

Billiards Everything Instruction

Contents

1	Installation	3
1.1	Prerequisite	3
1.2	Installing instruction	3
1.2.1	For Windows	3
1.2.2	For Linux	3
1.2.3	For Mac with an intel chip	4
1.2.4	For Mac with an arm chip	4
2	User's guide	5
2.1	Database	6
2.2	Main interface	7
2.2.1	Section categorization	7
2.2.2	Tools	7
2.3	Iteration Window	10
2.4	Cover	11
2.5	LiCover	13
2.6	LiPattern Iteration Window	13
2.7	Triple Window	15
2.8	VaryL Window	15
2.9	MVL Window	16
2.10	LiLuMaxVary Window	17
2.11	SuperLiLuVary Window	18
2.12	LiPattern Calculator	19
2.13	LiBain Tetra/Bar	20
2.14	LiCycle	21
2.15	JKFindHoles	22
2.16	JKPatch	23
2.17	JKAUTOtriple	25

3	How to be a periodic path hunter	27
3.1	Identify your given open space	27
3.2	Filling in spaces/hole (Stables)	28
3.2.1	Menu - PolyVary	28
3.2.2	Menu - Side Sum	29
3.2.3	Menu - Match V3/Save V3	29
3.3	Filling in spaces/holes (triples)	30
3.4	Finding more holes to fill	31
3.4.1	Using the Cover tool	31
3.4.2	Using the Hole Finder tool	32
3.5	What should a great periodic path hunter do next	33
3.6	Bonus fun on iterations	33

Chapter 1

Installation

1.1 Prerequisite

The program is working on Windows, Linux Ubuntu 22.04 LTS and Mac.

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

Step 1. Ubuntu 18.04LTS is not supported. Check your system using `$ cat /etc/os-release`

Step 2. Unzip the zip file

Step 3. Open a terminal, locate yourself inside the folder, and then type in:
`$ chmod +x ./run.sh java/bin/*`

Step 4. launch the programme: `$ ./run.sh`

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.

1.2.4 For Mac with an arm chip

a. Setup

Step 1. Download and install the installer

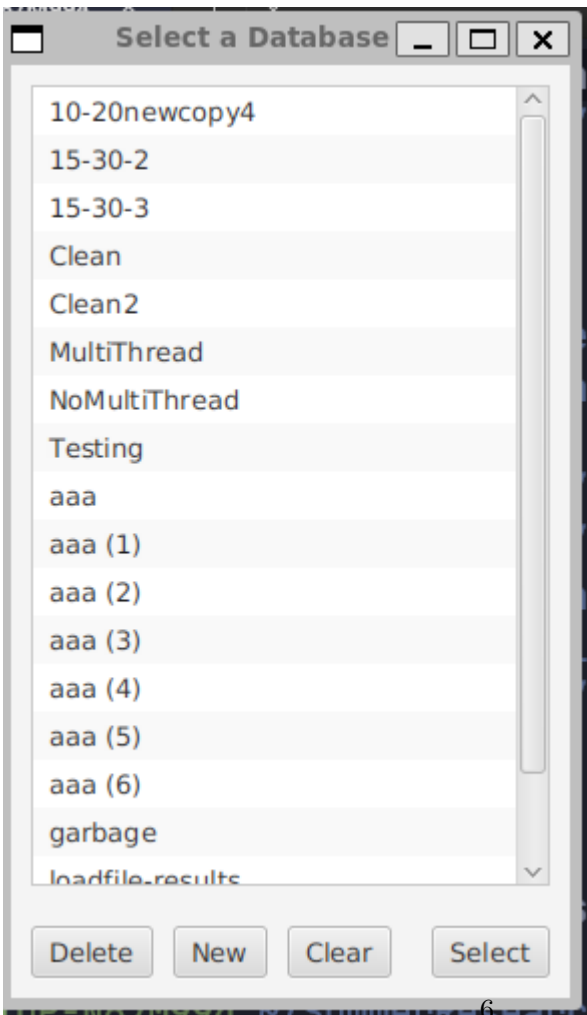
b. Run

Go to application and right click on BilliardViewer and open a terminal at the folder. Next navigate using command `cd Contents/MacOS/` Next open application with command `./BilliardViewer` Allow appropriate permission in case system blocks it.

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.

