Project 4

Dr. Donald Davendra
CS471 - Optimization

May 8, 2019

1 Introduction

Project 4 introduces three new meta-heuristics. These are the Particle Swarm Optimization (PSO), Firefly Algorithm (FA) and Harmony Search Algorithm (HS).

This lab requires you to code these three algorithms in a common framework and conduct experiments on the selected problems. The algorithms are described in the following sections.

2 Particle Swarm Optimization

Inspired by the flocking and schooling patterns of birds and fish, Particle Swarm Optimization (PSO) was invented by Russell Eberhart and James Kennedy in 1995. Originally, these two started out developing computer software simulations of birds flocking around food sources, then later realized how well their algorithms worked on optimization problems.

Particle Swarm Optimization might sound complicated, but it's really a very simple algorithm. Over a number of iterations, a group of variables have their values adjusted closer to the member whose value is closest to the target at any given moment. It's an algorithm that's simple and easy to implement.

The algorithm keeps track of three global variables:

- Target value or condition
- Global best (gBest) value indicating which particle's data is currently closest to the Target
- Stopping value indicating when the algorithm should stop if the Target isn't found

Each particle consists of:

- Data representing a possible solution
- A Velocity value indicating how much the Data can be changed
- A personal best (pBest) value indicating the closest the particle's Data has ever come to the Target

The particles' data could be anything. In the flocking birds example above, the data would be the X, Y, Z coordinates of each bird. The individual coordinates of each bird would try to move closer to the coordinates of the bird which is closer to the food's coordinates ($gBest$). If the data is a pattern or sequence, then individual pieces of the data would be manipulated until the pattern matches the target pattern.

The velocity value is calculated according to how far an individual's data is from the target. The further it is, the larger the velocity value. In the birds example, the individuals furthest from the food would make an effort to keep up with the others by flying faster toward the gBest bird. If the data is a pattern or sequence, the velocity would describe how different the pattern is from the target, and thus, how much it needs to be changed to match the target.

Each particle's $pBest$ value only indicates the closest the data has ever come to the target since the algorithm started.

The $gBest$ value only changes when any particle's $pBest$ value comes closer to the target than $gBest$. Through each iteration of the algorithm, $gBest$ gradually moves closer and closer to the target until one of the particles reaches the target.

It's also common to see PSO algorithms using population topologies, or "neighborhoods", which can be smaller, localized subsets of the global best value. These neighborhoods can involve two or more particles which are predetermined to act together, or subsets of the search space that particles happen into during testing. The use of neighborhoods often help the algorithm to avoid getting stuck in local minima.

The general outline of the PSO algorithm is given in the following pseudocode.

input : *Iterations*: maximum number of iterations
Particles: number of particles p_i
gBest: the best solution in the population
pBest: the best solution found by specific particle
Bounds: Problem bounds (U - upper bound, L - lower bound)
output: *gBest*: best particle found

for i = *1,...,Particles* **do**
  /* generate particles randomly */
  p_i = L+rand[0,1](U-L)
  /* calculate particles velocity */
  v_i = random(0, 0.5 $\cdot(U - L)$)
  /* calculate particles fitness */
  u_i = f(p_i)
  /* set *pBest* for each particle */
  $pBest_i = u_i$
end for
/* set gBest from all particles */
$gBest$ = min($pBest$)
for t = *1,...,Iterations* **do**
  for j = *1,...,Particles* **do**
    /* calculate new velocity v_j of particle p_j */
    ${v_j}^{(t+1)} = {v_j}^{(t)} + c_1 \cdot rand \cdot \left({pBest^{(t)}}_j - {p_j}^{(t)}\right) + c_2 \cdot rand \cdot \left(gBest^{(t)} - {p_j}^{(t)}\right)$
    /* update particle p_j */
    ${p_j}^{(t+1)} = {p_j}^{(t)} + {v_j}^{(t+1)}$
    /* calculate particles fitness u_j */
    u_j = f(p_j)
    /* check if the particle velocity has improved */
    if $u_j < pBest_j$ **then**
      /* update *pBest* of particle */
      $pBest_j = u_j$
    end if
    /* check if *gBest* has improved */
    if $pBest_j < gBest$ **then**
      /* update *gBest* */
      $gBest = pBest_j$
    end if
  end for
end for

