Instituto Politécnico Nacional

ESC Superior de Cómputo

"EVOLUTIONARY COMPUTING" 1-FRACTALS AND THE NATURE EQUATION

Jorge Luis Rosas Trigueros

Saldaña Aguilar Gabriela

111/16/16

THEORETICAL FRAMEWORK

Evolutionary algorithms have been shown to be successful for a wide range of optimization problems. While these randomized search heuristics work well for many optimization problems in practice, a satisfying and rigorous mathematical understanding of their performance is an important challenge in the area of genetic and evolutionary computing. It has been proven for various combinatorial optimization problems that they can be solved by evolutionary algorithms in reasonable time using a suitable representation together with mutation operators adjusted to the given problem. The representations used in these papers are different from the general encodings working with binary strings as considered earlier in theoretical works on the runtime behavior of evolutionary algorithms. the chosen representations reflect some properties of partial solutions of the problem at hand that allow to obtain solutions that can be extended to optimal ones for the considered problem.

Dynamic programming is a well-known algorithmic technique that helps to tackle a wide range of problems. A general framework for dynamic programming has been considered by e. g. Woeginger and Kogan . The technique allows the design of efficient algorithms, that solve the problem at hand to optimality, by extending partial solutions to optimal ones.

Framework for Evolutionary Algorithms An evolutionary algorithm consists of different generic modules, which have to be made precise by the user to best fit to the problem. Experimental practice, but also some theoretical work, demonstrate that the right choice of representation, variation operators, and selection method is crucial for the success of such algorithms. We assume that the problem to be solved is given by a multi-objective function g that has to be optimized. We consider simple evolutionary algorithms that consist of the following components. The algorithm (see Algorithm 2) starts with an initial population P0.

During the optimization the evolutionary algorithm uses a selection operator sel(·) and a mutation operator mut(·) to create new individuals. The d-dimensional fitness function together with a partial order par on R d induce a partial order dom on the phenotype space, which guides the search. After the termination of the EA, an output function out(·) is utilized to map the individuals in the last population to search points from the original search space.

FRACTALS IN NATURE

Fractals are patterns formed from chaotic equations and contain self-similar patterns of complexity increasing with magnification. If you divide a fractal pattern into parts you get a nearly identical reduced-size copy of the whole.

The mathematical beauty of fractals is that infinite complexity is formed with relatively simple equations. By iterating or repeating fractal-generating equations many times, random outputs create beautiful patterns that are unique, yet recognizable.[4]

In theory, one can argue that everything existent on this world is a fractal:

- the branching of tracheal tubes,
- the leaves in trees,
- the veins in a hand,
- water swirling and twisting out of a tap,
- a puffy cumulus cloud,
- tiny oxygene molecule, or the DNA molecule,
- the stock market

This idea of being detailed relates to another feature that can be understood without mathematical background: Having a fractional or fractal dimension greater than its topological dimension, for instance, refers to how a fractal scales compared to how geometric shapes are usually perceived. A regular line, for instance, is conventionally understood to be 1-dimensional; if such a curve is divided into pieces each 1/3 the length of the original, there are always 3 equal pieces. In contrast, consider the Koch snowflake. It is also 1-dimensional for the same reason as the ordinary line, but it has, in addition, a fractal dimension greater than 1 because of how its detail can be measured. The fractal curve divided into parts 1/3 the length of the original line becomes 4 pieces rearranged to repeat the original detail, and this unusual relationship is the basis of its fractal dimension.[3]

Characteristics

- Self-similarity, which may be manifested as:[1]
  - Exact self-similarity: identical at all scales; e.g. Koch snowflake
  - Quasi self-similarity: approximates the same pattern at different scales; may contain small copies of the entire fractal in distorted and degenerate forms; e.g., the Mandelbrot set's satellites are approximations of the entire set, but not exact copies.
  - Statistical self-similarity: repeats a pattern stochastically so numerical or statistical measures are preserved across scales; e.g., randomly generated fractals; the well-known example of the coastline of Britain, for which one would not expect to find a segment scaled and repeated as neatly as the repeated unit that defines, for example, the Koch snowflak
  - Qualitative self-similarity: as in a time series
  - Multifractal scaling: characterized by more than one fractal dimension or scaling rule
- Fine or detailed structure at arbitrarily small scales. A consequence of this structure is fractals may have emergent properties (related to the next criterion in this list).
- Irregularity locally and globally that is not easily described in traditional Euclidean geometric language. For images of fractal patterns, this has been expressed by phrases such as "smoothly piling up surfaces" and "swirls upon swirls".
- Simple and "perhaps recursive" definitions see Common techniques for generating fractals

