
   c. Put in specified increment
4. a. Put in valid horizontal pattern code
   b. Put in specified number of iterations
   c. Put in specified pattern of each element
   d. Press Button to calculate expando iterations
5. Code Sequence Box (triples require commas)
6. Label (Not required)
7. Options for the iterations (read instructions)
8. Calculate iterations using 3 and 7

2.4 Cover

In Cover, you want to have a polygon, which is the region you want to cover it completely with periodic paths, hence the name Cover.



- Polygon: the area you want to cover, for example

  5 5

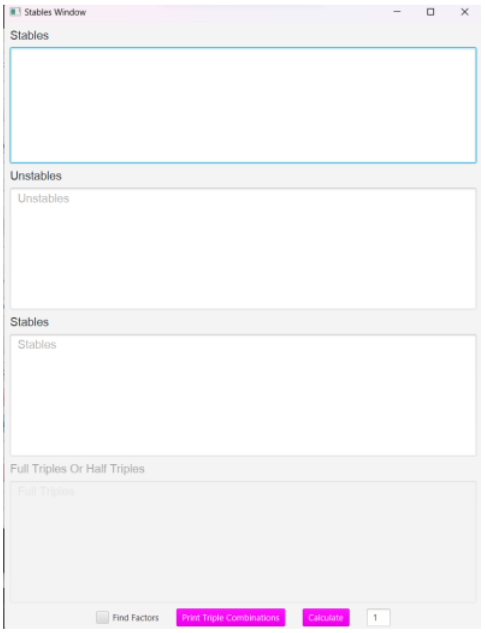
  5 6

  6 5

  is the area of the triangle with those 3 vertices.

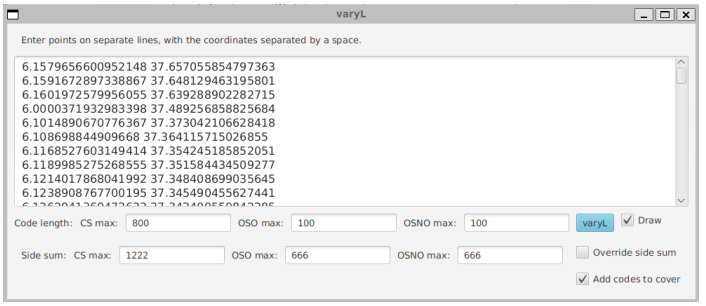
- Stables: side sequences that cover some part of the region
- Triples: triples that cover some part of the region
- MRR/All: Choosing the all options from the radio button to find all equations including the MRR equations and guarantees every square will satisfy the gradient algorithm.
- decimals: how accurate you want to be
- magnification: square division
- Calculate button: start covering the region with the code you put in stables and triples.
- empty squares: the number of empty squares that you are looking to find
- label: label your progress

2.5 Triple Window



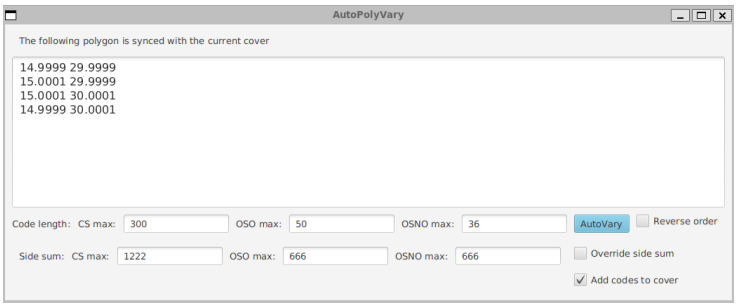
The triple window will pair up 2 stables with 1 unstables to find a triple.

2.6 VaryL Window



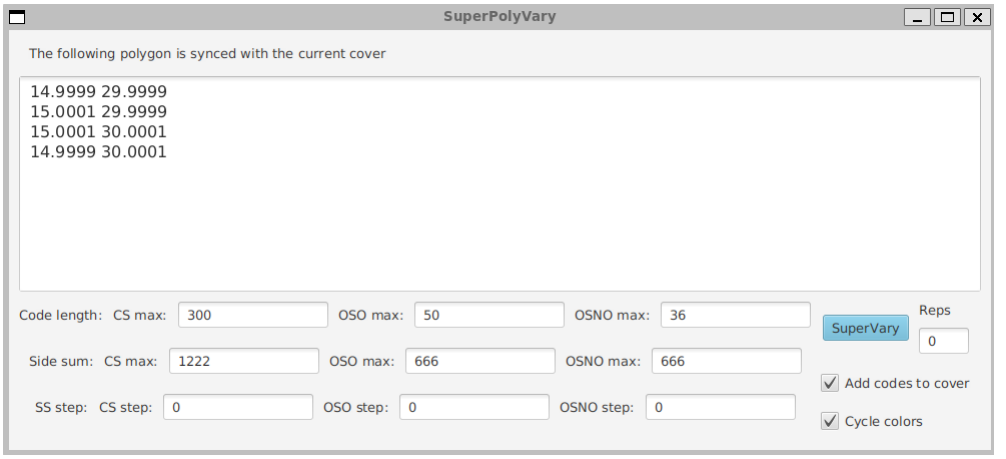
To run VaryL, you first have look at this. You will see a set of coordinates after pressing Calculate. Copy those coordinates into the box and press the varyL button as the program will run vary on all those coordinates. Codes that are found will go directly into your cover. You can also specify seperate maximums for side sum and code length. To override side sums, you must check the box labeled "Override side sum".

