SYSTEM PROGRAMMING PROJECT

Report

Optimisation Algorithm : Glowworm Swarm Optimistaion

(4th Semester Project)

BTech. in Information Technology

Submitted by

Roll No	Names of Students
B416014	Arijeet Satapathy
B416044	Seva Mahapatra
B416063	Ritesh Kr. Pandey

Under the guidance of

Prof. Pushpanajli Mahapatra

Department of Computer Science and Engineering

International Institute Of Information Technology

Bhubaneswar,Odisha -751003

Spring 2018

Contents

Chapter 1

Optimization Algorithms and Methods

Optimization algorithms, which try to find the minimum values of mathematical functions, are everywhere in engineering. Among other things, theyre used to evaluate design tradeoffs, to assess control systems, and to find patterns in data.

1.1 Multi Modal Optimization

Multi-modal optimization Optimization problems are often multi-modal; that is, they possess multiple good solutions. They could all be globally good (same cost function value) or there could be a mix of globally good and locally good solutions. Obtaining all (or at least some of) the multiple solutions is the goal of a multi-modal optimizer.

This Project present the following Optimization Algorithm:

1. Glowworm Swarm Optimization[12][13]

Chapter 2

Glowworm Swarm Optimization

In this chapter, the development of the glowworm swarm optimization (GSO) algorithm is presented. Initially, the basic working principle of GSO is introduced, which is followed by a description of the phases that constitute each cycle of the algorithm.GSO, in its present form, has evolved out of several significant modifications incorporated into the earlier versions of the algorithm. Many ideas were considered in the development process before converging upon the current GSO version.

2.1 The Glowworm Swarm Optimization (GSO) Algorithm

2.1.1 Introduction

Glowworm swarm optimization (GSO), a swarm intelligence algorithm modeled after the behavior of glowworms (also known as fireflies or lightning bugs). GSO was introduced by Krishnanand and Ghose in 2005 [1]. Subsequently,the authors used GSO in various applications and several papers on this work have appeared in the literature [2, 3, 4]-[8, 9, 10]. Later, other researchers have proposed the firefly algorithm [11], which es-

sentially follows the same logic as GSO with some minor variations. GSO is originally developed for numerical optimization problems that involve computing multiple optima of multimodal functions, as against other swarm intelligence algorithms which aim to identify the global optimum.

2.1.2 Different phases of GSO

Each glowworm is attracted by, and moves toward, a single neighbor whose glow is brighter than that of itself; when surrounded by multiple such neighbors, it uses a probabilistic mechanism to select one of them.

Each glowworm uses an adaptive neighborhood to identify neighbors; it is defined by a local-decision domain that has a variable range bounded by a hard-limited sensor range $r_s(0 < r_d^i <= r_s)$.A glowworm i considers another glowworm j as its neighbor if j is within the neighborhood range of i and the luciferin level of j is higher than that of i .

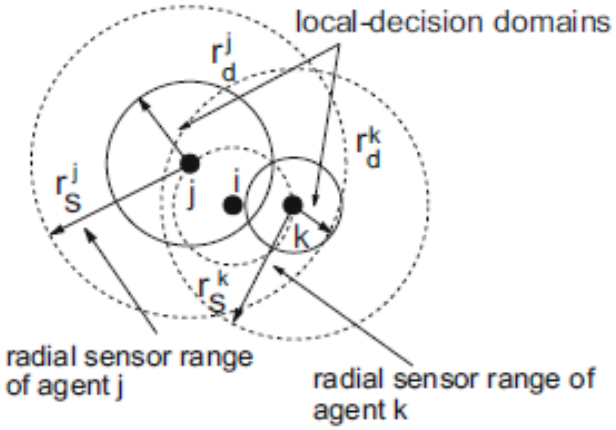


Fig 2.1 Agent i is in the sensor range of (and is equidistant to) both j and k. But, they have different decision-domains.Hence, only j uses the information of i.

For instance, in Fig. 2.1, glowworm i is in the sensor range of (and is equidistant to) both j and k. However, j and k have different neighborhood sizes, and only j uses the information of i.

