Algorithm 1 [44]

```
import random
import math
def fitness(x):
return x**2 - 4*x + 3
# Define the search space
low_bound = -10
top_bound = 10
# Define the LOA parameters
pride_size = 5
max_iterations = 100
m = 5
a = 0.01
# 1Initialize the pride with random positions
Pride = [{'position': random.uniform(low_bound, top_bound), 'fitness': None}
for  i in range(pride_size)]
# Calculate the fitness value for each lion in the pride
for lion in pride:
lion['fitness'] = fitness(lion['position'])
# Set the current best lion
best_lion = min(pride, key=lambda lion: lion['fitness'])
# LOA main loop
for i in range(max_iterations):
for j, lion in enumerate(pride):
# Chasing step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
Xj = lion['position']
Xbest = best_lion['position']
new_position = Xj + a * (r1 * (Xbest - m * Xj))
# Following step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
Xj = lion['position']
Xk = pride[random.randint(0, pride_size-1)]['position']
new_position = Xj + a * (r1 * (Xk - m * Xj))
# Roaming step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
new_position = lower_bound + r1 * (top_bound - low_bound)
# Clamp the new position within the search space
Output: new_position = max(low_bound, min(top_bound, new_position))
```

Salp-Swarm Optimization

Algorithm 2 [45]

```
f(x) = x^2 - 4x + 3
import random
import math
def fitness(x):
return x**2 - 4*x + 3
# Define the search space
low_bound = -10
top_bound = 10
# Define the SSO parameters
population_size = 10
max_iterations = 100
c1 = 2
c2 = 2
w = 0.7
velocity_range = (top_bound - low_bound) / 10
population = [{'position': random.uniform(low_bound, top_bound), 'velocity':
random.uniform (velocity_range, velocity_range), 'fitness': None}
```

```
for i in range(population_size)]
# Determine each salp's fitness value in the population.
for salp in population:
salp['fitness'] = fitness(salp['position'])
# Set the current best salp
best_salp = min(population, key=lambda salp: salp['fitness'])
# SSO main loop
for i in range(max_iterations):
for j, salp in enumerate(population):
# Attraction step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
Xj = salp['position']
Xbest = best_salp['position']
Vj = salp['velocity']
new_velocity = w * Vj + c1 * r1 * (Xbest - Xj) + c2 * r2 *
(population[random.randint(0,
population_size-1)]['position'] - Xj)
new_position = Xj + new_velocity
# Repulsion step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
Xj = salp['position']
 for k, other_salp in enumerate(population):
if k != j:
 if other_salp['fitness'] < salp['fitness']:
 Xk = other_salp['position']
distance_jk = abs(Xj - Xk)
repulsion_factor = math.exp(-distance_jk)
new_position -= c1 * r1 * repulsion_factor * (Xk - Xj)
 # Check if the new position is within the search space
if new_position < low_bound:
new_position = low_bound
elif new_position > top_bound:
 new_position = top_bound
 # Update the salp's position and velocity
salp['position'] = new_position
salp['velocity'] = new_velocity
# Update the salp's fitness value
salp['fitness'] = fitness(salp['position'])
# Update the best salp
 if salp['fitness'] < best_salp['fitness']:
Output: best_salp = salp
```

Spider Monkey Optimization

```
Algorithm 3 [46]
f(x) = x^2 - 4x + 3
import random
import math
def fitness(x):
return x**2 - 4*x + 3
# Define the search space
low_bound = -10
top_bound = 10
# Define the SMO parameters
population_size = 10
max_iterations = 100
m = 1
```

```
c = 0.7
sigma = 0.1
population = [{'position': random.uniform (low_bound, top_bound), 'fitness':  None}
 for i in range(population_size)]
for spider_monkey in population:
spider_monkey['fitness'] = fitness(spider_monkey['position'])
# Set the current best spider monkey
best_spider_monkey = min (population, key=lambda spider_monkey:  spider_monkey['fitness'])
# SMO main loop
for i in range(max_iterations):
for j, spider_monkey in enumerate(population):
# Movement step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
Xj = spider_monkey['position']
Xbest = best_spider_monkey['position']
new_position = Xj + m * (Xbest - Xj) + sigma * r1
# Diffusion step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
Xj = spider_monkey['position']
Xk = population[random.randint(0, population_size-1)]['position'] new_position = Xj + c * (Xk
- Xj) + sigma * r2
# Jump step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
```