2.7 AustinMaxVary Window



To run AustinMaxVary, you first have to look at this. You will see a set of coordinates after pressing Calculate. Put those coordinates in an empty file and then use "Load one by one file" in the tools menu to load that file. Then open the AustinMaxVary window and press AutoVary as the program run BoyanVary on all the coordinates in the file. Codes that are found will go directly into your cover. You can also specify seperate maximums for side sum and code length. To override side sums, you must check the box labeled "Override side sum".

2.8 SuperAustinVary Window



SuperAustinVary allows you to run either BoyanVary or AustinMaxVary multiple times. This is useful for automating large portions of covers. First put a non-negative integer in the box labeled "Reps" to denote how many times you want it to run. Then set the initial code length and side sum, as well as the step, which will be added to the initial side sums after each

iteration. The initial values must be non-negative integers, the step can be any integer, including negative values. Codes that are found will go directly into your cover.

Chapter 3

How to be a periodic path hunter

This chapter will assume you have contacted George Tokarsky. He will give you a coordinates so that you can find the perdiodic path in the given space.

3.1 Identify your given open space

Step 1. Your polygon (or quadrilateral) coordinates will be given as below when you contacted George Tokarsky or you can just create one independently, they are the coordinates of the vertices of your polygon. For example:

8 58

9 57

9 55

8 56

Step 2. Now go to the tools menu and click on "Cover" in the bottom left region. These polygon coordinates should then be copied and pasted into the first big input box with the hint "Polygon". This is the first of three big input boxes titled polygon, stables, triples.

Step 3. Look towards the bottom of the Cover pop-up and make sure MRR is selected and use these suggested numbers as follows 22 20 [Calculate] 50 respectively.

Calculate is used to outline your polygon and to find uncovered holes in the viewer. The 3 numbers around the calculator button are specifications.

First small box is the number of decimal places, which you do not need to change.

Second small box is the number of subdivisions, which you may need to change or increase towards the end of covering the polygon.

Third small box is the list of point coordinates displayed where those points are not covered. Generally 50-100 points is good enough for a single session to start. Eventually you will need to use thousands of points and leave the program run overnight or longer.

3.2 Filling in spaces/hole (Stables)

There are many ways of filling empty space. The possible ways of filling spaces will be loosely ordered by the size of the holes to fill.
Most values, such as subdivisions and side sum or code sum are dependent on your processor specs. Loose ranges and example values will be called low, medium or high and will be specified in the steps below. Here, we will introduced severeal tools to fill in spaces/hole

3.2.1 Menu - PolyVary

Generally used for a new polygon.

Step 1. Subdivision should be set to a low-medium range (3 - 5).

Step 2. Side sum should be of medium range (800 - 1200).

Step 3. CS should be selected first.

Step 4. Do not forget to change your number of cores beside "Shots".

Step 5. Click on BoyanVary and put in the given polygon coordinates appearing there and press the Vary button seen inside.

Step 6. Copy all code sequences that show up in the console.

Step 7. Paste these code sequences into the second big input box under Cover (if it is empty then there will be a hint text that says stables.)

Note: Increasing the subdivisions, side sum or code sum can help find smaller holes, as well as zooming to enlarge the viewer with empty space which can help to find more code sequences but in return will take longer.

3.2.2 Menu - Side Sum

When the side sum is over 1000 or so, it could be time to increase your point coordinates into the thousands.

Step 1. Put your coordinate list inside varyL and press the varyL button inside.

Step 2. Make sure that you check the box on Draw.

Note: the max boxes are in code sums not side sums.

A typical start is CS max 333 and increase that as needed up to the 500's.

For the OSO max use a range of 122-166 or so and for the OSNO use a range of 50-136 or so.