General Equation of Nature

S -> e * S *

An equation based on Mathematical linguistics and fractals, with everything that can represent the structure of multiple elements of nature, including trees, clouds, stars, mountains, snails and rivers and which we propose as one of the fundamental equations of nature

EQUIPMENT AND MATERIAL

SOFTWARE: The program is made in Python

BODY

We were told to write down the code for making a landscape using the formula mentioned above(see e. g. [1]).

First I'm going to talk about the code for making a tree.

Trees are represented by the family of one e() and several S(), where e() is the function that draws a line and S() is the recursive call, so the number of branches we want is given by the number of S()calls.[2][5]

#S->e*S* S->eSS Two Branches tree. S->eSSS.... N Branches Tree

Tree:

```
def arbol_tupido(w,x0,y0,l,an,color):#S() x,y,alto,angulo entre as grande mas espacio entre ramas
    ind=1;
    if (l > ind ):
        coseno = math.cos(an/57.29578)#coseno de el angulo
        seno = math.sin(an/57.29578)#seno de el angulo
        x1=x0-(l*coseno);
        y1=y0-(l*seno);
        t1=l/1.5
        w.create_line(x0,y0,x1,y1,fill=color)#e()
        arbol_tupido(w,x1,y1,t1,an-47,color)#The number next to an is how much we want the tree to be close
        arbol_tupido(w,x1,y1,t1,an+0,color)
        arbol_tupido(w,x1,y1,t1,an+47,color)
    return
```

Árbol generado con w_1=47, w_2=0, w_3=-47

Árbol generado con w_1=5, w_2=0, w_3=-5

Árbol generado con w_1=85, w_2=0, w_3=-85

Como se puede ver aun jugando con un solo parámetro los resultado pueden ser disimbolos, en un caso se pueden tener árboles "normales" (w ### 47), tipo cipreses (w ### 5) o araucaria (w ### 85).

Por otro lado si w es igual a 47 o 46 o 48 los árboles se parecen entre si.

