14

Glowworm Swarm Optimization: A Tutorial

Krishnanand Kaipa
Department of Mechanical and Aerospace Engineering
Old Dominion University, Norfolk, Virginia, United States

Debasish Ghose
Department of Aerospace Engineering
Indian Institute of Science, Bangalore, India

CONTENTS

14.1 Introduction 175
14.1.1 Basic principle of GSO 176
14.1.2 The Glowworm Swarm Optimization (GSO) algorithm 177
14.1.3 Algorithm description 178
14.2 Source-code of GSO algorithm in Matlab 180
14.3 Source-code of GSO algorithm in C++ 183
14.4 Step-by-step numerical example of GSO algorithm 185
14.5 Conclusions 190
References 190

14.1 Introduction

This book chapter focuses on the Glowworm Swarm Optimization (GSO) algorithm [1-3], a swarm intelligence algorithm loosely based on the behavior of glowworms, which are also known as fireflies or lightning bugs. Although there are other swarm intelligence algorithms in the literature based on certain perceived behaviors of fireflies (for instance, the firefly algorithm [4] and its extensions), the GSO algorithm precedes these by several years. The GSO algorithm was originally intended to be implementable in robotic systems where a team of mobile robots, or glowworms, equipped with sensors that would enable them to mimic the sensing capabilities of glowworms, would explore a signal landscape to identify multiple signal sources. The mathematical formulation of this problem devolved into a numerical optimization problem requiring the computation of multiple optima of a multimodal function (as against computation of the global optimum of such a function that most other swarm intelli-

gence algorithms aim for). Glowworm swarm optimization (GSO), inspired by the behavior of glowworms, is a swarm intelligence algorithm introduced by Kaipa and Ghose in 2005 [1]; it subsequently underwent several experiments, modifications and changes by the authors and other researchers to evolve into a stable algorithm that not only served the purpose of computing multiple optima of a multimodal function, but was also designed in a way that would make it suitable for implementation in a swarm of mobile robots. This evolution of GSO and its various versions is described in detail in the book by Kaipa and Ghose [3].

Most swarm intelligence algorithms seek out the global optimum of a multimodal function as it is considered to be the best solution to a mathematical formulation of an optimization problem. The GSO works on a somewhat different premise that arises from the real world problem which the optimization problem seeks to solve. Often, the dynamic nature of the constraints in the search space makes a previous optimum solution infeasible to implement, making an alternative locally optimum solution more feasible. One domain where identification of multiple optimum solutions is required is financial decision making where locally feasible examples are used as a basis for learning [5] and in cases where it is important to identify diverse rules as a basis for a classifier [6]. Perhaps the most easily understood example of the need to search for multiple local optima is the problem of identifying multiple sources of signals in a landscape.

GSO fosters decentralized decision-making and movement protocols that are implementable in swarm robotics applications. For instance, a robot swarm can use the search protocol of GSO to move in an unknown region to carry out disaster response tasks comprising searches for multiple unknown signal sources. Examples of such applications include identifying nuclear spill points and hazardous chemical spill locations, identifying leaks in pressurized systems, and locating forest fires and dowsing them. It has been recognized that the GSO's approach of explicitly having the capability of partitioning a swarm into sub-swarms that would lead to multiple source localization is a very effective device [7], and while other methods have addressed this problem in an indirect way, GSO is the first algorithm to do so directly. In this chapter, the basic working principle of GSO is introduced, which is followed by a description of the phases that constitute each cycle of the algorithm. Next, the pseudocode, MATLAB code, and C++ code of the algorithm are presented. Finally, the working of various steps of the algorithm is illustrated by using a numerical example.

14.1.1 Basic principle of GSO

GSO is developed based on the behavior pattern of glowworms by which they can change the intensity of bio-luminescence and appear to glow at different intensities. Real glowworms carry a luminous pigment called luciferin, which is the source of bio-luminiscence. In the GSO algorithm, the quantity of luciferin encodes the fitness of its location in the search space. This makes the agent

glowworms glow at an intensity approximately proportional to the function value being optimized. It is assumed that agents glowing that are brighter attract those that glow with lower intensity. In the algorithm, each glowworm selects, using a probabilistic mechanism, a neighbor that has a luciferin value higher than its own and moves toward it.

