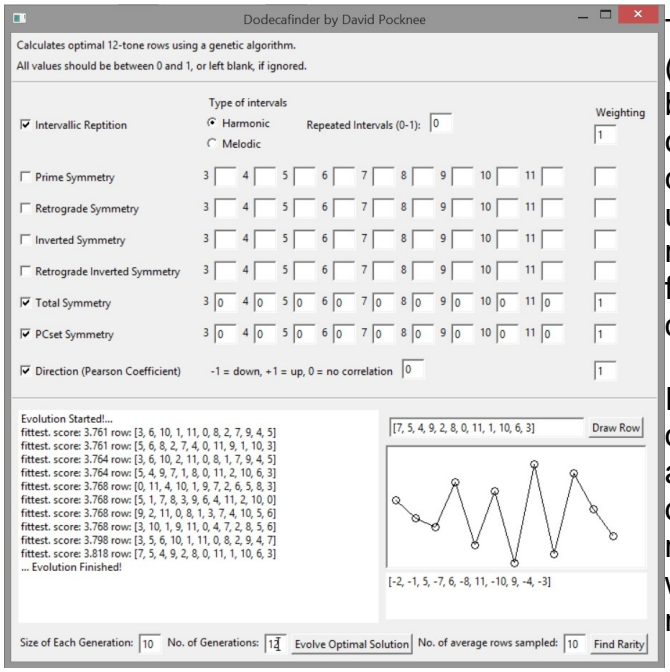
Dodecafinder

Software for finding optimal 12-tone rows using a genetic algorithm

David Pocknee



There are 479 million possible 12-tone rows (12! = 479,001,600). This makes doing a brute-force search for a row with specific characteristics time-consuming and computationally intensive. *Dodecafinder* uses a genetic algorithmic approach to reduce the time and computation needed to find rows which optimally match a given set of characteristics.

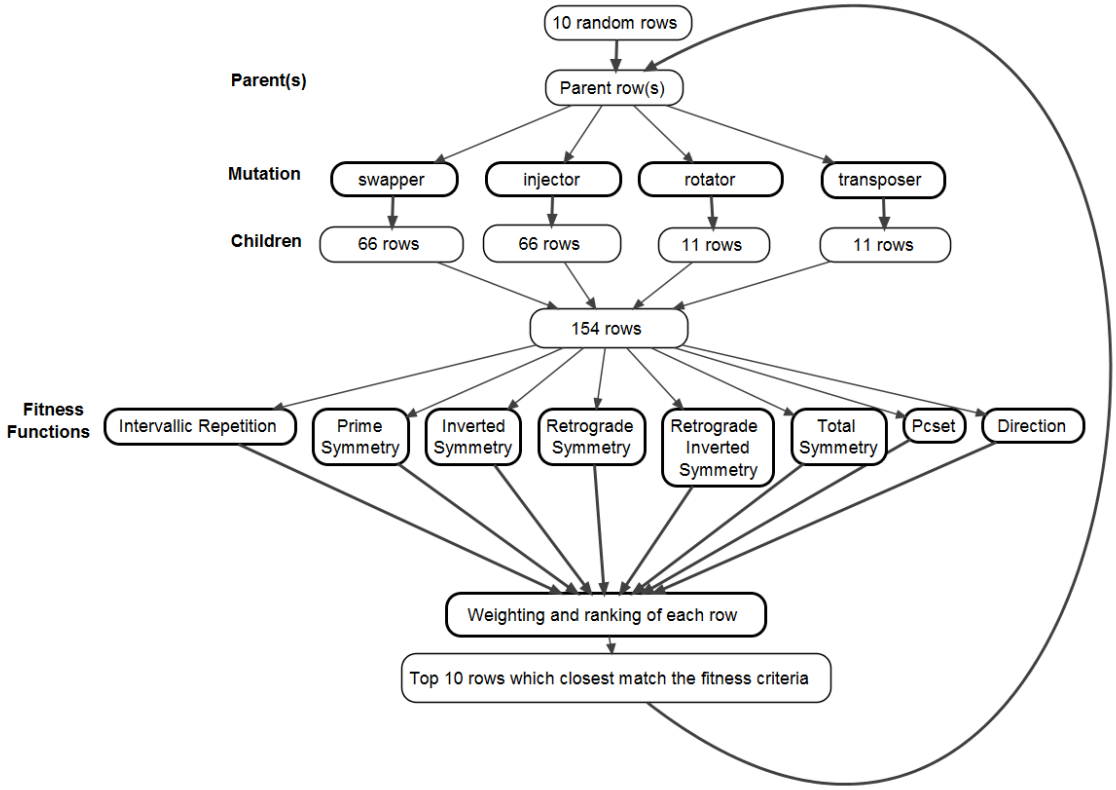
Imagine you want to find 12-tone row with only retrograde symmetries involving 3,4 and 9 notes, that moves in an upwards direction but which has minimal intervallic repetition. *Dodecafinder* can find a row which optimally matches this criteria in a matter of seconds.