Caution: The checkboxes in the Tools Menu should be on for the CS first, then with the CS and OSO on and then lastly with the CS,OSO and OSNO on.

3.2.3 Menu - Match V3/Save V3

This tool can be used to find holes individually. It should be used when almost finished with the cover but can be used whenever the user likes to.

Step 1. Side sum should be set to between 2000 to 5000

Step 2. Make sure that CS is the only code type selected as the other two types take much more time.

Step 3. Zoom in enough to see the hole.

Step 4. Make sure Match V3's checkbox is checked. Click Select as the radio option.

Step 5. Click on the corners of the given empty space on the viewer. Match V3 will find the common sequences of all the coordinates clicked on the viewer.

Step 6. Next, check the console for something similar:
//————————- Matching Code Sequences ————————-//

CS (56, 384) 1 1 6 2 20 2 9 1 2 1 19 2 8 2 19 1 2 1 9 2 20 2 6 1 1 19 2 8 2 20 2 8 2 20 2 7 1 3 20 2 8 2 20 3 1 7 2 20 2 8 2 20 2 8 2 19

CS (88, 412) 1 1 4 1 1 18 1 1 4 1 1 18 1 1 4 1 1 18 1 1 4 1 1 18 1 1 4 1 1 18 1 2 1 10 1 2 1 19 3 1 7 2 19 1 2 1 10 1 2 1 18 1 2 1 10 1 2 1 18 1 2 1 10 1 2 1 19 2 7 1 3 19 1 2 1 10 1 2 1 18

...

Step 7. Copy a single code sequence and put it in the sequence drawer and click on "Calculate".

If the code sequence covers what you wish it to cover, then put it in the "Stables" input box in Cover(second input box).

If it does not cover what you wish, you can either try the next matching sequence or you can keep the sequences and repeat this process for the smaller empty space.

Alternatively you can also save them all together in any file you choose by pressing Save V3 and load

them all when ready by pressing the load button.

If the hole looks like a line, you can click on both sides of the line to see if there is any matching code

sequences. You can also try using a triple below.

3.3 Filling in spaces/holes (triples)

These holes are very long rectangles that look like lines. A triple is just a combination of stable, unstable, stable codes.

Example of a triple with commas:

1 1 10 2 23, 1 2 1 13 2 20 2 13, 1 1 12 2 20 2 13 1 2 1 13 2 20 2 12 1 1 23 2 12 2 20 2 12 2 23

To find the two stables, click one side of the line and then the other side and you should get the two stables as long as they intersect in a common boundary line.

To find the unstable involves loading low side sum (100 should be good enough) CNS and ONS and

connecting the existing sequences to the left and right of the CNS/ONS sequence found.

To find the unstable use the two radio buttons next to the two rows of coordinates near the top in the Tools

Menu.

1. Click on the first radio button and click on one end of the line (a close approximation is all you need)

and its coordinates will appear.

2. Click on the second radio button and click on the other end of the line and its coordinates will appear.

3. Change the Side Sum to 144 or so and only have the CNS and ONS check boxes selected.

4. Press Vary and you will see a list of unstables in the console.

5. Copy all the code sequences found into a text file which we will call lines.txt

6. Click "Load File". Find and click on lines.txt. (some will say empty and others don't work).

7. Go back to the viewer and click anywhere on the given line. Find any CNS or ONS that appears in the

Coordinate/Codes Column underneath the coordinates.

If none appears, try again with a bigger Side Sum and more shots. Hopefully you will find one and you

can take the first unstable in the list.

Now go to the Tools Menu under Cover, paste the sequence stable, unstable, stable in the third input box

as with the example and with the hint triples.

To double check you can copy the triple sequence into the box next to Calculate in the Tools Menu to see

if the line has been filled or partially.

Alternatively if using a triple, you can try using a stable OSO or OSNO to cover that line.

3.4 Finding more holes to fill

After following Identify your given open space.

Step 1. Go to "Cover" in the bottom left of the tools menu and click on the Calculate button

Step 2. Go to the console. Within the generated text should be something similar

1653711 stable squares used in the cover

0 triple squares used in the cover

2310 stables used in the cover

0 triples used in the cover

MRR at 22 decimals, deepest magnification 27

Total stable cost: 831208131

485334 squares were not filled in

8.2754087448120117 57.724614143371582

8.2933473587036133 57.561535835266113

...

Not Covered

Time elapsed: 22m 20.514s

Step 3. Copy all pairs of coordinates and paste into an accessible text file.