A critical aspect of the GSO algorithm is that it incorporates an adaptive neighborhood range by which the effect of distant glowworms is discounted when a glowworm has a sufficient number of neighbors with brighter glow or when the range to a neighboring agent goes beyond the maximum range of perception. These movements, based only on local information and selective neighbor interaction, enable the swarm of glowworms to split into disjoint subgroups that converge to high function value points. It is this property of the algorithm that allows it to be used to identify multiple optima of a multimodal function. It has been shown [3] that GSO can tackle the following class of multimodal functions: unequal peaks, equal peaks, peaks of concentric circles, peaks surrounded by regions with step-discontinuities (non-differentiable objective functions), peaks comprising plateaus of equal heights, peaks located at irregular intervals, change in landscape features with change in scale, and non-separability involving interdependence of objective function variables. In addition to these, other researchers have shown its applicability to a much larger class of problems involving multiple optima.

In real life, natural glowworms are attracted to brighter glowing mates and form a cluster of glowworms that collectively grow brighter than their more scattered neighbors, which are, in turn, attracted to this brighter glow. The general idea in GSO is similar in the following aspects:

1. Agents are assumed to be attracted to move toward other agents that have higher luciferin value (brighter luminescence).
2. The multiple peaks can be likened to the nuclei of clusters that serve as bright beacons.

14.1.2 The Glowworm Swarm Optimization (GSO) algorithm

GSO initially distributes a swarm of agents randomly in the search space. These agents will be called glowworms from now onwards. Further, they have other behavioral traits that are not found in their natural counterparts. Accordingly, the algorithm encapsulates the interplay between the following three mechanisms:

1. **Fitness broadcast**: Glowworms carry a luminescent pigment called *luciferin*, whose quantity encodes the fitness of their locations in the objective space. This allows them to glow at an intensity that is proportional to the function value being optimized. It is assumed

that the luciferin level of a glowworm as sensed by its neighbor does not reduce due to distance[1].

2. **Positive taxis**: 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 (described in Section 14.1.3) to select one of them.
3. **Adaptive neighborhood**: Each glowworm uses an adaptive neighborhood to identify neighbors; it is defined by a local-decision domain that has a variable range r_d^i bounded by a hard-limited sensor range r_s ($0 < r_d^i \leq r_s$). A suitable heuristic is used to modulate r_d^i (described in Section 14.1.3). 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.

Note that the glowworms depend only on information available in the local-decision domain to decide their movements. Each glowworm selects, using a probabilistic mechanism, a neighbor that has a luciferin value higher than its own and moves toward it. These movements, that are based only on local information and selective neighbor interactions, enable the swarm of glowworms to partition into disjoint subgroups that steer toward, and meet at, multiple optima of a given multimodal function.

The significant difference between GSO and most earlier approaches to multimodal function optimization problems is the adaptive local-decision domain, which is used effectively to locate multiple peaks.

The description given above has been abstracted from the description in [2, 3].

14.1.3 Algorithm description

In the following, the algorithm is explained through maximization problems. However, the algorithm can be easily modified and used to find multiple minima of multimodal functions. GSO starts by placing a population of n glowworms randomly in the search space so that they are well dispersed. 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. The GSO algorithm is given in Alg. 13. The following description has been abstracted from [2, 3].

Algorithm 13 Glowworm Swarm Optimization (GSO) Algorithm.

1: Set number of dimensions m
2: Set number of glowworms n
3: Set step size s

[1] In natural glowworms, the brightness of a glowworm's glow as perceived by its neighbor reduces with increase in the distance between the two glowworms.

for $(i = 1 : n)$ **do**
  $x_i(0) \leftarrow deploy\_glowworm\_randomly(i, m)$
  $\ell_i(0) \leftarrow \ell_0$
  $r_d^i(0) \leftarrow r_0$
end for
Set maximum iteration number $= t_{max}$
Set $t = 1$
while $(t \leq t_{max})$ **do**
  for $(i = 1 : n)$ **do**
    $\ell_i(t) \leftarrow (1 - \rho)\ell_i(t - 1) + \gamma J(x_i(t))$
  end for
  for $(i = 1 : n)$ **do**
    $N_i(t) \leftarrow \{j : d_{ij}(t) < r_d^i(t); \ell_i(t) < \ell_j(t)\}$
    for $(j \in N_i(t))$ **do**
      $p_{ij}(t) \leftarrow \frac{\ell_j(t) - \ell_i(t)}{\sum_{k \in N_i(t)} \ell_k(t) - \ell_i(t)}$
    end for
    $j \leftarrow select\_glowworm(\vec{p})$
    $x_i(t + 1) \leftarrow x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}\right)$
    $r_d^i(t + 1) \leftarrow \min\{r_s, \max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\}$
  end for
  $t \leftarrow t + 1$
end while