Algorithm 1: Particle Swarm Optimization

3 Firefly Algorithm

Firefly algorithm (FA) is a new swarm intelligence algorithm developed by Yang in 2010. It is inspired by the social behavior of fireflies based on the flashing and attraction characteristics of fireflies. In the past five years, the research of FA has attracted much attention. Different versions of FA has been designed to solve bench- mark or real-world optimization problem

In the FA, the fitness function for a given problem is associated with the light intensity. The brighter the firefly is, the better the firefly is. That means a brighter firefly has a better fitness value. The search process of FA depends on the attractions between fireflies. Based on these attractions, a firefly tends to move other brighter fireflies. If a firefly is brighter than another one, the brighter firefly will not conduct any search. When the current firefly is brighter than another one, a local search operation is conducted on the current one to provide more chances of finding more accurate solutions.

3.1 Description

As mentioned before, the FA mimics the behavior of the social behavior of the flashing characteristics of fireflies. To simply the behavior of fireflies and construct the search mode of FA, three rules are used as follows:

1. All fireflies are unisex so that one firefly is attracted to other fireflies regardless of their sex;

2. Attractiveness is proportional to their brightness. For any two fireflies, the less bright one is attracted by the brighter one. The attractiveness is proportional to the brightness and they both decrease as their distance increases. If no one is brighter than a particular firefly, it moves randomly;

3. The brightness or light intensity of a firefly is affected or determined by the land- scape of the objective function to be optimized. For a minimization problem, the brightness can be proportional to the objective function. It means that the brighter firefly has smaller objective function value.

As light intensity and thus attractiveness decreases as the distance from the source increases, the variations of light intensity and attractiveness should be monotonically decreasing functions. This can be approximated by the following Equation (1).

$$I\left(r\right) = I_0 e^{-\gamma r^2} \tag{1}$$

where I is the light intensity, I_0 is the original light intensity, and c is the light absorption coefficient. The attractiveness of a firefly is proportional to the light intensity. The attractiveness β of a firefly can be defined by Equation (2):

$$\beta\left(r\right) = \beta_0 e^{-\gamma r^2} \tag{2}$$

where β_0 is a constant and presents the attractiveness at $r = 0$. The distance between $r_{i,j}$ between any two fireflies i and j can be calculated by Equation (3):

$$r_{i,j} = \|X_i - X_j\| = \sqrt{\sum_{d=1}^{D} \left(x_{i,d} - x_{j,d}\right)^2} \tag{3}$$

where D is the dimensional size of the given problem. Based on the above definitions, the movement of this attraction is defined by Equation (4):

$$x_{i,d}^{(t+1)} = x_{i,d}^{(t)} + \beta_0 \cdot e^{-\gamma r_{i,j}^2} \cdot \left(x_{j,d}^{(t)} - x_{i,d}^{(t)}\right) + \alpha \cdot \varepsilon_{i,d}^{(t)} \tag{4}$$

where $x_{i,d}$ and $x_{j,d}$ is the d^{th} dimension of firefly i and j, respectively, a is a random value with the range of [0,1], $\varepsilon_{i,d}$ is a Gaussian random number for the d^{th} dimension, and t indicates the index of generation.

The operating parameters for the FA algorithm is given in Table 1.

Table 1: Table of FA parameters

Parameter	value
α	0.5
β_0	0.2
γ	1.0

The general outline of the Firefly algorithm is given in the following Algorithm 2.

input : *Iterations*: maximum number of iterations
D: dimension of the problem
$Fireflies$: number of fireflies f_i
I: light intensity
γ: light absorption coefficient
Bounds: Problem bounds (U - upper bound, L - lower bound)
output: $gBest$: best firefly found

```
for i = 1,...,Fireflies do
    /* generate fireflies randomly                                    */
    f_i = L+rand[0,1](U-L)
    /* calculate particles fitness                                    */
    I_i = f(f_i)
end for
for t = 1,...,Iterations do
    for i = 1,...,Fireflies do
        for j = 1,...,Fireflies do
            if I_j < I_i then
                /* Move firefly j towards firefly i (Eqn.3)           */
                /* Attractiveness varies with distance r via exp[-γr]
                   (Eqn.2)                                            */
                /* evaluate and update the worst firefly in population
                   (Eqn.4)                                            */
            end if
        end for
    end for
end for
```

Algorithm 2: Firefly Algorithm

4 Harmony Search

In order to explain the Harmony Search in more detail, let us first idealize the improvisation process by a skilled musician. When a musician is improvising, he or she has three possible choices:

- play any famous piece of music (a series of pitches in harmony) exactly from his or her memory;
- play something similar to a known piece (thus adjusting the pitch slightly);
- compose new or random notes. Zong Woo Geem et al. formalized these three options into quantitative optimization process in 2001, and the three corresponding components become: usage of harmony memory, pitch adjusting, and randomization

The usage of harmony memory is important, as it is similar to the choice of the best-fit individuals in genetic algorithms. This will ensure that the best harmonies will be carried over to the new harmony memory. In order to use this memory more effectively, it is typically assigned as a parameter $r_{accept} \in [0, 1]$, called harmony memory accepting or considering rate. If this rate is too low, only few best harmonies are selected and it may converge too slowly. If this rate is extremely high (near 1), almost all the harmonies are used in the harmony memory, then other harmonies are not explored well, leading to potentially wrong solutions. Therefore, typically, we use $r_{accept} = 0.7$ to 0.95.

The second component is the pitch adjustment determined by a pitch bandwidth b_{range} and a pitch adjusting rate r_{pa}. Though in music, pitch adjustment means to change the frequencies, it corresponds to generate a slightly different solution in the Harmony Search algorithm. In theory, the pitch can be adjusted linearly or nonlinearly, but in practice, linear adjustment is used. So we have

$$x_{new} = x_{old} + b_{range} \cdot \varepsilon \tag{5}$$

where x_{old} is the existing pitch or solution from the harmony memory, and x_{new} is the new pitch after the pitch adjusting action. This essentially produces a new solution around the existing quality solution by varying the pitch slightly by a small random amount $[1, 2]$. Here ε is a random number generator in the range of $[-1, 1]$. Pitch adjustment is similar to the mutation operator in genetic algorithms. We can assign a pitch-adjusting rate (r_{pa}) to control the degree of the adjustment. A low pitch adjusting rate with a narrow bandwidth can slow down the convergence of HS because the limitation in the exploration of only a small subspace of the whole search space. On the other hand, a very high pitch-adjusting rate with a wide bandwidth may cause the solution to scatter around some potential optima as in a random search. Thus, we usually use $r_{pa} = 0.1$ to 0.5 in most applications.

The third component is the randomization, which is to increase the diversity of the solutions. Although adjusting pitch has a similar role, but it is limited to certain local pitch adjustment and thus corresponds to a local search. The use of randomization can drive the system further to explore various diverse solutions so as to find the global optimality.

```
Function Harmony Search() /* Harmony Search Algorithm routines */
    Objective function f(x), x = (x_1, x_2, ..., x_d)^T
    Generate initial harmonics (real number arrays)
    Define pitch adjusting rate (r_pa), pitch limits and bandwidth
    Define harmony memory accepting rate (r_accept)
    while !Terination_Criteria do
        Generate new harmonics by accepting best harmonics
        Adjust pitch to get new harmonics (solutions)
        if rand < r_accept then
            choose an existing harmonic randomly;
        else if rand < r_pa then
            adjust the pitch randomly within limits;
        else generate new harmonics via randomization;
        Accept the new harmonics (solutions) if better
    end while
    Find the current best solution
```

Algorithm 3: Harmony Search Algorithm

The three components in harmony search can be summarized as the pseudocode shown in Algorithm 3. In this pseudocode, we can see that the probability of randomization is:

$$P_{random} = 1 - r_{accept} \tag{6}$$

and the actual probability of adjusting pitches is

$$P_{pitch} = r_{accept} \cdot r_{pa} \tag{7}$$

5 Experimentation

The student is required to code all three algorithms in the language of their choice. Ideally, these algorithms should share same auxiliary structures, such as population generation, memory management, problems definitions etc.

The experimentation parameters is given in Table 2.

Table 2: Experiment parameters

Parameters	Values
Population size	500 (min)
Iterations	500 (min)
Dimensions	30

Submission

The student must submit the following separate files to canvas:

1. A detailed README
2. source codes
3. A Doxygen generated PDF
4. a LaTeX typeset report on the results and its analysis
5. all raw results in a folder

The report must contain an introduction to the algorithms, the full experimentation results in tabular format and condensed results with statistical analysis compared with the results of DE in Project 3.

The files must be submitted through Canvas by 5PM May 17, 2019.