Instituto Politécnico Nacional

ESC Superior de Cómputo

“EVOLUTIONARY COMPUTING” P5: ACKLEY AND RASTRIGIN SWARM OPTIMIZATION

Jorge Luis Rosas Trigueros

Saldaña Aguilar Gabriela

25/10/16

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.

The first thing we need to ask ourselves is ¿What I´m going to evaluate with my genetic algorithm?

R= We need to evaluate how good is a solution for the given problem, it means that we will have a bunch of solutions and we want the better one. In short Chromosome=Possible solution.

SWARM INTELLIGENCE

Swarm intelligence: two or more Individuals Independently collect information is processed through That social interaction and provides a solution to a problem cognitive That is not available to Individuals single.

This phenomeno was first observed in animals, it is seen that they can achive more easly their goals and protect themselves against predators: such as a flock of birds, ant colonies to shoal of fishes. We look for an integration of the animals's SI work on groups of humans.

In humans there is not only interindividual variation in cognitive abilities but in some cases there are high performers Individuals that are by any standard "geniouses".It is true also that not all groups of animals or humans are SI but there is a potential for it.

SI is used mainly for making decitions or forecasting in modern society.Some times SI works when Individuals are concerned of what the context is and what we want to achieve in this case if we ask the Individuals to achieve a task: such as "to tell ¿how many marbles are there in a jar "they will do it imprecisely but their guess can result in a close approximation in other cases SI may fail when there is an excel difference of knowledge between individuals, an example could be to ask the individuals" estimate how many times a coin needs to be tossed for the probability that the coin shows heads on all occasions " their guess is clearly devided for the Individuals who knows probability and the ones that does not.

There is also the notion of diversity in a group, in some studies Increased diversity was not beneficial to the group whereas in others to clear permormance advantage was found, this results are partly owing to the fact That Differences in perspective can create communications barriers, and some of them depends on the respect for each other.

What is clear is that diversity of perspective is good for SI, but in some quantities, they can not be so close to each other because they would act as a single individual but it can not be so spaced because of the communication barriers .

We can conclude that SI performs well if there is diversity of opinion, independence of opinion, an Initiative for truthful reporting, and if Individuals are only influenced by imprecision like the marble's problem and not by a systematic bias: such as the probability problem.However despite the fact that there are some criteria for effective use of SI, there is lack of a baseline for this behavior.

PARTICLE SWARM OPTIMIZATION

We are going to have a bunch of particles that explore the search space in this way we will have a variety of behaviors because they are distant from each other far enough to find new solutions instead of having only one particle following one path. Each particle has a velocity, position and direction in the search space.

Let Pi be the best known position of particle i and let LIDER be the best known position of the swarm:

For each particle we need to initialize their velocities , Fip,Fig and W, and values of particle Xi the best position of this group would be the “LIDER” so the algorithm goes like this:

```
For each particle i
  For each dimention
        Pick a random number for rg,rp and velocities
        Update the velocity: Vid=W*Vid+Fiprp(Pid-Xid) + Figrg(LIDER - Xid)
Update Xid=Xid+Vid
If f(Xid) < f(Pid)
        Pid=Xid
If f(Pid)<f(LIDER)
        g=Pid
```

This would tell us the behavior of the swarm, depending in what values we get in updating the velocity they could chase their own inertia or their own experience or they can follow the lider.

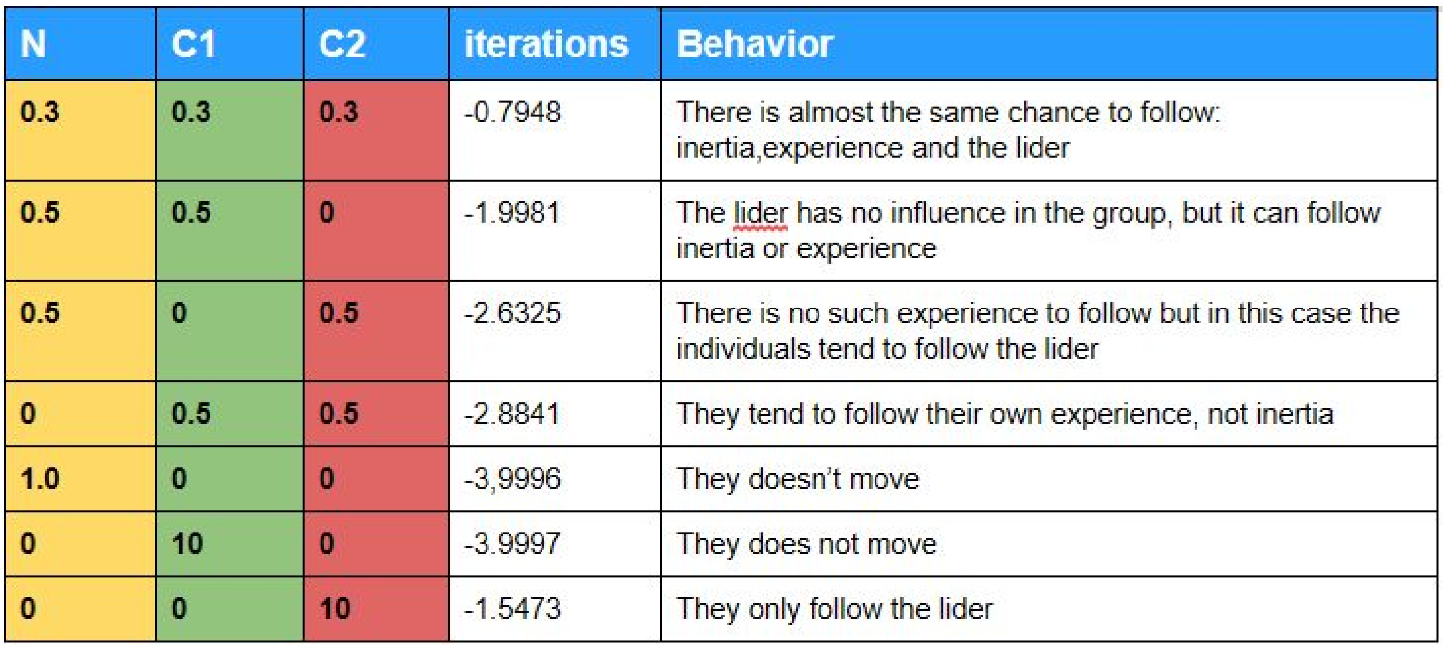
EQUIPMENT AND MATERIAL.