Luciferin update phase: The luciferin update depends on the function value at the glowworm position. 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:

$$\ell_i(t + 1) \quad = \quad (1 - \rho)\ell_i(t) + \gamma J(x_i(t + 1)) \tag{14.1}$$

where $\ell_i(t)$ represents the luciferin level associated with glowworm i at time t, ρ is the luciferin decay constant $(0 < \rho < 1)$, γ is the luciferin enhancement constant and $J(x_i(t))$ represents the value of the objective function at agent i's location at time t.

Movement phase: During the movement phase, each glowworm decides, using a probabilistic mechanism, to move toward a neighbor that has a luciferin value higher than its own. That is, glowworms are attracted to neighbors that glow brighter. For each glowworm i, the probability of moving toward a neighbor j is given by:

$$p_{ij}(t) \quad = \quad \frac{\ell_j(t) - \ell_i(t)}{\sum_{k \in N_i(t)} \ell_k(t) - \ell_i(t)} \tag{14.2}$$

where $j \in N_i(t)$ and

$$N_i(t) = \{j : d_{ij}(t) < r_d^i(t) \text{ and } \ell_i(t) < \ell_j(t)\} \tag{14.3}$$

is the set of neighbors of glowworm i at time t, $d_{ij}(t)$ represents the Euclidean distance between glowworms i and j at time t, and $r_d^i(t)$ represents the variable neighborhood range associated with glowworm i at time t. Let glowworm i select a glowworm $j \in N_i(t)$ with $p_{ij}(t)$ given by (14.2). Then, the discrete-time model of the glowworm movements can be stated as:

$$x_i(t+1) = x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{\|x_j(t) - x_i(t)\|}\right) \tag{14.4}$$

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

Neighborhood range update phase: Each agent i is associated with a neighborhood whose radial range r_d^i is dynamic in nature $(0 < r_d^i \leq r_s)$. The fact that a fixed neighborhood range is not used needs some justification. When the glowworms depend only on local information to decide their movements, it is expected that the number of peaks captured would be a function of the radial sensor range. In fact, if the sensor range of each agent covers the entire search space, all the agents move to the global optimum and the local optima are ignored. Since we assume that *a priori* information about the objective function (e.g., number of peaks and inter-peak distances) is not available, it is difficult to fix the neighborhood range at a value that works well for different function landscapes. For instance, a chosen neighborhood range r_d would work relatively better on objective functions where the minimum inter-peak distance is more than r_d rather than on those where it is less than r_d. Therefore, 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:

$$r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\} \tag{14.5}$$

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

The quantities $\rho, \gamma, s, \beta, n_t$, and ℓ_0 are algorithm parameters for which appropriate values have been determined based on extensive numerical experiments and are kept fixed in this book (Table 14.1).The quantity r_0 is made equal to r_s in all the experiments. Thus, n and r_s are the only parameters that influence the algorithm behavior (in terms of the total number of peaks captured) and need to be selected.

TABLE 14.1
Values of algorithmic parameters that are kept fixed for all the experiments.

ρ	γ	β	n_t	s	ℓ_0
0.4	0.6	0.08	5	0.03	5

14.2 Source-code of GSO algorithm in Matlab

The Matlab source-code for optimizing the objective function $J(.)$ using the GSO algorithm is given in GSO.m below. In this code, GSO maximizes a two-dimensional objective function $J(.)$:

$$\begin{aligned} J(x,y) &= 3(1-x)^2 e^{-[x^2+(y+1)^2]} - 10\left(\frac{x}{5} - x^3 - y^5\right) e^{-(x^2+y^2)} \\ &- \left(\frac{1}{3}\right) e^{-[(x+1)^2+y^2]} \end{aligned} \tag{14.6}$$