Dodecafinder works by generating a set of random rows (parents), which are then "mutated" to create related rows. These "mutations" involve note-swapping, shifting of notes, rotation and transposition. The mutated rows and the original rows are then evaluated as to how closely they fit the target criteria for a row, and the top-scoring rows are then carried forward to become the "parents" for the next generation, which are then mutated. At the end of multiple iterations of this processes, the "survival of the fittest" set-up of the algorithm results in a row which is optimally suited to the starting criteria.

The diagram on the following page shows this process in action, starting with a set of 10 random parents, and carrying over 10 rows between each generation. Notice how each row only generates 154 mutated child rows, this means that, given a set of 10 parent rows and 12 generations, the computer only has to evaluate 154x10x12 = 18,480 rows, as opposed to 479 million - a reduction by a factor of 25,920.

Installing *Dodecafinder*

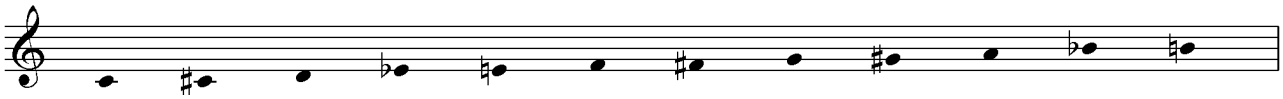
Dodecafinder is written in ruby, using the *fxruby* gem. It is distributed as a standalone windows .exe file, and as ruby source code. If using the ruby source code, *fxruby* will need to be installed prior to use (if ruby is already installed, fxruby can be installed by typing `gem install fxruby` in the terminal/command prompt etc.).



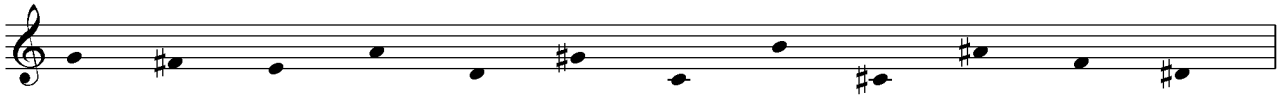
Using *Dodecafinder*

Dodecafinder checks rows according to 8 criteria:

Intervallic Repetition
This is the number of times an interval between two consecutive notes is repeated over the course of the row. A row such as:



would have a rating of 0, as all the notes here are a semitone apart, but a row such as:



would have a rating of 1, as all of the intervals are different.

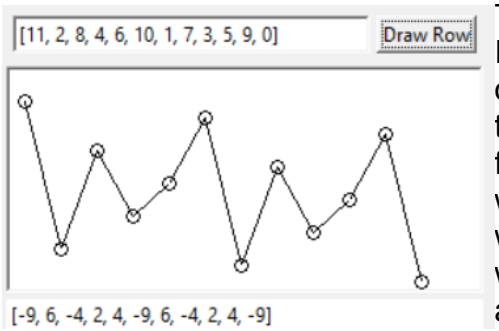
The intervals in a row can be represented as an array of vectors. For instance, the intervals of the first of the rows above can be represented as [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], and the second of the rows above can be represented as [-1, -2, 5, -7, 6, -8, 11, -10, 9, -4, -3].