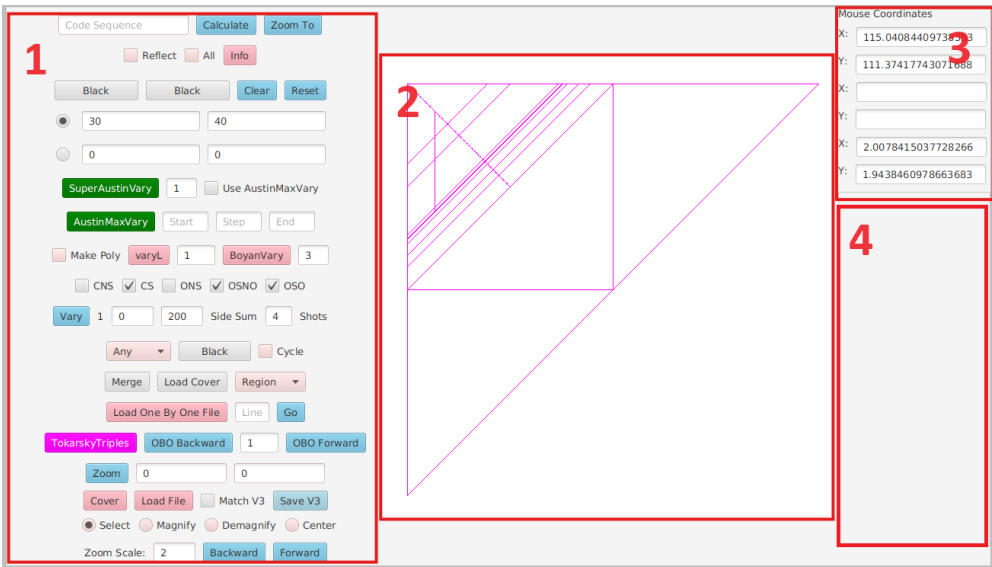
In the image above, the "test" database is already created. If a database has not been created yet, then click on "New" to create a new one.

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

"Delete" will simply delete 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

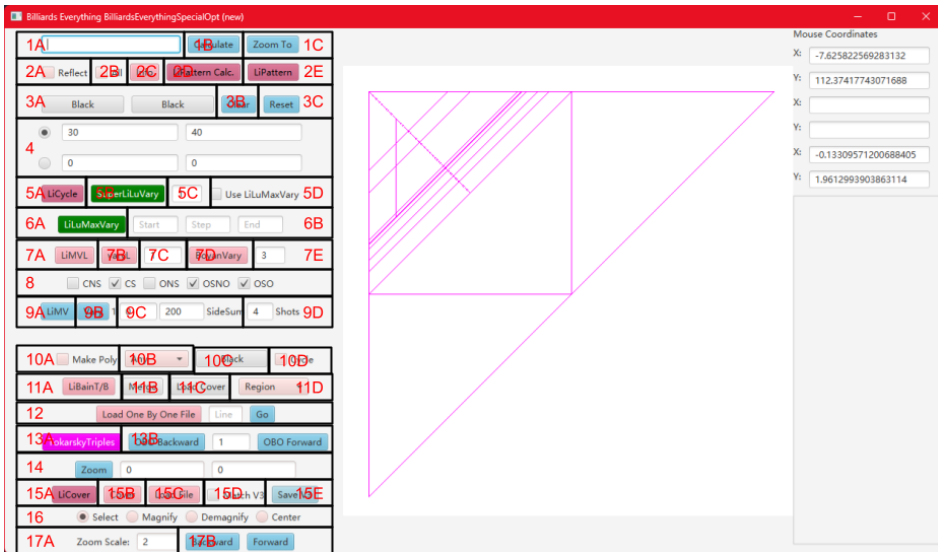
2 - Viewer: Show results

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.

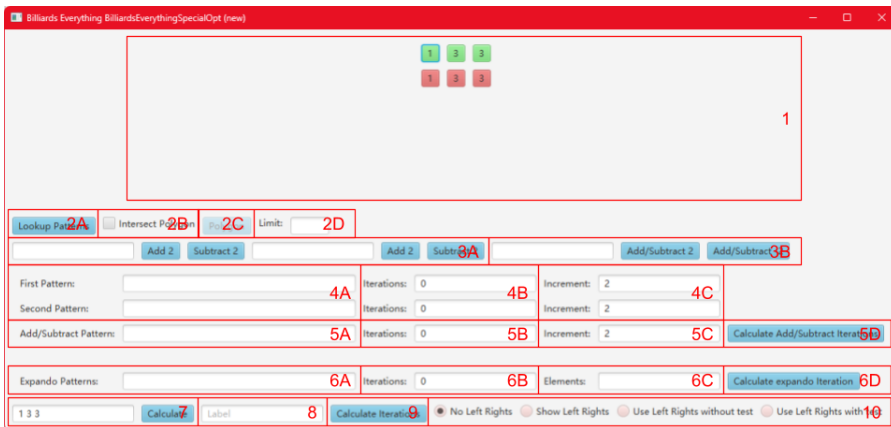


1.
 - a. Code sequence Box(triples require commas)
 - b. Calculate Using Code Sequence, when clicked the Iteration Window will appear
 - c. Zoom The Code Sequence
2.
 - a. Reflect the Viewer
 - b. Permutes x,y,z coordinates
 - c. Information about the Code Inputted
 - d. Li Pattern Calculator
 - e. Li Pattern Iterator
3.
 - a. Colour of the Regions in Viewer
 - b. Clears Viewer
 - c. Resets to Default
4. X Y Coordinates
5.
 - a. LiCycle
 - b. SuperLiLuVary
 - c. Subdivision step
 - d. Use LiLuMaxVary with SuperLiLuVary
6.
 - a. LiLuMaxVary