2.2 Algorithm Flow Chart

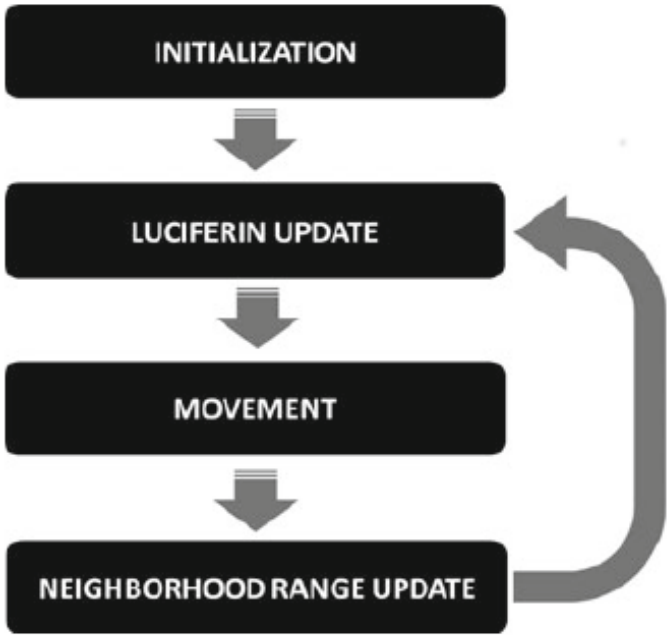


Fig 2.2 GSO Flowchart

2.3 Algorithm Description

Initially, all the glowworms contain an equal quantity of luciferin l_0. Each cycle of the algorithm consists of a luciferin update phase, a movement phase, and a neighborhood range update phase (Fig. 2.2). The GSO algorithm is given in Fig. 2.3.

Luciferin update phase: During the luciferin-update phase, each glowworm adds, to its previous luciferin level, a luciferin quantity proportional to the fitness of its current location in the objective function space. Also, a fraction of the luciferin value is subtracted to simulate the decay in luciferin with time.

The luciferin update rule is given by:

$$l_i(t+1) = (1-\rho)l_i(t) + \gamma J(x_i(t+1)) \tag{2.1}$$

where, $l_i(t)$ represents the luciferin level associated with glowworm i at time t, is the luciferin decay constant $(0 < \rho < 1)$, γ is the luciferin enhancement constant.

Movement Phase: During the movement phase, each glowworm decides, using a probabilistic mechanism, to move towards a neighbor that has a luminescence value more than its own, that is, they are attracted to neighbors that glow brighter.For each glowworm i, the probability of moving towards a neighbor j is given by:

$$p_j(t) = \frac{\tau j(t)}{\sum_{j \epsilon N_i(t)} \tau j(t)} \tag{2.2}$$

where $j \epsilon N_i(t)$, $N_i(t) = j : d(i,j) < r_d^i$ and $\tau_i(t) < \tau_j(t)$, t is the time index, d(i, j) represents the euclidean distance between glowworms i and j, $\tau_j(t)$ represents the luminescence level associated with glowworm j at time t, and rs represents the radial range of the luminescence sensor.

Let glowworm i select a glowworm j Ni (t) with pi j (t) given by (2.2). Then, the discrete-time model of the glowworm movements can be stated as:

$$x_i(t+1) = x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{||x_j(t) - x_i(t)||}\right) \tag{2.3}$$

where, $x_i(t) \epsilon R^m$ is the location of glowworm i , at time t, in the m-dimensional real space R^m , |..| represents the Euclidean norm operator, and s(>0) is the step size.

A suitable function is chosen to adaptively update the local-decision domain range of each glowworm. This is given by:

$$r_d^i(t+1) = \frac{r_s}{1 + \beta D_i(t)} \tag{2.4}$$

$$D_i(t) = \frac{N_i(t)}{\pi r^2} \tag{2.5}$$

Neighborhood range update phase: Each agent i is associated with a neighborhood whose radial range rid is dynamic in nature ($0 < r_d^i <= r_s$).

GSO uses an adaptive neighborhood range in order to detect the presence of multiple peaks in a multimodal function landscape. Let r_0 be the initial neighborhood range of each glowworm (that is, $r_d^i(0) = r_0 \forall i$).

To adaptively update the neighborhood range of each glowworm, the following rule is applied:

$$r_d^i(t+1) = min\{r_s, max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\} \tag{2.6}$$

where, β is a constant parameter and n_t is a parameter used to control the number of neighbors.