D. Whale optimization

Algorithm 4 [47]

```
f(x) = x^2 - 4x + 3
import random
import math
# Define the fitness function to be minimized
def fitness(x):
return x**2 - 4*x + 3
# Define the search space
low_bound = -10
top_bound = 10
# Define the WOA parameters
num_agents = 5
max_iterations = 100
a = 2.0
A = 0.5
encircling_bounds = [0, 2]
# Initialize the search agents with random positions
agents = [{'position': random.uniform(low_bound, top_bound)}
for i  in range(num_agents)]
# Set the current best agent
best_agent = min(agents, key=lambda agent: fitness(agent['position']))
# WOA main loop
for i in range(max_iterations):
for agent in agents:
# Encircling prey step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
A1 = 2 * A * r1 - A
C1 = 2 * r2
D1 = abs(C1 * best_agent['position'] - agent['position'])
new_position = best_agent['position'] - A1 * D1
# Spiral update step
r1 = random.uniform(0, 1)
r2 = random.uniform(0, 1)
```

```
A2 = 2 * A * r1 - A
C2 = 2 * r2
new_position = new_position - A2 * math.sin(2 * math.pi * C2) *  D1
# Search for prey step
r3 = random.uniform(0, 1)
A3 = 2 * A * r3 - A
D3 = abs (C1 * best_agent['position'] - agent['position'])
```