Step 4. From the Tools Menu, go to "Load One By One File" and choose the text file that contains all the copied coordinates. The viewer will automatically move to the first coordinate in the text file. To go to the next pair of coordinates, click on "OBO Forward". Repeat filling the holes until the given polygon is covered. When done the console should say "Covered" like this:

// 0 squares were not filled in

// 1111 stable squares used in the cover

// 24 triple squares used in the cover

// 31 stables used in the cover

// 2 triples used in the cover

// MRR at 7 decimals, 1x1 number of squares, deepest magnification 17

// Total stable cost: 65809

// Covered

// Time elapsed: 0.128s

Note: you can used AutoPolyVary as well as VaryL to do the steps automatically.

3.5 What should a great periodic path hunter do next

Go to the jar called Billiards Everything and you will find a folder called Cover. Send that to me and I will check it and run it to the All proof that goes through all the equations and we will add your name, date and cover to the Great Hunt. You can do that yourself if you wish by checking the All check box instead of the MRR check box. If you do, be prepared that

it could take 20 times or more and days or weeks or months or years. To ensure that the polygon assigned is full look for two things:

// 0 squares were not filled in
// 1111 stable squares used in the cover
// 24 triple squares used in the cover
// 31 stables used in the cover
// 2 triples used in the cover
// MRR at 7 decimals, 1x1 number of squares, deepest magnification 17
// Total stable cost: 65809
// Covered
// Time elapsed: 0.128s

This shows that your polygon is covered and I will run the proof to ensure that the polygon has been fully covered.

3.6 Bonus fun on iterations

Iteration

When you put in a code which can be a stable, a non-stable or a triple and press Calculate, the iterations bar comes up.

If you use a triple, you will see six rows of boxes, three green and three pink. Otherwise you will see two rows of boxes. Pressing a green increases the code by two and pressing the pink decreases the code by two.

Alternatively you can use the Add 2/Subtract 2 boxes to change a given code. In the empty box to the left, if you put in for example 3 6 10 then the 3rd, 6th and 10th code number will add 2 or subtract 2 to those spots.

Similarly for triples if you put in 3, 2, 3 5 and press add 2 it will add 2 to the 3rd spot of the first stable, add 2 to the 2nd spot of the unstable and add 2 to the 3rd and 5th spots of the other stable.

You can also create patterns using the next three horizontal lines of boxes. For example using the First Pattern box and putting in 3 6 10 for those spots, and putting 12 in the second box will give the number of iterations and putting 6 in the third box will give the increment to all of those spots. Then press the button Calculate Iterations and you will see in the viewer all those regions belonging to all of those codes.

Furthermore, there are also four options you can choose from for calculating iterations:

First is the 'no left rights' option which means it doesn't use any previous left rights equations in the calculation for the following iterations, and then each time it calculates and uses the current left rights equations which will take the longest time.

The second is ‘show left rights’ which does the same as the first but will also print out the left rights for each code in the iterations in the console.

The third option is ‘use left rights without test’ which means using the left rights found in the previous iterations in the calculation for the following iterations and thus it does NOT always give you the correct result. The trade off is if you trust the pattern, then this is the fastest option.

The last option is ‘use left rights with test’ which means using left rights found in the previous iteration in the calculation for the following iterations with a test. This test can filter out many wrong results and then use the normal method to calculate instead. It is the second-fastest option but again is not foolproof.

NOTE: the third and the fourth option works with only the stable codes and doesn’t work for the unstable codes.

Expando Iterations

Expando iterations are used when the code sequences expand horizontally in a pattern and where the left rights of the code sequences also have a pattern. Here is an example:

1 1 2 2 1 1 3 3
1 1 2 2 2 2 1 1 3 2 2 3
1 1 2 2 2 2 2 2 1 1 3 2 2 2 2 3
... Left Right
(2, 0, 7, 0)
(8, 0, 5, 0)
(8, 0, 1, 0)
(4, 0, 1, 0)
Left Right
(2, 0, 9, 0)
(10, 0, 7, 0)
(10, 0, 1, 0)
(4, 0, 1, 0)
Left Right
(2, 0, 11, 0)
(12, 0, 9, 0)
(12, 0, 1, 0)
(4, 0, 1, 0)
...

To use and press the Calculate expando iteration button, put in the expando pattern box a pattern substituted with the capital letters.

For example input 1 1 2 2 A 1 1 2 2 B 1 1 2 2 A in the Expando pattern box and put in the elements box for example: 2 2,2 2 separated by commas with A becomes 2 2 and B becomes 2 2. This also allows expanding patterns involving A,B,C,...

The Expando iteration has only two options to use:

a. use 'left rights with test' means all the code sequences in the iterations are calculated the normal way which is the slower method and this acts as the test.

b. use 'left rights without test' means using the left rights found in the previous iteration in the calculation for the following iterations. Again this is not foolproof