Chapter 3

Psuedocode and Implementation of GSO

3.1 Algorithm Flow Chart

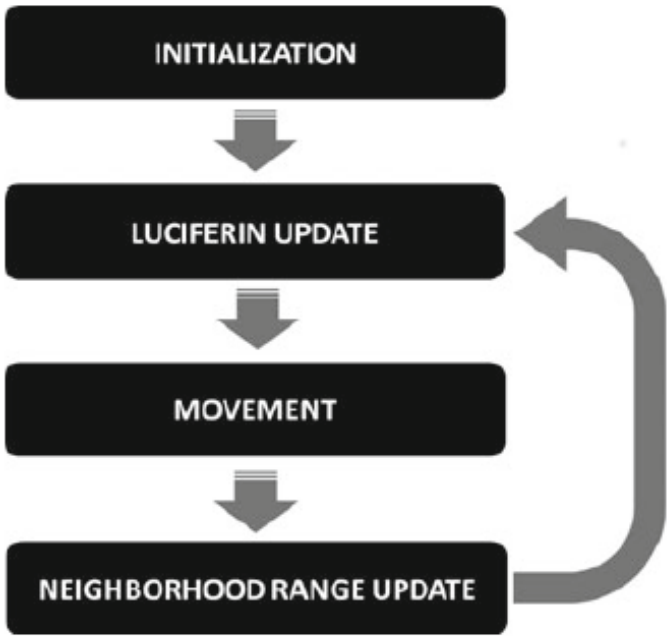


Fig 3.1 GSO Flowchart

3.2 GSO Algorithm

GLOWWORM SWARM OPTIMIZATION (GSO) ALGORITHM

Set number of dimensions $= m$
Set number of glowworms $= n$
Let s be the step size
Let $x_i(t)$ be the location of glowworm i at time t
deploy_agents_randomly;
for $i = 1$ to n do $\ell_i(0) = \ell_0$
$r_d^i(0) = r_0$
set maximum iteration number $= iter\_max$;
set $t = 1$;
while $(t \leq iter\_max)$ do:
{
for each glowworm i do: % Luciferin-update phase
$\ell_i(t) = (1-\rho)\ell_i(t-1) + \gamma J(x_i(t))$; % See Eq. (2.1)

for each glowworm i do: % Movement-phase
{
$N_i(t) = \{j : d_{ij}(t) < r_d^i(t); \ell_i(t) < \ell_j(t)\}$;
for each glowworm $j \in N_i(t)$ do:
$p_{ij}(t) = \frac{\ell_j(t)-\ell_i(t)}{\sum_{k\in N_i(t)} \ell_k(t)-\ell_i(t)}$; % See Eq. (2.2)
$j = select\_glowworm(\vec{p})$;
$x_i(t+1) = x_i(t) + s\left(\frac{x_j(t)-x_i(t)}{\|x_j(t)-x_i(t)\|}\right)$ % See Eq. (2.3)
$r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\}$;

}
$t \leftarrow t+1$;
}

3.3 Python Code

We peresent an example of 20 worms on a 10 x 10 grid optimizing for maximum. Given a worms location, its' score is determined by the fitness function, which in this example was made up to provide multiple local maximums within the area.

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
from matplotlib.collections import PatchCollection
from scipy.spatial import distance as dist
import copy
%matplotlib inline


# Since the fitness function rates some worms negatively,
#the lower bound exists
# to add to all of the scores so they are
#positive before normalization
# The influence factor is divided form all the
#positive scores to determine the
# range of influence.

dims = 10
num_worms = 20
nturns = 100
lower_bound = 70
influence_factor = 30
max_jitter = .2

def fitness_function(xy_tuple):
    x = xy_tuple[0]/2
    y = xy_tuple[1]/2
    return (np.sin(x**3+ (y-5)**3) + np.cos((y-5)**2) *10
    + np.cos(x*2) * 12 * (y-5))

def starting_points():
    return np.random.rand(num_worms,2) * 10
```

```
def get_score(pop):
    temp = [ (fitness_function(tup)) for tup in pop ]
    normal = [ x + lower_bound for x in temp ]
    return [ x / influence_factor for x in normal ]