- b. start end LiLuMaxVary
- 7. a. LiMVL
 - b. VaryL
 - c. Maximum Number of Regions Drawn From Coordinates of VaryL, or maximum number of groups to print for Li1MVL.
 - d. BoyanVary
 - e. Maximum Number of Subdivisions for BoyanVary
- 8. Code type
- 9. a. LiMV. Same as Vary but prints only the first, middle, and last code sequences of each group. Code sequences that have the same code type, code length, and odd-even pattern belong to the same group.
 - b. Finds Codes and the number represents the Number of Codes Found after done
 - c. Min Max Side Sum
 - d. Number of Shots
- 10. a. Creates Polygon Vertices Coordinates
 - b. Total Number of Selected Polygons
 - c. Colour of Cover Squares
 - d. Cycles the Colours in the Cover
- 11. a. LiBain Tetra/Bar
 - b. Merges Abutting Covers
 - c. Loads a Previous Cover
 - d. Draws Selected Bounding Polygon
- 12. Load One By One File
- 13. a. Triples, when clicked the Triple Window will appear
 - b. Direction of One By One
- 14. Center the Viewer at the Coordinates
- 15. a. LiCover
 - b. Cover a Polygon with Stables and Triples, when clicked the Cover Window will appear
 - c. Draws Codes From a Selected File

- d. Find Intersecting Code From Clicked Location
 - e. Save Code
16. Select and Press on Viewer
 17. a. Zoom Times Size
 - b. Undo Redo Viewer

2.3 Iteration Window



1. Increase/Decrease code Individually By 2
2.
 - a Look up the iteration patterns used with the current code sequence in the past.
 - b Intersect with a polygon. Code sequences that intersect with the polygon will be automatically added to the cover along with their iteration pattern.
 - c Open a new window to specify the polygon to check intersections.
 - d When calculating iterations, the maximum number of codes to add to the cover.
3.
 - a Add 2/Subtract 2 at specified positions (for triples separate by comma)
 - b Add 2/Subtract 2 and Add -2/Subtract -2 at specified positions (for triples separate by comma)
4.
 - a. Put in specified positions (for triples separate by comma)
 - b. Put in specified number of iterations

- c. Put in specified increment
- 5. a. Put in specified positions (for triples separate by comma)
 - b. Put in specified number of iterations
 - c. Put in specified increment
 - d. Press the button to calculate add/subtract iterations
- 6. a. Put in valid horizontal pattern code
 - b. Put in specified number of iterations
 - c. Put in specified pattern of each element
 - d. Press the button to calculate Expando iterations
- 7. Code Sequence Box (triples require commas)
- 8. Label (Not required)
- 9. Calculate iterations using 3 and 7
- 10. Options for the iterations (read instructions)

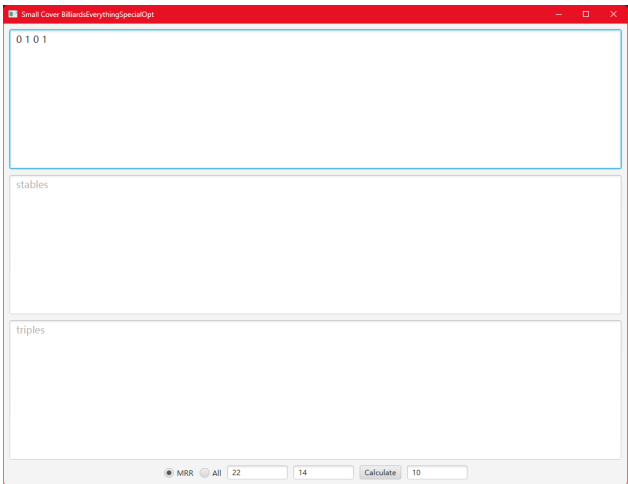
2.4 Cover

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
- Add to Small Cover: add empty squares to LiCover