where the input is the set of locations of all the swarm members stored in A, $x = A_i(1)$ and $y = A_i(2)$, for $i \in 1, 2, \ldots, n$. The result of the GSO code is an n-dimensional column vector with the objective function values for each glowworm i.

```
%---GSO.m (main front-end code)-------
global n m A_init A Ell gamma ro step1 r_d r_s ...
       beta r_min n_t Ave_d bound
%-----Parameter initialization---------
m = 2;                  % No. of dimensions
n     = 100;            % No. of agents
r_s   = 3;              % Sensor range
r_d   = r_s*ones(n,1); % Local decision range
r_min = 0;              % Threshold decisin range
gamma = 0.6;            % Luciferin enhancement constant
ro    = 0.4;            % Luciferin decay constant
step1 = 0.03;           % Distance moved by a glowworm at each
    step
beta  = 0.08;           % decision range gain
n_t   = 5;              % Desired no. of neighbors
%-----Initialization of variables-------
bound = 3;              % Workspace range parameter
DeployAgents;           % Deploy the glowworms randomly
Ell = 5*ones(n,1);      % Initialization of Luciferin levels
j = 1;                  % Iteration index
iter = 250;             % No. of iterations
Ave_d = zeros(iter,1); % Average distance
%-----Main loop-------------------------
while (j <= iter)
   UpdateLuciferin;     % Update luciferin levels of glowworms
   Act;                 % Select a direction and move
   for k = 1 : n        % store the state histories
    agent_x(k,j,:) = A(k,1); agent_y(k,j,:) = A(k,2);
   end
   j = j + 1;
```

```
end
figure(1); % Plot of glowworm trajectories
plot(A_init(:,1),A_init(:,2),'x'); xlabel('X'); ylabel('Y');
hold on;
DefineAxis;
for k = 1 : n plot(agent_x(k,:,:),agent_y(k,:,:)); end
plot([-0.0093;1.2857;-0.46], [1.5814;-0.0048;-0.6292],'ok');
figure(2); % Plot of final locations of glowworms
plot(A(:,1),A(:,2),'.'); DefineAxis; grid on; hold on;
plot([-0.0093;1.2857;-0.46], [1.5814;-0.0048;-0.6292],'ok');
% Functions ----------------------
function DeployAgents % Fun 1: DeployAgents.m
global n m A_init A bound
B = -bound*ones(n,m); A_init = B + 2*bound*rand(n,m); A = A_init;
function UpdateLuciferin % Fun 2: UpdateLuciferin.m
global n A J Ell gamma ro
for i = 1 : n
        x = A(i,1); y = A(i,2);
        J(i,:) = 3*(1-x)^2*exp(-(x^2) - (y+1)^2) ...
                 - 10*(x/5 - x^3 - y^5)*exp(-x^2-y^2) ...
                 - 1/3*exp(-(x+1)^2 - y^2);
        Ell(i,:) = (1-ro)*Ell(i,:) + gamma*J(i,:);
end %--------------------------------------------------
function Act % Fun 3: Act.m
global n r_s r_d N N_a beta n_t
N(:,:) = zeros(n,n); N_a(:,:)= zeros(n,1);
for i = 1 : n
    FindNeighbors(i); FindProbabilities(i);
    Leader(i) = SelectAgent(i);
end %--------------------------------------------------
for i = 1 : n
    Move(i,Leader(i));
    r_d(i) = max(0, min(r_s,r_d(i) + beta*(n_t-N_a(i))));
end %--------------------------------------------------
function FindNeighbors(i) % Fun 4: FindNeighbors.m
global n m A N r_d N_a Ell
n_sum = 0;
for j = 1 : n
    if (j~=i) square_sum = 0;
        for k = 1 : m
            square_sum = square_sum + (A(i,k)-A(j,k))^2;
        end
        d = sqrt(square_sum);
        if (d <= r_d(i)) & (Ell(i) < Ell(j))
            N(i,j) = 1; n_sum = n_sum + 1;
        end
    end
    N_a(i) = n_sum;
end %--------------------------------------------------
function FindProbabilities(i) % Fun 5: FindProbabilities.m
global n N Ell pb
Ell_sum = 0;
for j=1:n Ell_sum = Ell_sum + N(i,j)*(Ell(j) - Ell(i)); end
if (Ell_sum == 0) pb(i,:) = zeros(1,n); else
 for j=1:n pb(i,j) = (N(i,j)*(Ell(j)-Ell(i)))/Ell_sum; end
end %--------------------------------------------------
function j = SelectAgent(i) % Fun 6: SelectAgent.m
```

```
global n pb
bound_lower = 0; bound_upper = 0; toss = rand; j = 0;
for k = 1 : n
 bound_lower = bound_upper; bound_upper = bound_upper + pb(i,k);
 if (toss > bound_lower) & (toss < bound_upper) j = k; break; end
end %--------------------------------------------------------
function Move(i,j) % Fun: Move.m
global A m step1 Ell bound
if (j~=0) & (Ell(i) < Ell(j))
 temp(i,:) = A(i,:) + step1*Path(i,j); flag = 0;
 for k = 1 : m
  if (temp(i,k) < -bound)|(temp(i,k) > bound) flag=1;break;end
 end
 if (flag == 0) A(i,:) = temp(i,:); end
end %--------------------------------------------------------
function Del = Path(i,j) % Fun 8: Path.m
global A m
square_sum = 0;
for k = 1 : m
    square_sum = square_sum + (A(i,k)-A(j,k))^2;
end
hyp = sqrt(square_sum);
for k = 1 : m Del(:,k) = (A(j,k) - A(i,k))/hyp;
end %--------------------------------------------------------
function DefineAxis % Fun: DefineAxis.m
global bound
axis([-bound bound -bound bound]); grid on;
```