SOFTWARE: The program is made in Python

BODY

We were told to write down the problems mentioned above and fill the table above:

(see e. g. [1]).



N	C1	C2	iterations	Behavior
0.3	0.3	0.3	-0.7948	There is almost the same chance to follow: inertia,experience and the lider
0.5	0.5	0	-1.9981	The lider has no influence in the group, but it can follow inertia or experience
0.5	0	0.5	-2.6325	There is no such experience to follow but in this case the individuals tend to follow the lider
0	0.5	0.5	-2.8841	They tend to follow their own experience, not inertia
1.0	0	0	-3,9996	They doesn't move
0	10	0	-3.9997	They does not move
0	0	10	-1.5473	They only follow the lider

F1: BEHAVIOR TABLE

The next thing to do was to change the code for an optimization using Ackley and Rastrigin functions using two dimentions for the particle swarm optimization.

RASTRIGIN

```
def f(x,y):
        a=20.0
        b=0.2
        c=(2.0*math.pi)
        return ((-a)*((-b)*((((x**2)*(0.5)) + ((y**2) *(0.5)))**(0.5))))-(math.exp(((math.cos(c+x))*(0.5))
+((math.cos(c+y))*(0.5)))) + a + math.exp(1.0)
```

ACKLEY

```
def f(x,y):
        return ((20.0) + ((x**2)- (10.0*(math.cos((2.0*math.pi)*x)))) + ((y**2) - (10.0*(math.cos((2.0*math.pi)*
y)))))
```

CODE

```
#Ackley example:

from Tkinter import *
from numpy import *
#import random
lower_limit=-20
upper_limit=20
n_particles=10
n_dimensions=2
# Initialize the particle positions and their velocities
X = lower_limit + (upper_limit - lower_limit) * random.rand(n_particles, n_dimensions)
#print "X:",X
assert X.shape == (n_particles, n_dimensions)
V = zeros(X.shape)
assert V.shape == (n_particles,n_dimensions)
#print "V:",V
X_lbest=1*X
assert X_lbest.shape == (n_particles, n_dimensions)
#print "X_lbest",X_lbest
X_gbest= 1*X_lbest[0]
X_gbest.shape == ( 1,n_dimensions)
#print "X_gbest",X_gbest
#evaluamos por parejas
def f(x,y):
	return ((20.0) + ((x**2)- (10.0*(math.cos((2.0*math.pi)*x)))) + ((y**2) - (10.0*(math.cos((2.0*math.pi)*
y)))))
for I in range(0, n_particles):
	if f(X_lbest[I][0],X_lbest[I][1])<f(X_gbest[0],X_gbest[1]):
	X_gbest[0]=1*X_lbest[I][0]
	X_gbest[1]=1*X_lbest[I][1]
#print "X_gbest desp",X_gbest
def iteracion():
	global X,X_lbest,X_gbest,V
	weight=0.3
	C1=0.3
	C2=0.3
	print "Best particle in:",X_gbest[0],X_gbest[1]," gbest: ",f(X_gbest[0],X_gbest[1])
	# Update the particle velocity and position
	for I in range(0, n_particles):
	R1 = random.rand()#rp
	R2 = random.rand()#rg
	#updating velocity
	V[I][0] = (weight*V[I][0]+ C1*R1*(X_lbest[I][0] - X[I][0])+ C2*R2*(X_gbest[0] - X[I][0]))
	V[I][1] = (weight*V[I][1]+ C1*R1*(X_lbest[I][1] - X[I][1]) + C2*R2*(X_gbest[1] - X[I][1]))
	#updating best particle(solution)
	X[I][0] = X[I][0] + V[I][0]
```

```
    X[I][1] = X[I][1] + V[I][1]
    #updating lider and best position
    if f(X[I][0],X[I][1])<f(X_lbest[I][0],X_lbest[I][1]):
    X_lbest[I][0]=1*X[I][0]#Pid
    X_lbest[I][1]=1*X[I][1]#Pid
    if f(X_lbest[I][0],X_lbest[I][1])<f(X_gbest[0],X_gbest[1]):
    X_gbest[0]=1*X_lbest[I][0]#g
    X_gbest[1]=1*X_lbest[I][1]#g

for K in range(10):
    iteracion()
```

CONCLUSION

I liked it the way we took the class in the lab like if it was free style, but with the help of the professor, also it is good to explain what will be the next topic to cover in the lab so the students can find it out and investigate a little bit before entering the class because it could be confusing if it is our first time coding evolutionary algorithms.

BIBLIOGRAPHY

Evolutionary Computing class