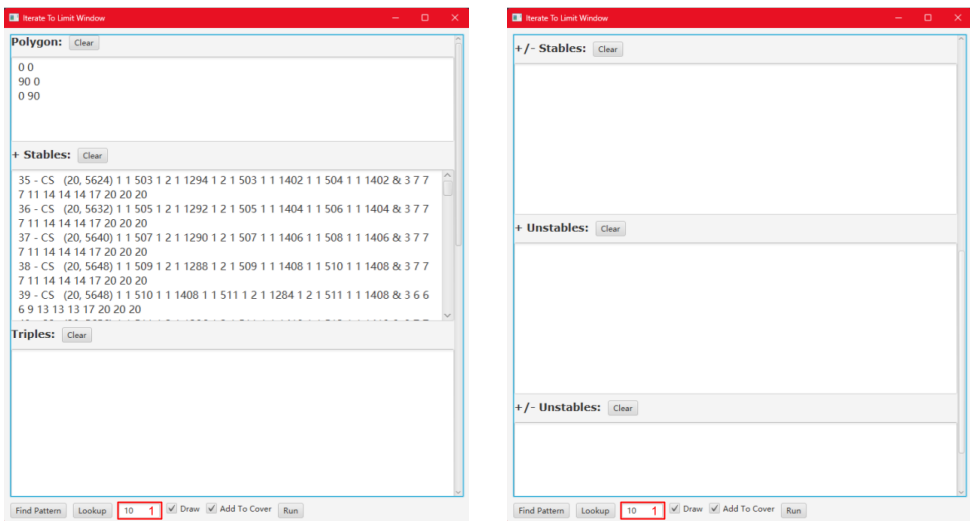
2.5 LiCover



Everything in the LiCover is the same as in Cover, except for the representation of the polygon, or more accurately, the squares. The squares in LiCover are each represented by four numbers, where each number is a fractional value of $\frac{\pi}{2}$. The first two numbers are the two boundary points of the x-interval of the square, whereas the last two numbers are the two boundary points of the y-interval. In addition, you can put multiple squares, each occupying a separate line. The LiCover is especially useful for checking whether a small empty square has been covered without running the check on the entire region.

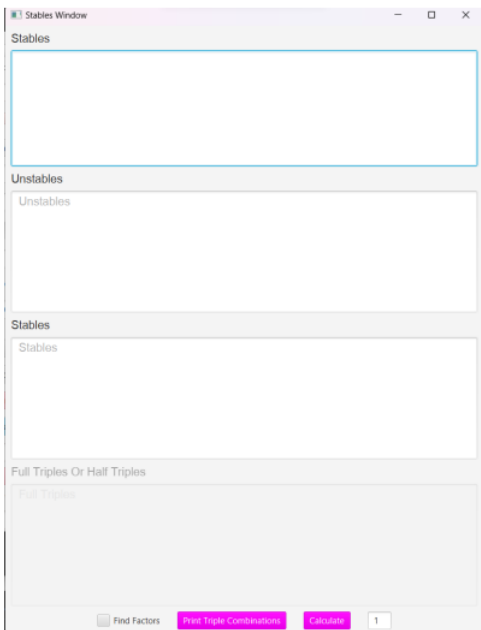
2.6 LiPattern Iteration Window

1. Find iteration patterns for code sequences.
 - a Put stable code sequences in the "+ Stables" text box if you want LiPattern to find all positive iteration patterns for the stable code sequences; put them in the "+/-" text box if you want LiPattern to find plus/minus iteration patterns.



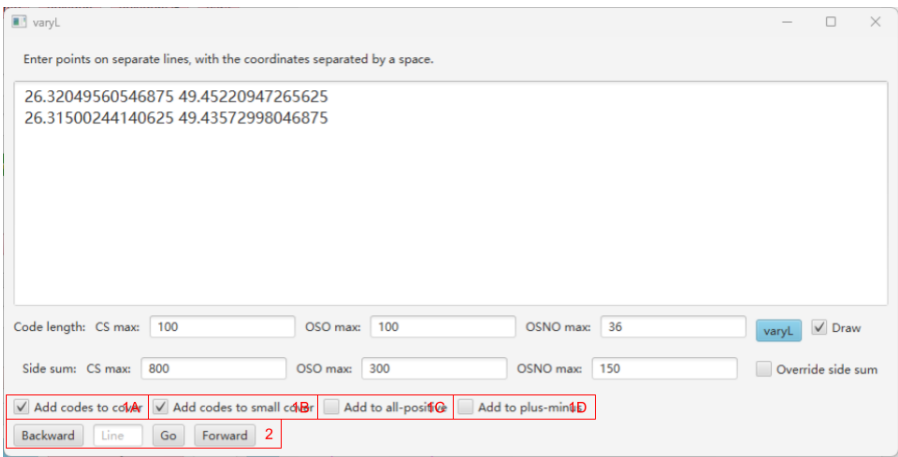
- b Do the same for unstable code sequences. Use the "+ Unstables" and "+/- Unstables" text boxes.
 - c Put triples in the "Triples" text box.
 - d Press the "Find Pattern" button. The iteration patterns will appear behind each code sequence, prefixed by a "&".
 - e You can also look up previously used code sequence-iteration pattern pairs using the "Lookup" button.
2. Enter the coordinates of the polygon you wish to cover.
3. Enter the iteration limit (Element 1 in the above figures). This is the maximum number of times you want the iteration patterns applied to the code sequences.
4. Press the "Run" button. For each code sequence, LiPattern will apply its iteration pattern for the maximum number of times that was specified in the previous step both in the forward direction (addition) and the backward direction (subtraction). Each new code sequence that intersects with the polygon will be drawn on the viewer and added to the cover.