Listing 14.1
GSO code in Matlab.

14.3 Source-code of GSO algorithm in C++

```
#include <iostream>
#include<fstream>
#include <algorithm>
#include <math.h>
using namespace std;
// Constant Parameters
#define n 1000     // n - Number of glowworms
#define r 125.0    // r - Sensor range
#define rho 0.4    // rho - Luciferin decay constant
#define gama 0.6   // gama - Luciferin enhancement constant
#define beta 0.08 // beta - neighborhood ehancement constant
#define s 0.03
#define d 2        // d - dimension of the search space
#define nd 5       // nd - Desired number of neighbors
#define PI 3.14159
#define W 3         // W - Workspace size
#define IterMax 500
/*Variables: Lc - Luciferin value, Rd - Neighborhood range,
P - Probability matrix, Ld - Leader set, N - Neighborhood matrix,
Na - Actual number of neighbors, X - Glowworm positions */
static int Ld[n], N[n][n], Na[n], randSeed = 1;
```

```
22 static double X[n][d], Lc[n], Rd[n], P[n][n],
23                Sol[]={-PI/2, PI/2};
24 static ofstream outFile ("peaks_sol.dat",ios::app);
25 static double Distance(int i, int j) {double dis = 0;
26  for(int k = 0; k < d; k++) dis = dis + pow((X[i][k]-X[j][k]),2);
27  return sqrt(dis); }
28 static void DeployGlowworms(float lim) {
29  for(int i = 0; i < n; i++) {for(int j = 0; j < d; j++)
30  {X[i][j]=-lim+2*lim*rand()/(RAND_MAX+1.0);}}}
31 static void UpdateLuciferin() {
32  for (int i = 0; i < n; i++) {double x = X[i][0], y = X[i][1];
33   double J = 3*pow((1-x),2)* exp(-pow(x,2) - pow((y+1),2))
34     - 10*(x/5 - pow(x,3) - pow(y,5))*exp(-pow(x,2) - pow(y,2))
35     -  1/3 * exp(-pow((x+1),2) - pow(y,2));
36     Lc[i] = (1-rho)*Lc[i] + gama*J;}}
37 static void FindNeighbors() {for(int i = 0; i < n; i++) {
38  N[i][i] = 0; Na[i] = 0;for(int j = 0; j < n; j++){if (j!=i){
39  if ((Lc[i] < Lc[j]) && (Distance(i,j) < Rd[i])) N[i][j] = 1;
40     else N[i][j] = 0; Na[i] = Na[i] + N[i][j];}}}}
41 static void FindProbabilities() {for(int i = 0; i < n; i++)
42  {double sum = 0; for (int j = 0; j < n; j++)
43   sum = sum + N[i][j]*(Lc[j] - Lc[i]);
44   for(int j = 0; j < n; j++) {if (sum != 0)
45   P[i][j] = N[i][j]*(Lc[j] - Lc[i])/sum;else P[i][j] = 0;}}}
46 static void SelectLeader() {for (int i = 0; i < n; i++) {
47   double b_lower = 0; Ld[i] = i;
48   double toss = rand()/(RAND_MAX + 1.0);
49   for (int j = 0; j < n; j++) {if (N[i][j] == 1) {
50   double b_upper = b_lower + P[i][j];
51     if ((toss >= b_lower) && (toss < b_upper))
52     {Ld[i] = j;break;} else b_lower = b_upper;}}}}
53 static void Move() {for (int i = 0; i < n; i++) {if (Ld[i]!=i){
54  int flag = 0;double temp[d]; double dis = Distance(i,Ld[i]);
55  for (int j = 0; j < d; j++) {
56     temp[j] = X[i][j] + s*(X[Ld[i]][j] - X[i][j])/dis;
57     if (fabs(temp[j]) > W) {flag = 1;break;}} if (flag == 0)
58     for (int j = 0; j < d; j++) X[i][j] = temp[j];}}}
59 static void UpdateNeighborhood() {
60 for (int i = 0; i < n; i++)
61  Rd[i] = max(0.0, min(r, Rd[i] + beta*(nd - Na[i]))); }
62 int main (int argc, char * const argv[]) {
63   for (int currParam = 0; currParam < argc; currParam++) {
64   if (strcmp(argv[currParam], "-r") == 0)
65      randSeed = atoi(argv[currParam+1]);} srand(randSeed);
66   DeployGlowworms(W);
67   for (int i = 0; i < n; i++) {Lc[i] = 5; Rd[i] = r;}
68   for (int t = 0; t < IterMax; t++ ) {
69   UpdateLuciferin(); FindNeighbors(); FindProbabilities();
70   SelectLeader(); Move(); UpdateNeighborhood(); }
71   for(int i = 0; i < n; i++) {
72    outFile << i << ' ' << X[i][0] << ' ' << X[i][1] << "\n";}
73   outFile.close(); exit(0); return 0;
74 }
```