There are two types of intervallic repetition: harmonic and melodic. In the *harmonic* setting, each interval is counted irrespective of the direction it moves in – so a minor 3rd up is seen as the same as a minor 3rd down (e.g +3 = -3). In the *melodic* setting a minor 3rd down would be seen as different to a minor 3rd up e.g. (+3 ≠ -3).

The “Repeated Intervals” box in the software should be filled with a number between 0 and

1; 0 meaning that the row should consist solely of repeated intervals, 1 meaning that there should be no repeated intervals.

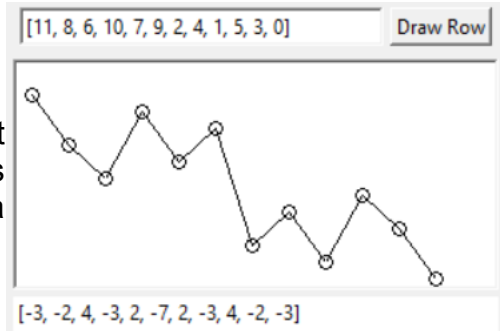
Prime Symmetry



This would normally be referred to as motivic repetition. In other words, it is a measure of how many times a set of intervals is repeated. Each of the numbered boxes in the software are used to specify the length of this motif, for instance, by typing a 1 in the "3" box, the software will try to find rows which have a large amount of motifs with a length of 3. Consider the example on the left, which has a high degree of prime symmetry around 3 and 4.

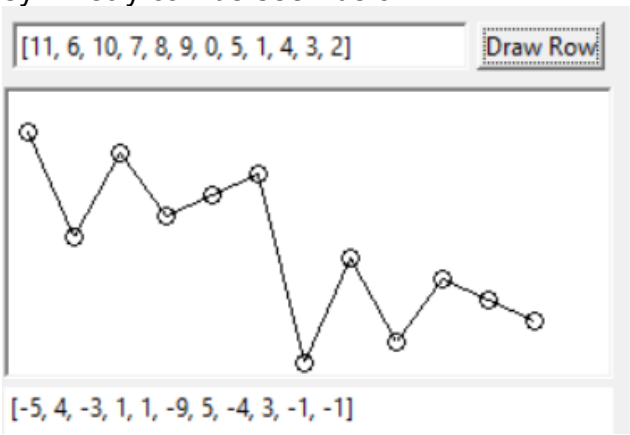
Retrograde Symmetry

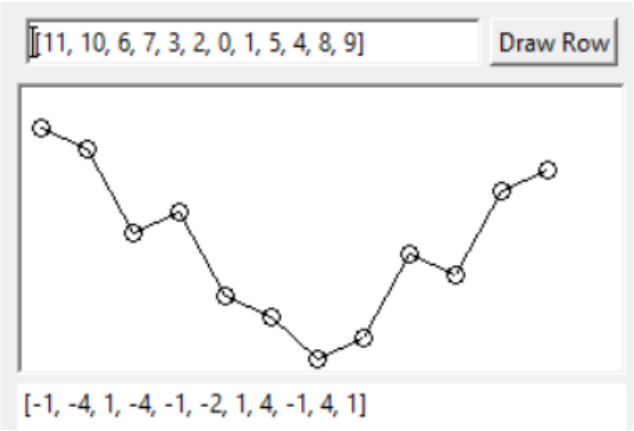
This uses a similar system to prime symmetry but instead tries to find rows in which the motif is retrograded. On the right is an example of a row with a high degree of retrograde symmetry.