2.7 Triple Window



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

2.8 VaryL Window

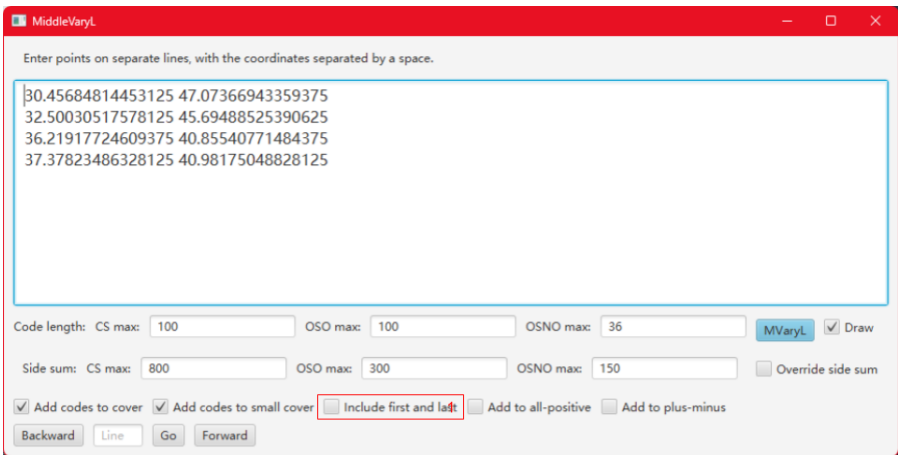


To run VaryL, you first have look at this. You will see a set of coordinates after pressing Calculate. Copy those coordinates into the box and press the varyL button as the program will run vary on all those coordinates. Codes that are found will go directly into your cover. You can also specify separate

maximums for side sum and code length. To override side sums, you must check the box labeled "Override side sum".

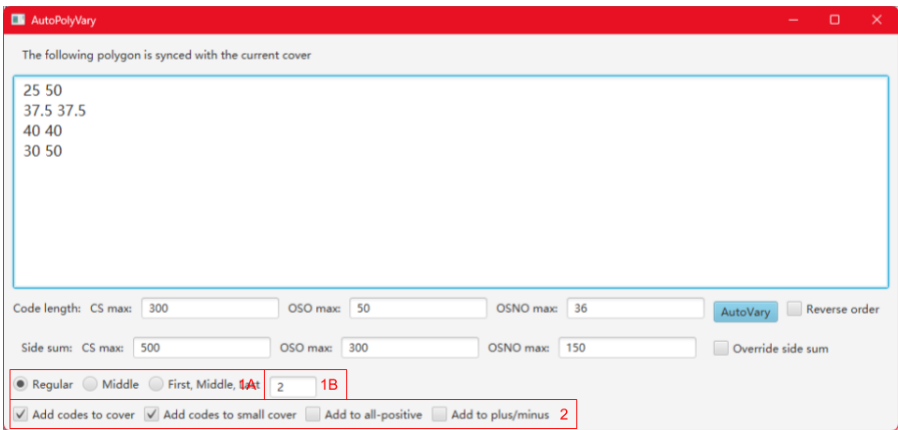
- 1. a Add code sequences to the Cover.
 b Add code sequences to LiCover
 c Add code sequence-iteration pattern pairs to the "+ Stables" text box of LiPattern Iteration Window
 d Add code sequence-iteration pattern pairs to the "+/- Stables" text box of LiPattern Iteration Window
- 2. Same as Tools 12B

2.9 MVL Window



Everything is the same as VaryL Window except that it separates code sequences into groups that are characterized by code type, code length, and odd-even pattern, and only prints the middle code sequence of each group (as well as the first and last code sequences if "Include first and last" is checked). The number of groups to print is specified Tools 7C.

2.10 LiLuMaxVary Window



To run LiLuMaxVary, 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 LiLuMaxVary 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 separate maximums for side sum and code length. To override side sums, you must check the box labelled "Override side sum".

1.
 - a
 - i. Regular: Print everything.
 - ii. Middle: Same as MVL.
 - iii. First, Middle, Last: Same as in MVL with "Include first and last" checked.
 - b For "Regular", the maximum number of code sequences to print. For "Middle" and "First, Middle, Last", the maximum number of groups to print.
2. Same as VaryL Window 1