Listing 14.2
Source-code of GSO in C++.

14.4 Step-by-step numerical example of GSO algorithm

Let us assume that we want to maximize the two-dimensional multimodal function given in (14.6), which has three peaks at $(-0.0093, 1.5814)$, $(1.2857, -0.0048)$, and $(-0.46, -0.6292)$. As mentioned earlier, GSO attempts to capture all the function peaks. The result of applying the GSO algorithm is shown in Fig. 14.1, in which 100 glowworms move from their initial locations to one of the peak locations. The steps of the algorithm are shown below:

1. The GSO algorithm parameters are determined: $n = 100$, $r_s = 3$, $\gamma = 0.6$, $\rho = 0.4$, $\beta = 0.08$, $n_t = 5$, $s = 0.03$, and $\ell_0 = 5$. The initial decision range $r_d(0)$ is set to 1 for all glowworms (1/3 rd of r_s) to limit the number of initial neighbors of all glowworms.
2. A swarm of 100 glowworms $\{G_1, G_2, \ldots, G_{100}\}$ with a corresponding set of initial locations $A = A_{init}$ is created by randomly deploying all the glowworms in a 2-D space, where $-3 \leq A(i, j) \leq 3$, for all $i \in \{1, 2, \ldots 50\}$ and $j \in \{1, 2\}$. We illustrate the working of the algorithmic steps of GSO by tracking how the state of one of the

