Pixel Regression GP

Evolutionary Algorithms Final Project
Rakhim Davletkaliyev

IDEA

Given a picture (BMP format, 32-bit per pixel), the program scans it pixel-by-pixel and stores this information in an array. Then, it creates a population, each individual of which is random pixels array of the same size. The program tries to evolve a random pixel picture into the given picture by applying mutation to particular pixels and crossover to whole array.

Program uses SDL (Simple Direct Layer) C++ library to work with graphics.

PIXELS

Each pixel is represented by 32-bit unsigned integer (Uint32 data type from ISO C90).

[RRRRRRRR][GGGGGGGG][BBBBBBBB][AAAAAAAA]

First 8 bits represent red portion of red color, next 8 bits – green color, next 8 bits – blue color and last 8 bits are alpha (transparency). Uint32 ranges from 0 to 4,294,967,295.

INDIVIDUAL

Each individual is a structure with a 2D-array of pixels, Uint32 variable representing fitness value and a 2D-array of booleans of the same size as pixel map that represents fixed points: pixels, that are regressed successfully to needed condition and need not to be mutated anymore. This array of fixed pixels is used for filling points around and speeding up the process.

CHROMOSOME

Chromosome is an array of pixels, by default the size is 640x480 = 307,200 pixels. The total size of chromosome is 307,200x32 bits per pixel = 9,830,400 bits ~ 9.3 Mb.

FITNESS

Fitness is total difference between pixels of an individual and ideal for every pixel. The ideal fitness is 0 – no difference with ideal. Since pixels represented by integers, we can calculate the difference pretty easily.

The table below shows how fitness is calculated. If pixel's value is twice as big or twice as small as ideal (100% difference), then 8 is added to fitness, if the difference is only 12% (0.88 or 1.12 of ideal value), then 1 is added to fitness, and so on.

128	64	32	16	8	4	2	1	0	1	2	4	8	16	32	64	128
<	.13	.25	.38	.5	.63	.75	.88	**1**	.12	1.25	1.38	2	2.13	2.25	2.38	>

MUTATION

Mutation starts with creation of a copy of a given individual's chromosome. Then, every pixel is compared with ideal's pixel on that position, similarly to fitness calculation. Pixel is mutated with some rate, which depends on the difference.

Mutated copy is then evaluated and it replaces the individual's chromosome only if its fitness is better.

KNOWN POINTS

The process is fairly slow, but we can speed it up by setting some percentage of points as “known”, so that at each mutation the program knows few pixels for sure. Percentage is set in allinc.h header file, KNOWN constant.

FILLING POINTS AROUND

Between crossover and mutation, program checks if any fixed points are present, and if they are, then 8 adjacent points around it are set to the same value (if they aren't fixed themselves).

Neighbor pixels are almost the same most of the time, but not exactly the same. The process of filling points around makes just a good guess, which then helps mutation operator to make the area of search smaller.

CROSSOVER

There are three types of crossover. Type is selected at random for each crossover case, each type has the same probability (1/3) to be used. CutPointCrossover generates two random coordinates and puts everything below and to the right of these coordinates from parent 2 to parent 1. OddHorizontalCrossover puts every odd horizontal line from parent 2 to parent 1. OddVerticalCrossover puts every odd vertical line from parent 2 to parent 1.