2.11 SuperLiLuVary Window

SuperPolyVary

The following polygon is synced with the current cover

25 50
37.5 37.5
40 40
30 50

Code length: CS max: 300 OSO max: 50 OSNO max: 36

Side sum: CS max: 500 OSO max: 300 OSNO max: 150

SS step: CS step: 0 OSO step: 0 OSNO step: 0

SuperVary

Reps
0

☒ Add codes to cover

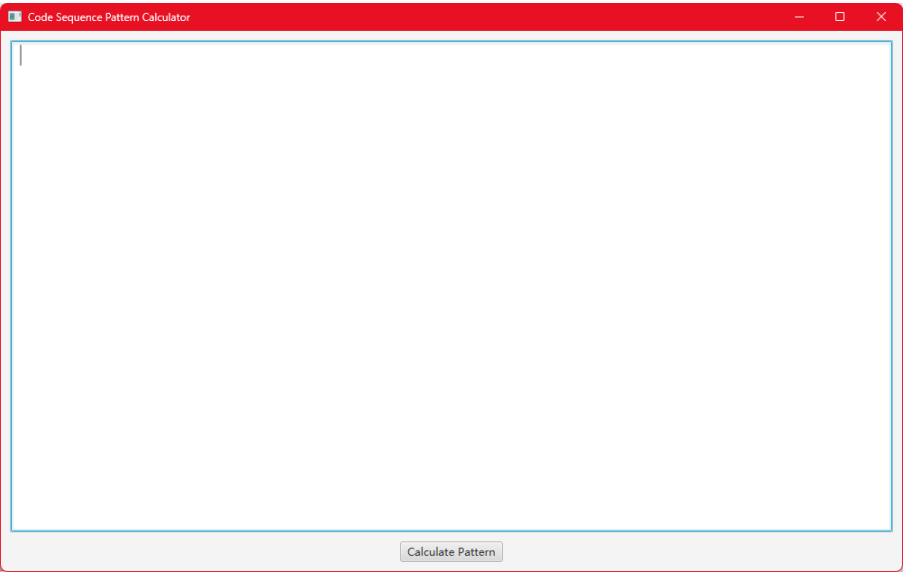
☒ Add codes to small cover

☒ Cycle colors

☐ Magnification: 2

SuperLiLuVary allows you to run either BoyanVary or LiLuMaxVary multiple times. This is useful for automating large portions of covers. First, put a non-negative integer in the box labelled "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 be added directly to your cover. You can also check the check box labelled "Magnification" to magnify the viewer by the specified factor after every rep. Magnifying the viewer is especially useful when you are trying to fill in smaller and smaller holes.

2.12 LiPattern Calculator



Calculates the iteration pattern between two code sequences with the same code length and odd-even pattern. Put the two code sequences on two separate lines in the text box, and press "Calculate Pattern". The iteration pattern will be printed on the console.

2.13 LiBain Tetra/Bar

Enter coordinates on each line. The x and y coordinates should be separated by a whitespace.

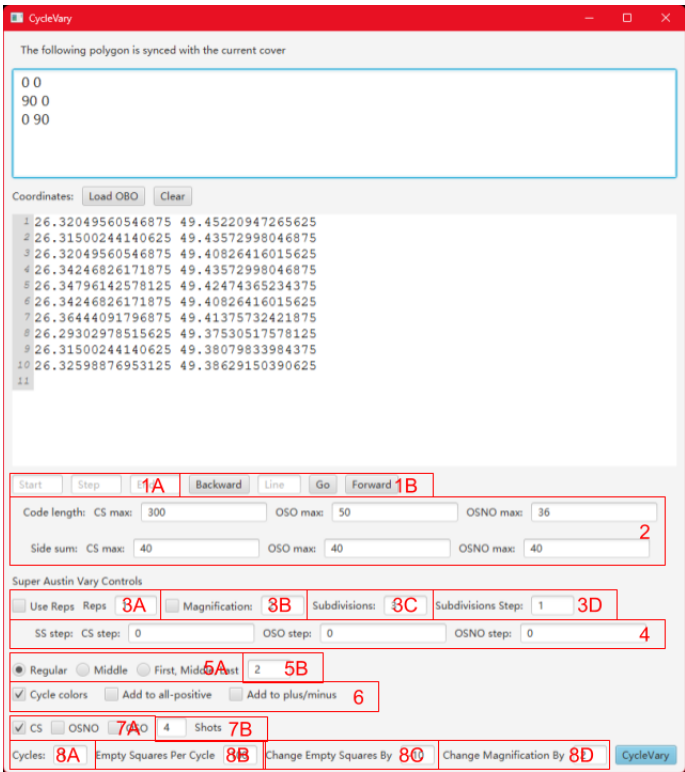
30 40

0.00000001 1 10 2 ☒ Tetra ☐ Bar ☒ Draw ☒ Add to cover Calculate

1. Offset: the value to offset the original coordinate by to create new points.
2. The maximum number of codes to print.

Enter the coordinates of the points you want to cover in the text box and press "Calculate". Vary will run on the two (if "Bar" is selected) or three (if "Tetra" is selected) new points that were created using the offset value. This is useful when you cannot find code sequences that cover the original point, but code sequences that cover the points around it also cover the original point.