# This returns a matrix with the distances between each
#worm calculated if the
# worm in the row is influenced by the worm in the column,
#else 0

def influence_matrix(pop, score):
    graph = np.array([np.zeros(num_worms)] * num_worms)

    for i in range(num_worms):
        for j in range(num_worms):
            if i == j:
                graph[i][j] = 0
            elif dist.euclidean(pop[i], pop[j]) <= score[j]:
                graph[i][j] = dist.euclidean(pop[i], pop[j])
            else:
                continue
    return graph

def next_turn(pop, score, im):
    n_turn = copy.deepcopy(pop)

    # X and Y movement is determined by the ratio of
    #distance between worms and
    # the radius of the influencing worm
    # This ensures that closer worms will have more
    # influence than further worms
    # with the same pull, and at the same time worms with
    # large influences will
    # have more pull than other worms of the same distance
    for i in range(num_worms):
        x_move = 0
        y_move = 0
        for j in range(num_worms):
            percent_move = 1 - (im[i][j] / score[j]) if
            im[i][j] != 0 else 0
            x_move += (pop[j][0] - pop[i][0]) *
            percent_move / 10 if score[i] < score[j] else 0
            y_move += (pop[j][1] - pop[i][1]) *
```

```
                percent_move / 10 if score[i] < score[j] else 0
            jitter_x = max_jitter * np.random.rand() * np.random.randint(-1,2)
            jitter_y = max_jitter * np.random.rand() * np.random.randint(-1,2)
            n_turn[i][0] += x_move + jitter_x
            n_turn[i][1] += y_move + jitter_y

            n_turn[i][0] = keep_in_bounds(n_turn[i][0], dims)
            n_turn[i][1] = keep_in_bounds(n_turn[i][1], dims)

    return n_turn

def keep_in_bounds(x,dims):
    if x < 0:
        return 0
    elif x > dims:
        return dims
    else:
        return x


### START ###
pop = starting_points()
for each in range(nturns):
    score = get_score(pop)


    ### PLOT ###
    plt.rcdefaults()
    fig, ax = plt.subplots()

    x = np.arange(0, dims, 0.1)
    y = np.arange(0, dims, 0.1)
    xx, yy = np.meshgrid(x, y, sparse=True)
    z = fitness_function([xx, yy])
    im = plt.imshow(z, interpolation='bilinear',
    origin='lower', cmap=cm.gray, extent=(0, dims, 0, dims ))
    plt.contour(x,y,z,alpha=0.1)

    plt.plot(pop[:,0], pop[:,1], 'ro' )

    patches = []
    for i in range(0,num_worms):
        patches.append( mpatches.Circle((pop[i][0],
```

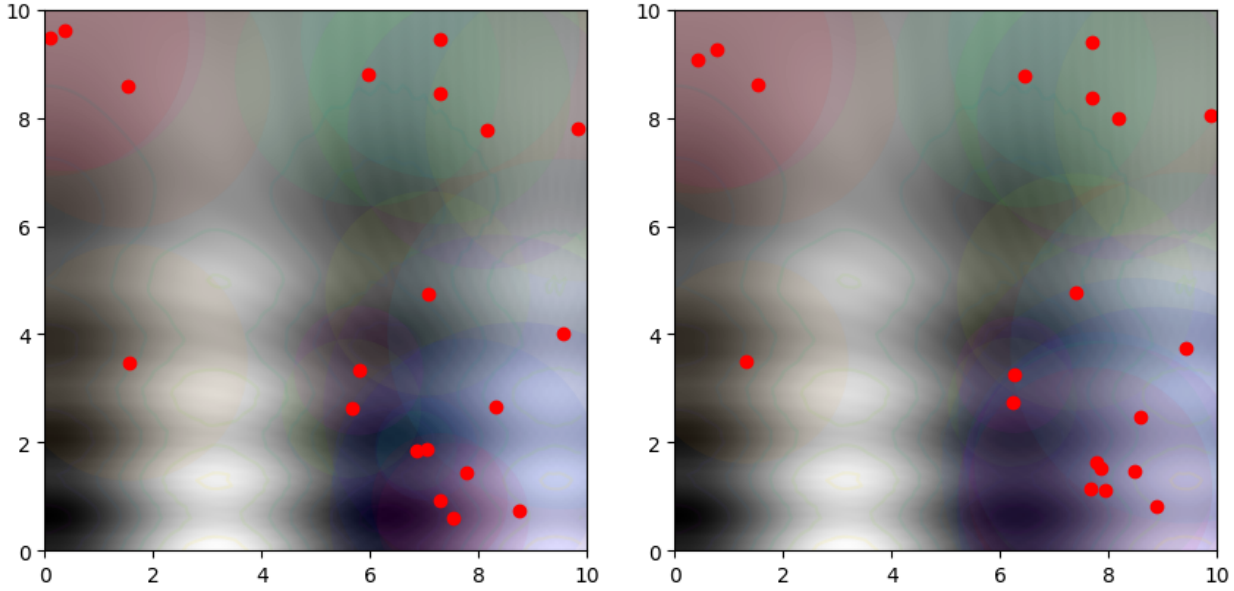
```
            pop[i][1]), score[i], ec = "none") )

