# Archimedes Optimization Algorithm

## Modern Optimization Methods

The modern optimization methods (also called nontraditional optimization methods) have emerged as powerful and popular methods for solving complex engineering optimization problems in recent years. These methods are simulated annealing, evolutionary programming (EP), Tabu search (TS), Neural-network based methods, genetic algorithm (GA), differential evolution (DE) algorithm, particle swarm optimization (PSO) technique [6] , seeker optimization algorithm (SOA), ant colony optimization (ACO) [5] algorithm, Bat algorithm (BA) [4] and Archimedes Optimization Algorithm (AOA)[7]. Most of these methods are labeled on certain characteristics and behavior of biological, molecular, swarm of insects and neurobiological systems. These new meta-heuristic tools have been combined among themselves and with knowledge elements, as well as with more traditional approaches such as statistical analysis to solve extremely challenging problems.

## Archimedes Optimization Algorithm

### Introduction

Archimedes Optimization Algorithm (AOA) [7] is based on Archimedes’ principle which states that “Any object, totally or partially immersed in a fluid or liquid, is buoyed up by a force equal to the weight of the fluid displaced by the object.”. AOA emulates the behavior of many objects, which have different densities and volumes, immersed in the same fluid and each one tries to reach equilibrium state.

### AOA Theory

If we assume that many object are immersed in the same fluid and each one of them tries to reach equilibrium state. The object will be in the equilibrium state if the buoyant force equal to the object’s weight :

|  |  |  |
| --- | --- | --- |
|  |  | (4-1) |

Where is the density, is the volume, and is the gravity or acceleration, subscripts and are for fluid and immersed object, respectively. This equation can be rearranged as:

|  |  |  |
| --- | --- | --- |
|  |  | (4-2) |

If there is another force influenced on the object like collision with another neighbouring object (), the equilibrium state will be:

|  |  |  |
| --- | --- | --- |
|  |  | (4-3) |

### Algorithmic steps

AOA can be considered as a global optimization algorithm as it encompasses both exploration and exploitation processes.

* **Step 1: initialize the positions of all objects using (4-4)**

|  |  |  |
| --- | --- | --- |
|  |  | (4-4) |

Where is the th object in a population of objects. and are the lower and upper bounds of the search-space, respectively.

Initialize volume ( and density () for each th object using (4-5):

|  |  |  |
| --- | --- | --- |
|  |  | (4-5) |

Where rand is a random number between [ 0, 1]. Then, initialize acceleration ( ) of th object using (4-6):

|  |  |  |
| --- | --- | --- |
|  |  | (4-6) |

Finally, evaluate initial population and select the object with the best fitness value. Assign , , and .

* **Step 2: update densities, volumes using (4-7)**

|  |  |  |
| --- | --- | --- |
|  |  | (4-7) |

* **Step 3: calculate Transfer operator and density factor using (4-8) and (4-9)**

|  |  |  |
| --- | --- | --- |
|  |  | (4-8) |

where transfer increases gradually with time until reaching 1. Here and are iteration number and maximum iterations, respectively. Similarly, density decreasing factor d also assists AOA on global to local search. It decreases with time using (9):

|  |  |  |
| --- | --- | --- |
|  |  | (4-9) |

where decreases with time that gives the ability to converge in already identified promising region. Note that proper handling of this variable will ensure balance between exploration and exploitation in AOA.

* **Step 4.1: Exploration phase (collision between objects occurs)**

if , collision between objects occurs, select a random object () and update object’s acceleration for iteration using (4-10):

|  |  |  |
| --- | --- | --- |
|  |  | (4-10) |

* **Step 4.2: Exploitation phase (no collision between objects)**

If , there is no collision between objects, update object’s acceleration for iteration using (4-11):

|  |  |  |
| --- | --- | --- |
|  |  | (4-11) |

* **Step 4.3: Normalize acceleration using (4-12)**

|  |  |  |
| --- | --- | --- |
|  |  | (4-12) |

where and are the range of normalization and set to 0.9 and 0.1, respectively. The determines the percentage of step that each object will change.

* **Step 5: update position**

If (exploration phase), the object’s position for next iteration using (4-13)

|  |  |  |
| --- | --- | --- |
|  |  | (4-13) |

Where is constant equals to 2.

Otherwise, if (exploitation phase), the objects update their positions using (4-14)

|  |  |  |
| --- | --- | --- |
|  |  | (4-14) |

where is a constant equals to 6. increases with time and it is directly proportional to transfer operator and it is defined using . increases with time in range and takes a certain percentage from the best position, initially. It starts with low percentage as this results in large difference between best position and current position, consequently step-size of random walk will be high. As the search proceeds, this percentage increases gradually to decrease difference between the best position and the current position. This leads to achieving an appropriate balance between exploration and exploitation.

is the flag to change the direction of motion using (4-15):

|  |  |  |
| --- | --- | --- |
|  |  | (4-15) |

Where

* **Step 6: Evaluation**

Evaluate each object using the objective function and remember the best solution found so far. Assign , , and .

Text

Description automatically generated