Inverted Symmetry / Retrograde Inverted Symmetry

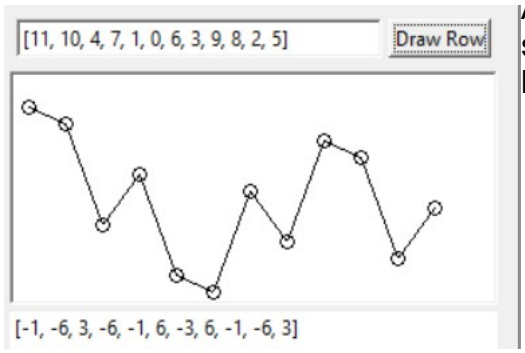
These are the logical extension of the types of symmetry seen above. An example of a row with a high level of inverted symmetry, and row with a high level of retrograde symmetry can be seen below.





Total Symmetry

If you are interested in rows which simply have high levels of symmetry, irrespective of whether they are prime, retrograde, inverted etc. then the *Total Symmetry* option is recommended. In any row, one type of symmetry will prevent another type from occurring, so *Total Symmetry* is designed to be used when a mix of symmetry types are being looked for but these do not need to be specified. See this option as the OR as opposed to the AND created by combining the different types of symmetry manually. An example of a row with a high level of total symmetry can be seen on the left.



PCset Symmetry
This functions in a similar way to the symmetry variables above but instead it looks for repeated pitch class sets in the row. 1 = high level of repeated pcsets, 0 = no repeated pcsets. Similarly, the size of pcset used can be specified.

Direction
This allows you to specify the direction and/or amount of correlation in a row. 0 = no overall direction in the row, +1 = the row goes upwards, -1 = the row goes downwards. This is calculated using the Pearson Correlation Coefficient.

General Use Of The Program

To generate a row which optimally fits a particular set of criteria, tick the check box to the left of the name of the criteria you wish to use. You must then also fill in a weighting on the far right of the interface. This specifies how much importance this particular category should have when evaluating each row. It should be a number between 0 and 1.

All boxes in the top half of the software interface take a number between 0 and 1, this can be a float. Leaving a box blank means that it will not be taken into account when assessing rows. If you are using a particular criteria, the weighting box for this criteria must always be filled. For the numbered symmetry boxes, placing a 0 in the box indicates that the software will search for rows in which that symmetry does not occur, whereas putting a 1 in the box will cause a search for rows with a high prevalence of that type of symmetry.

Once you are happy with the criteria you want to use and have chosen your preferred variables, you must fill the boxes titled “Size of each generation” and “No. of generations” at the bottom left of the interface.

“Size of each generation” is the number of random starting parents as well as the number of rows carried over between each generation. The default here is 10, but if the algorithm is giving poor results, I recommend increasing this.

“No. of generations” is the amount of generations the algorithm cycles through before stopping. I recommend increasing this if you are increasing the size of each generation. The default is 12.

Once these variables are specified you can then click “Evolve Optimal Solution”. The algorithm will then run, and the current process can be seen in the command prompt window which will open along with the main GUI. In this window, the current generation and its “fittest” mutations are shown. This output may be useful for debugging. The output shows:

```
[overall score [array of scores for each individual criteria selected] original row]
```

A higher score indicates a closer match to the criteria selected.

Once the algorithm has finished, the fittest mutations of the final generation are shown in the window in the left of the main interface. This output shows whether the row was a SOLUTION, meaning it exactly meets all the criteria you laid down, or is simply “fittest”, meaning it is one of the fittest of that generation but is not an exact match for the criteria given. The output also shows the score the row was given, in relation to the criteria you

specified, and the actual row as an array of numbers between 0 and 11.

This array can then be copied over to the box on the right of the interface next to the “Draw Row” button. When that box is filled with an array of a 12-tone row (scaled between 0 and 11) and the button is clicked, it produces a visual representation of the row, along with the vectors of the row underneath.

Find Rarity

This button, in the bottom right corner is an experimental algorithm for finding “interesting” rows by looking for rarity in the combinatorial space. The algorithm randomly samples from all possible twelve tone rows, analyses them using any criteria selected in the top of the interface, averages these values, and then tries to evolve a row which is as far away from the average of the combinatorial space as possible. I suggest setting the “No. of average rows sampled” variable relatively high for maximum success (+ 100).