        colors = np.linspace(0, 1, len(patches))
        collection = PatchCollection(patches,
        cmap=plt.cm.hsv, alpha=0.05)
        collection.set_array(np.array(colors))
        ax.add_collection(collection)
        #####

        plt.show()
        #print()
        name = "0000000" + str(each)
        name = name[-4:] + '.png'
        #plt.savefig(name, bbox_inches='tight')

        im = influence_matrix(pop,score)
        pop = copy.deepcopy(next_turn(pop,score,im))
plt.close("all")
```

These are the result of the code when implemented:



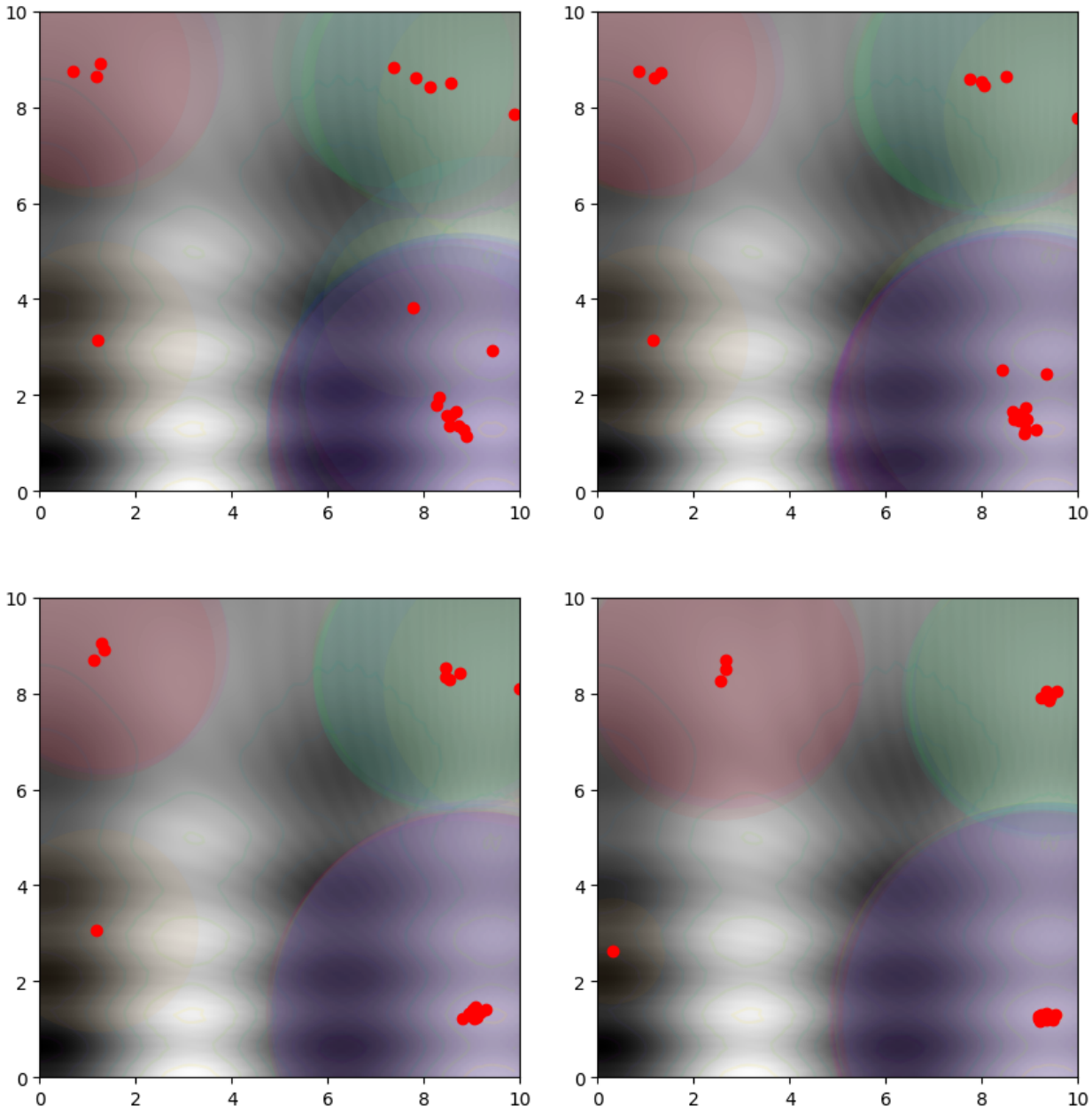


Fig 3.2 Python Implementation Of GSO

References

[1] K.N. Krishnanand and D. Ghose. Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In Proceedings of the IEEE Swarm Intelligence Symposium, pages 8491. Pasedena, California, June 2005.

[2] K.N. Krishnanand. Glowworm Swarm Optimization: A Multimodal Function Optimization Paradigm with Applications to Multiple Signal Source Localization Tasks. PhD thesis, Indian Institute of Science, Bangalore, 2007.

[3] K.N. Krishnanand, P. Amruth,M.H. Guruprasad, S.V. Bidargaddi, and D. Ghose. Glowworminspired robot swarm for simultaneous taxis towards multiple radiation sources. In Proceedings of the IEEE International Conference on Robotics and Automation, pages 958963. Orlando, Florida, May 2006.

[4] K.N. Krishnanand and D. Ghose. Multimodal function optimization using a glowworm metaphor with applications to collective robotics. In Proceedings of the Indian International Conference on Artificial Intelligence, pages 328346. Pune, India, 2005.

[5] K.N. Krishnanand and D. Ghose. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications. Multiagent and Grid Systems, 2(3):209 222, 2006.