2.14 LiCycle

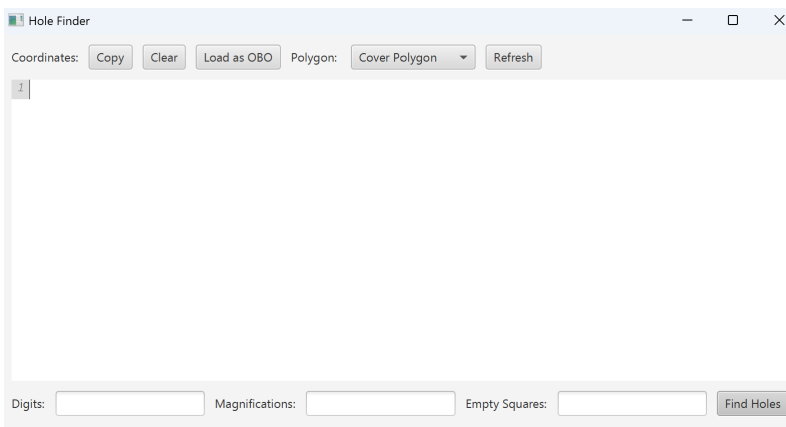


LiCycle performs either LiLuMaxVary or SuperLiLuVary on the coordinates of the empty squares each cycle. At the end of each cycle, coordinates in the "Coordinates" text box will be replaced by new coordinates (if there are more empty squares to be filled).

1. a Same as Tools 6B
 b Same as Tools 13B
2. Same as LiLuMaxVary Window
3. a Same as Reps in SuperLiLuVary Window
 b Same as Magnification in SuperLiLuVary Window
 c Same as Tools 7E
 d Same as Tools 5C
4. Same as the steps in SuperLiLuVary Window
5. a Same as LiLuMaxVary Window 1A

- b Same as LiLuMaxVary Window 1B
- 6. Same as VaryL Window 1C and 1D.
- 7.
 - a Same as Tools 8
 - b Same as Tools 9D
- 8.
 - a Maximum number of cycles.
 - b Maximum number of new coordinates at the end of each cycle.
 - c Changes the number of coordinates in the next cycle.
 - d Changes the magnification value of Cover at the end of each cycle.

2.15 JKFindHoles



JKFindHoles is a tool that can be used to find holes in a cover. It is especially useful when you have a large cover and want to find uncovered empty squares without the heavy calculations using Cover.

The Hole Finder Window uses the exact same algorithm as Cover to divide the polygon and find holes.

The **Coordinates** ribbon at the top of the window provides functionality for using your generated list of hole coordinates.

1. The **Copy** button copies the coordinates of the holes from the textbox to the clipboard.
2. The **Clear** button clears the contents of the textbox
3. The **Load as OBO** button saves the coordinates to a temporary file and loads them into the Tools Menu as a one-by-one file.

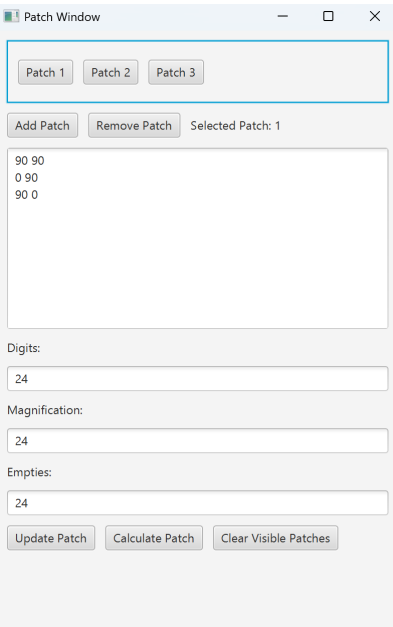
The **Polygon** ribbon at the top of the window provides functionality for specifying the polygon used during hole calculation. The dropdown menu will list the "Cover Polygon" and any patches that have been created. Always use the **Refresh** button after updating/creating a new patch to ensure an up-to-date polygon.

The large central textbox displays the list of coordinate holes found during calculation. Here they can be manipulated and edited but should remain in an acceptable format to be loaded as a one-by-one file.

The bottom section of the window contains the parameter controls for hole calculation.

1. The **Digits** field controls how accurate you want the resulting hole coordinates to be.
2. The **Magnification** field controls the number of square divisions the polygon undergoes.
3. The **Empty Squares** field controls the number of squares that you are looking for.
4. The **Find Holes** button will run the hole finding algorithm and populate the central textbox with the list of holes found.

2.16 JKPatch



JKPatch is a tool that can be used to create and calculate patch polygons. Patches can be used as subset of a larger polygon that can be calculated at larger magnifications faster than the larger polygon.

The top of the window contains a scrollable list of all patches that have been created. Below this is are the controls for patch creation and removal as well as a label showing your currently selected patch.

1. The **Add Patch** button creates a new patch with default values. The new patch will be added to the list of patches and will be selected for editing.
2. The **Remove Patch** button removes the currently selected patch and selects the next patch in the list, if possible.

The patch editor is located in the middle of the window and contains four text fields to specify the polygon, digits, magnifications and empty squares of the patch.