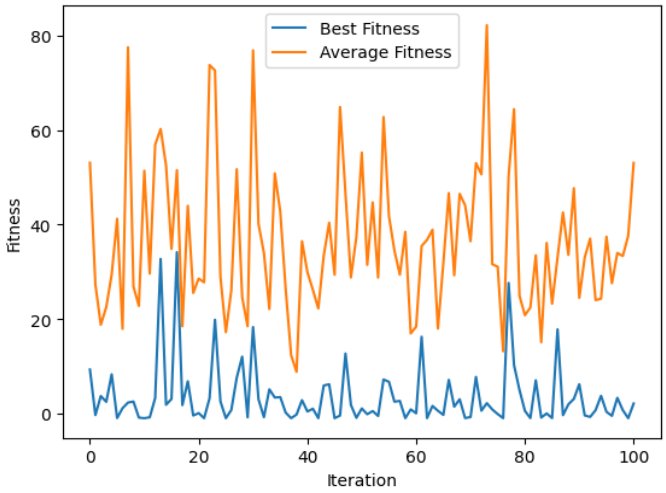

Fig. S1 Comparative analysis of best and average fitness values

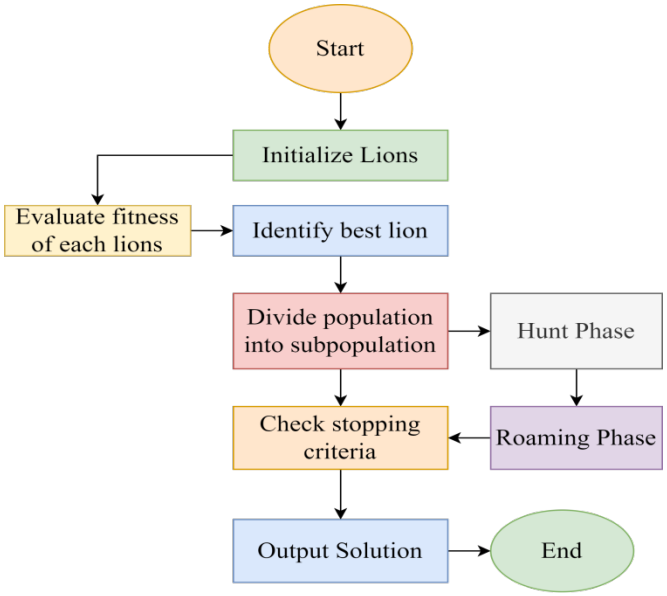

Fig. S2 Framework of LOA

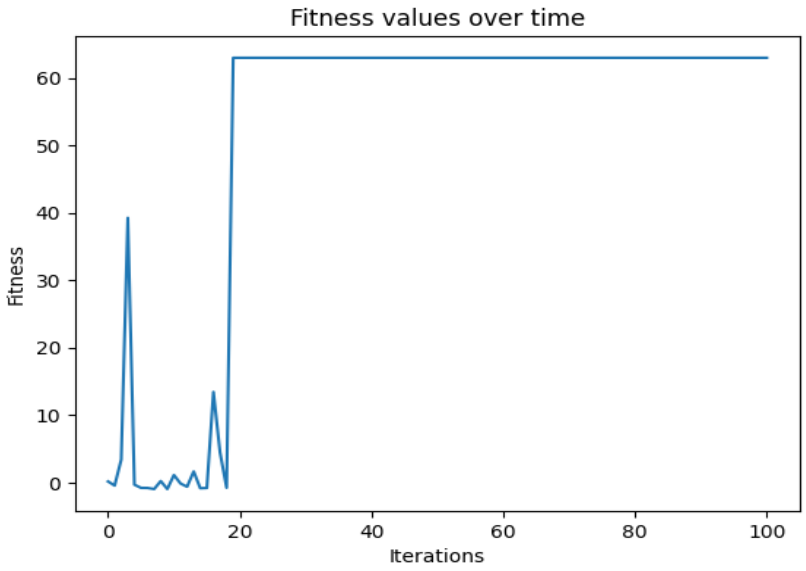

Fig.S3 Fitness values over the Time spans with various iteration

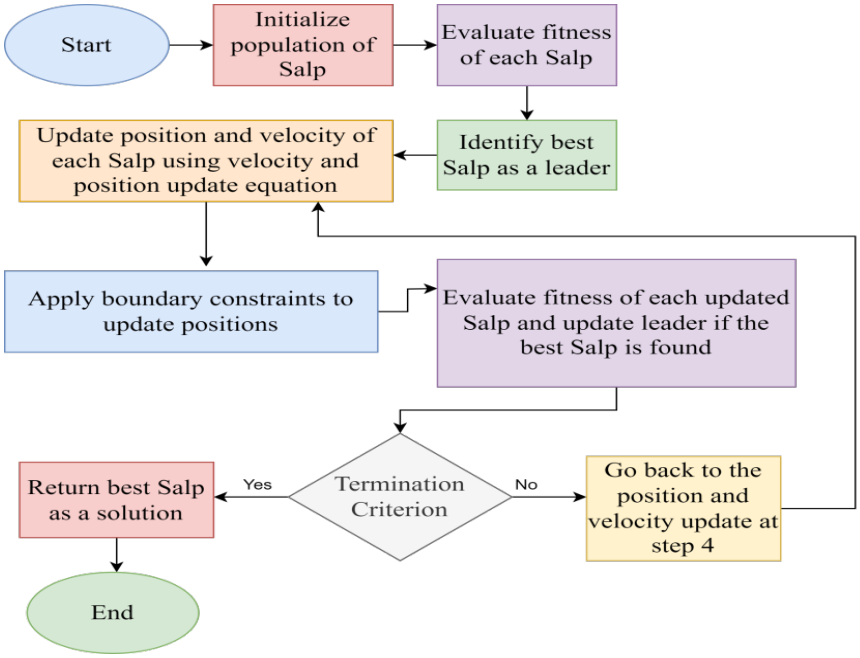

Fig.S4 SSO Framework for best Salp in DDoS Prevention