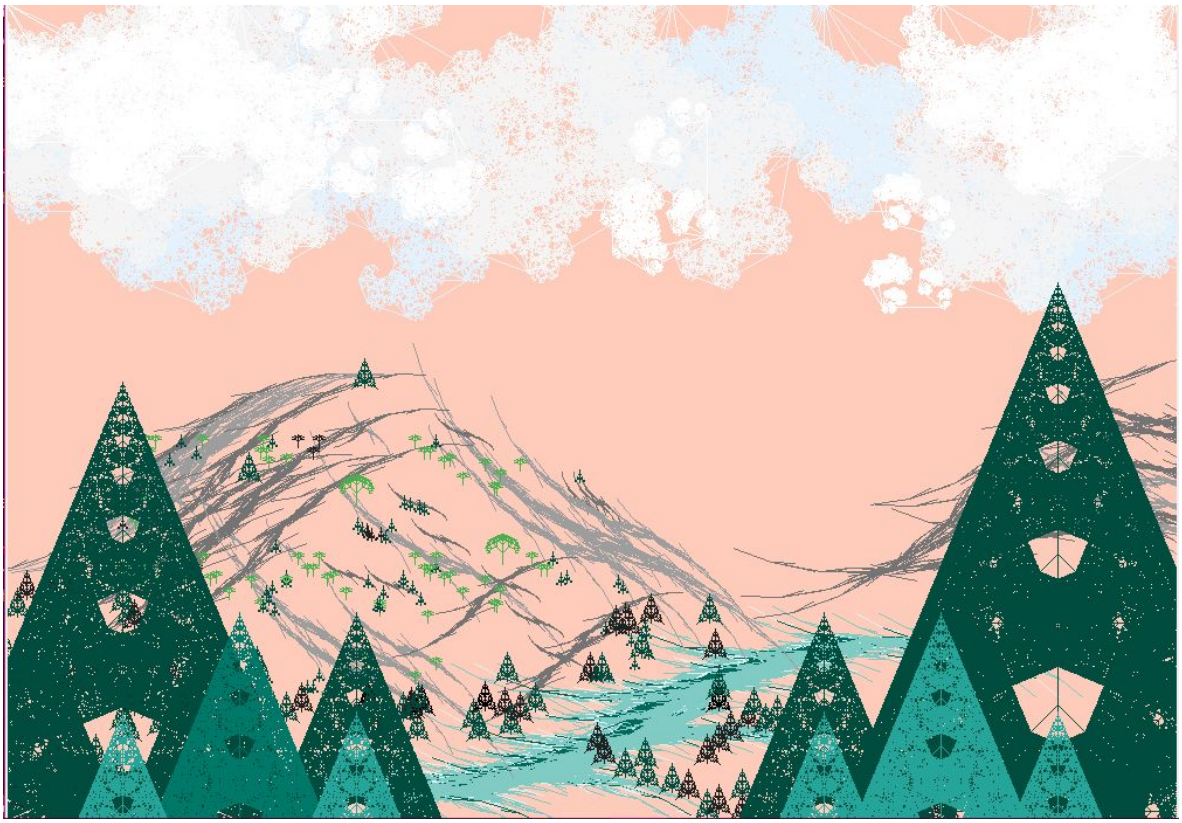
Por lo que se tiene un espacio de ángulos donde el comportamiento de los árboles es similar. O sea que se puede encontrar un espacio de valores para los cuales los árboles que se generar son parecidos o pertenecen a la misma familia. A un espacio de este tipo se le llama *espacio de caos*.

Clouds :

```
def nube_amplia(w,x0,y0,l,an,ind,color):#S()
   if ind > 0 :
      coseno = math.cos(an/57.29578)#coseno de el angulo
      seno = math.sin(an/57.29578)#seno de el angulo
      x1=x0-(l*coseno);
      y1=y0-(l*seno);
      t1=l/1.7
      t2=l/1.7
      t3=l/1.5
      w.create_line(x0,y0,x1,y1,fill=color)#e()
      nube_amplia(w,x0,y0,t1,an +30,ind-1,color)
      nube_amplia(w,x1,y1,t2,an + 50, ind-1,color)
      nube_amplia(w,x1,y1,t3,an + 100, ind-1,color)
   return
```

Mountains and Rivers:

```
def river(w,x0,y0,l,an,ind,color):
    if ind >0:
        coseno = math.cos(an/57.29578)#coseno de el angulo
        seno = math.sin(an/57.29578)#seno de el angulo
        x1=x0-(l*coseno);
        y1=y0-(l*seno);
        t1=l/1.2
        t2=l/2.0
        t3=l/1.1
        w.create_line(x0,y0,x1,y1,fill=color)#e()
        river(w,x1,y1,t1,an + 10,ind-1,color)
        river(w,x1,y1,t2,an + 172, ind-1,color)
        river(w,x1,y1,t3,an + 180, ind-1,color)
    return
```



F1: Landscape made with the functions mentioned above.

CONCLUSION

This activity was for me so much creative than the other ones, because you can actually see what you are making and you create your result up to your imagination. The way the equation works is pretty unique, because you can make a lot of shapes changing some parameters and it still has the same recursion calls for the S->e*S*, it is wonderful.

BIBLIOGRAPHY

[1] Falconer, Kenneth (2003). . John Wiley & Sons. xxv. ISBN 0-470-84862-6.

[2] http://www.fgalindosoria.com/ecuaciondelanaturaleza/una_ecuacion_naturaleza/ec_natu.pdf

[3] https://en.wikipedia.org/wiki/Fractal

[4] https://www.wired.com/2010/09/fractal-patterns-in-nature/

[5] http://www.fgalindosoria.com/ecuaciondelanaturaleza/index.htm