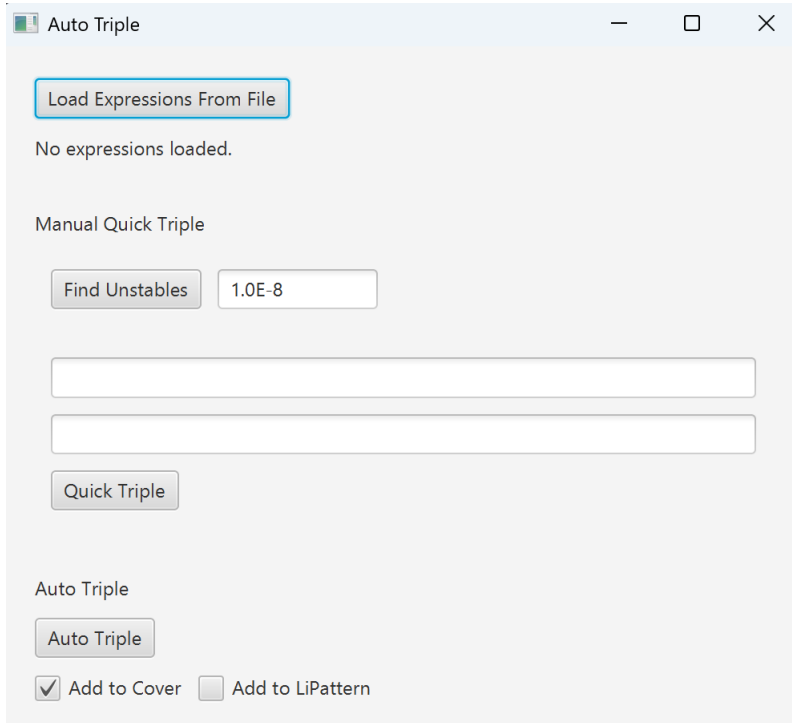
1. The **Polygon** field specifies the polygon of the patch. The polygon should be specified in the same format as the main polygon in Cover.
2. The **Digits** field controls how accurate you want the resulting hole coordinates to be.
3. The **Magnification** field controls the number of square divisions the polygon undergoes.
4. The **Empties** field controls the number of squares that you are looking for.

Below the patch editor is the patch controls.

1. The **Update Patch** button saves and verifies the current values to the selected patch. If the values are invalid, an error message will be displayed and the patch will not be updated.
2. The **Calculate Patch** button runs a cover calculation on the given patch. This operation is completely analogous to that ran when pressing the **Calculate** button in Cover.
3. The **Clear Visible Patches** button removes the visible patch polygons from the viewer. This does not remove the patches from the list of patches, but simply removes them from the viewer.

On a complete patch cover, the cover will be filled with gray.

2.17 JKAutoTriple



JKAutoTriple is a tool which provides a way to automatically find points that can be filled using triples and generate triples using a pattern heuristic. Note that this way of generating triples is not backed by any theorem but was developed through simple observation of other triples.

JKAutoTriple works by processing and comparing codes against a set of defined "classifying" expressions and generating unstables using a set of defined "resulting" expressions. Expressions are defined in a file that must be loaded before running "Quick Triple" or "Auto Triple".

The file is a simple text file with each line of the form: `CodeType, CodeLength, Locator, ClassifyngExpression, ResultingExpression`. The specific rules for the expressions can be found within comments in the file. George Tokarsky will be able to provide an up-to-date expressions file for reference.

The **Load Expressions From File** button loads the expressions file into JKAutoTriple.

The middle section of the window contains the Manual Quick Triple options.

1. The **Find Unstables** button uses the given ϵ value (1.0×10^{-8}) to

attempt to find points that can be filled using triples. That is, it checks for 2 unique stable code sequences within ϵ of the coordinate in the loaded OBO file.

2. The **Quick Triple** button uses the loaded expressions and two given stable code sequences to attempt to match and generate a triple.

The bottom section of the window contains the Auto Triple options.

1. The **Auto Triple** button uses the loaded expressions and the loaded OBO file to attempt to find points that can be filled using triples and generate triples for them.

Chapter 3

How to be a periodic path hunter

This chapter will assume that you have contacted George Tokarsky. He will give you coordinates so you can find the periodic 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 several 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 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. There are two approaches to finding more holes to fill. The first is to use the Cover tool and the second is to use the Hole Finder tool.

3.4.1 Using the Cover tool

- 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 use VaryL, MVL, LiLuMaxVary, and SuperLiLyVary with Tools 5C checked to do the steps automatically.

### 3.4.2 Using the Hole Finder tool

Step 1. Go to "JKFindHoles" in the top right of the tools menu

Step 2. Make sure the correct polygon is selected in the dropdown menu

- Step 3. Make sure your desired number of digits, magnification, and empty squares are set
- Step 4. Click on "Find Holes"
- Step 5. Use the top ribbon to copy the coordinates of the holes to a text file or load them as a one-by-one file
- Step 6. See JKFindHoles for more information on the hole finder tool

Note: The holes produced using the cover and hole finder tool given the same parameters will always be the same.

### 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