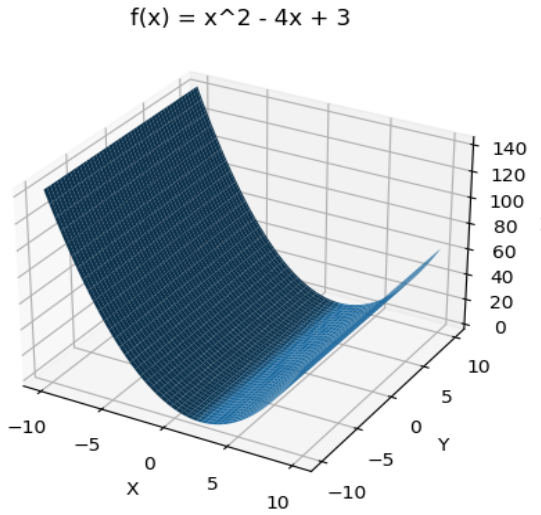

Fig.S5 SMO optimization best position for DDoS Prevention

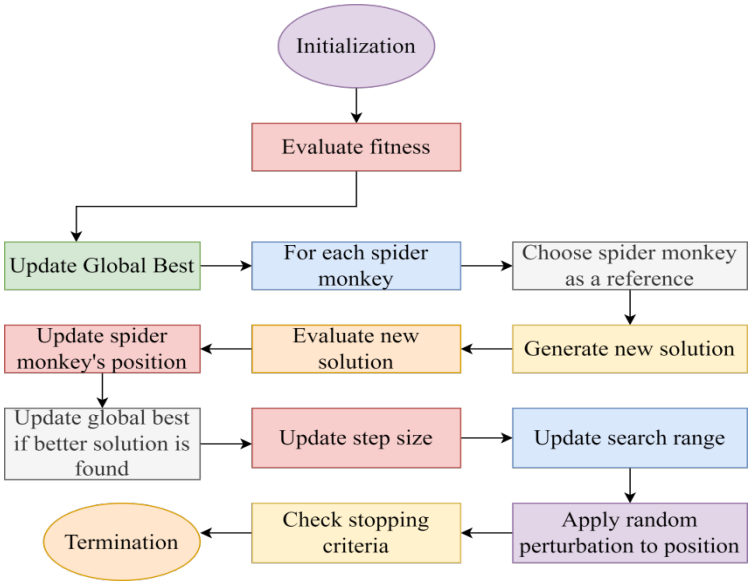

Fig.S6 SMO Framework for DDoS Prevention

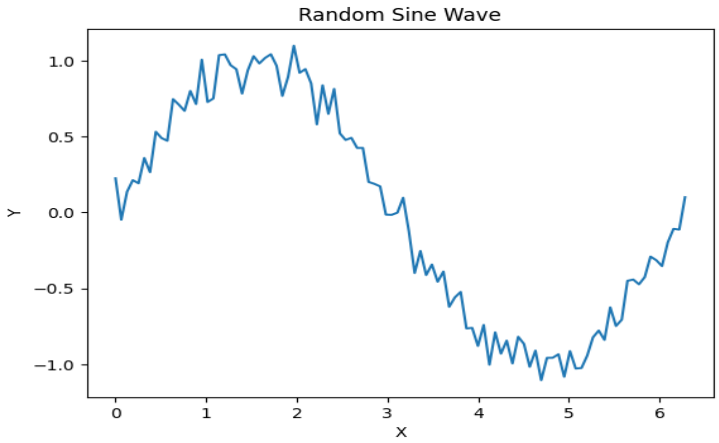

Fig.S7 WOA random sine wave generation for DDoS Prevention

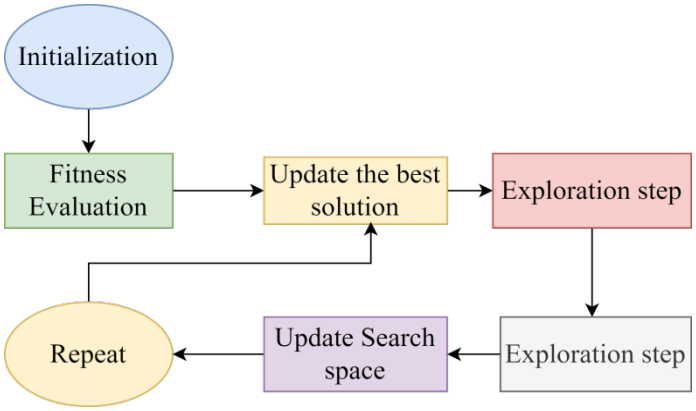

Fig.S8 Framework of WOA to optimize the best position to prevent DDoS