<https://docs.google.com/document/d/1V_sfzlnILqFFMS3mJLrN7s4lKMwGqr_fctLkJDIiACA/edit?usp=sharing>

Introduction:

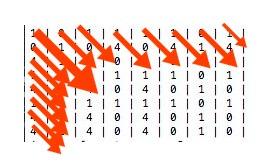
My java code for the Matrix Profile currently runs based on two files. Everything runs inside the child class, Distance Matrix which implements the class SquareGrid. Different versions of the code either run based on my test arrays or based on arrays created via .txt files full of the raw integer data points Tide Pool provides.

Given two array of equal length, the code creates a distance matrix. When two values being compared are exactly equal the MP will spit out a zero. We are looking for continuous strings of zeros, and total zeros per comparison.

Here is a simple example:



A Distance Matrix based on these values looks like this:



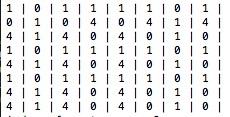
Explanation of the traversal:

The computation to make a Distance Matrix is rather simple, especially when all you care about is exact matches (zeros). My code makes the grid via horizontal traversals rather than going row by row. Making the grid row by row is simpler to code, but is only comparing one value from a set against every value from the other set. Building the Distance Matrix horizontally, allows for consecutive comparisons. So (looking at the example data points above). Comparing row by row would be to compare the first 3 in ArrayX agaisnt every value in ArrayY consecutively. Whereas, horizontal comparision allows both Arrays to increment what value is being compared simultaneously. So ArrayX compares its 3 to ArrayY’s 2, and then ArrayX compares its 2 against ArrayY’s 3, and then ArrayX compares its 3 against ArrayY’s 1 and so on.

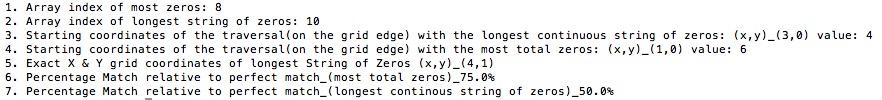
What I just described is starting in the top left corner and incrementing the x and y values one at a time, till I’m in the bottom left corner. The coordinate grid values are only positive and are formatted (y,x). Y being the Y axis(up/down) X being the X axis(left/right) So if the top left is (0,0) I go to (1,1),(2,2)...(7,7). That is one traversal. There are as many traversals as there are edge positions on the grid or ((sidelength\*2)-1). Next, I go to a new edge piece(lets say (0,1) the position one down of (0,0), and repeat till I hit the bottom of the grid/have no more values left to compare. The MatrixHelper method completes one traversal then sets up the starting position for the next one. Within this method are various information gathering tools. Different Arrays, and Matrix, count each zeros(match) in a traversal, the longest string of consecutive zeros, and its starting position. With each array index corresponding to a given starting position of a traversal

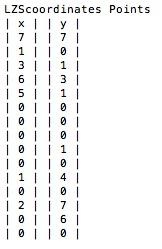
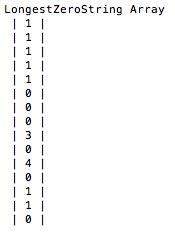
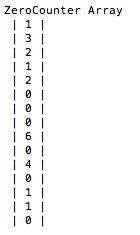
DistanceMatrix:

\*Remember the top left of the grid starts at (0,0)



Results Section:





Example of how to read the Data Structures:

Each index in these arrays and matrix are assigned to a specific traversal. So for index 0 (the traversal is from top left to bottom right) one zero was found, thus the longest string of zeros found was one long, and the coordinates of the longest string of zeros was at (7,7) the bottom right hand corner on the distance matrix.

Analysis of results:

So based on 3. We determine the traversal with the longest string of continuous zeros starts at the edge position (3,0) with length 4 and based on 5. the position where the string of zeros truly starts is coordinates (4,1) which represents ArrayX index 4 and ArrayY index 1, the 2nd comparison in the given traversal. And according to 7. This length of 4 represents 50% of what a theoretical perfect match would be, 8 zeros in a row starting at (0,0) ending at (7,7). According to 4. However that are more matches total in the traversal that starts at (1,0), at 6 matches, which according to 6. Represents a 75% match relative to the perfect match.

Problems and next steps:

The main concern I have right now is that all the math and code is designed around the grid being a square, so the two arrays of data points inputted must be equal length. I am not sure if appending a unique value such as zero, a value that will never be matched in our data, to the shorter array so the lengths matched would solve this, but it is a potential adaptation. It would however be computationally expensive because it requires a full copy and fill in the blank. Ed also mentioned gaps in the data that have to be accounted for, which again I believe would be best sorted with a unique value that could never be mistaken for a match. (Perhaps if the unique value is found) it doesn’t break/or add to an ongoing string of matches.

Another concern, given the immense amount of data tidepool is handling, is the computational complexity of the algorithm. The Distance Matrix profile is exact, has no risk of false negatives even with missing data, and provides a lot of information. The UCR Matrix Profile page suggests that a Matrix Profile can have constant time complexity. However, given the internal nested loops inherent to my code and the traversals it takes to build the code, I believe the time complexity is O(n^2), but I am not 100% sure. Using the java files DistanceMatrixUpdate2.java and SqaureGridUpdate2.java I did a mock speed test. Given two sets of 11,044 data points to compare , and create a Distance Matrix for, the code takes roughly 4.90 seconds, (I can’t call it any closer than ,1/10th of a second, given it was simply me along using my phone timer) on my 2011 MacBook Air to generate only the results section. At roughly 288 data points a day, that is 38.347222 days in 4.90secs. That equates to 46.63secs per one patient’s yearly data (assuming daily data donations). It would be interesting to see this done on different computers, because this test far exceeded by speed expectations. Some testruns got as low as 4.4sec. This trial didn’t require the program to print anything information with each match, loop break, or to print the matrix itself.

Bugs\*(9/12/18)- The coordinates for the longest zero string, found in the Matrix LZScoordinates Points, are sometimes one unit too high or low in their report. The errors tend to occur with strings of zeros that hit the bottom hand corner or begin at the start of the grid. (WILL INVESTIGATE)