[6] K.N. Krishnanand and D. Ghose. Design and Control of Intelligent Robotic Systems, chapter AGlowworm Swarm OptimizationBased Multi-robot System for Signal Source Localization, pages 4968. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[7] K.N. Krishnanand and D. Ghose. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intelligence, 3(2):87124, 2009.

[8] K.N. Krishnanand and D. Ghose. Handbook of Swarm Intelligence: Concepts, Principles and Applications, chapter Glowworm swarm optimization for multimodal search spaces, pages 451467. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[9] K.N. Krishnanand, A. Puttappa, G.M. Hegde, S.V. Bidargaddi, and D. Ghose. Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006, Brussels, Belgium, September 47, 2006. Proceedings, chapter Rendezvous of Glowworm-Inspired Robot Swarms at Multiple Source Locations: A Sound Source Based Real-Robot Implementation, pages 259269. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[10] K.N. Krishnanand andD.Ghose. Glowworm swarm optimisation: a newmethod for optimising multi-modal functions. International Journal of Computational Intelligence Studies, 1(1): 93 119, 2009.

[11] X-S Yang. Proceedings of the 5th International Symposium on Stochastic Algorithms: Foundations and Applications, Sapporo, Japan, chapter Firefly algorithms for multimodal optimization, pages 169178. Springer Berlin Heidelberg, Berlin, Heidelberg, October 2009.

[12] K.N. Krishnanand, D. Ghose, Detection of Multiple Source Locations using a Glowworm Metaphor with Applications to Collective Robotics, Swarm Intelligence Symposium, 2005, pp. 84-91.

[13] Kaipa, K. and Ghose, D. (2017). *Glowworm Swarm Optimization:Theory, Algorithms, and Applications*. Cham: Springer.