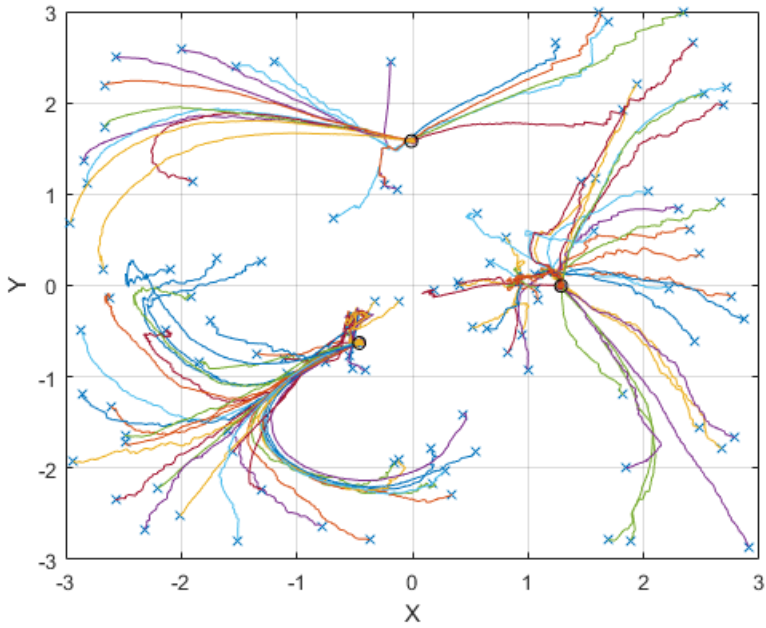

FIGURE 14.1
Paths traversed by all 100 glowworms from their initial locations to one of the peak locations.

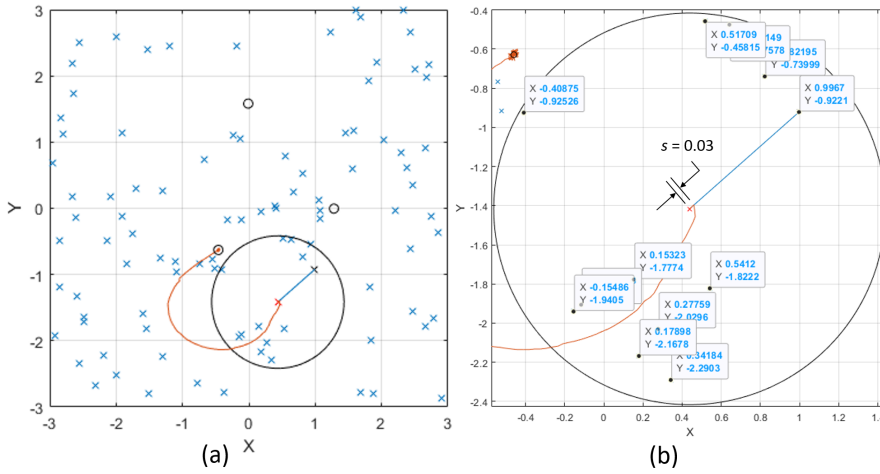

FIGURE 14.2
(a) Initial state (iteration $i = 0$) of the glowworm G_{45}: decision range $r_d(45, 0) = 1$, number of neighbors $Nbr(45, 0) = 11$, and the selected leader glowworm G_{31}. The overall path of G_{45} from its initial location to one of the peaks and the initial direction of movement toward the selected leader are also shown. (b) Zoom-in view of the initial state of G_{45} to show the locations of all its neighbors locating within an adaptive range of $r_d(45, 0) = 1$. Note that G_{45} makes a movement of step-size $s = 0.03$ toward its selected leader G_{31}.

glowworms G_{45} evolves as a function of iterations. Figure 14.2(a) shows the initial locations of all the 100 glowworms and the initial state (at iteration $i = 0$) of the glowworm G_{45}. For illustration purposes, the state of G_{45} at iterations $i = 0$, 1, 2, 5, 50, 100, 150, 200, and 249 are tabulated as shown in Table 14.2. The state at each iteration i includes the following variables:

(a) Current location, $A(45, i)$
(b) Objective function value, $J(45, i)$
(c) Luciferin value, $\ell(45, i)$
(d) Adaptive range radius, $r_d(45, i)$
(e) Number of neighbors, $Nbr(45, i)$
(f) Leader glowworm, $L(45, i)$
(g) Leader location, $A(L(45, i), i)$
(h) Updated location, $A(45, i + 1)$
(i) Updated adaptive range radius, $\ell(45, i + 1)$

3. The initial objective function value associated with each glowworm's position $J(i, 0)$, $i = 1, 2, \ldots, 100$ is computed. For example, the

TABLE 14.2
The state of glowworm G_{45} at different iterations.

i	$A(i)$	$J(i)$	$\ell(i)$	$r_d(i)$	$Nbr(i)$	$L(i)$	$A(i)$	$A(L(i))$	$r_d(i+1)$
0	(0.438, -1.416)	-5.682	-0.409	1	11	31	(0.997, -0.922)	(0.460, -1.396)	0.52
1	(0.459, -1.396)	-5.506	-3.549	0.52	1	77	(0.529, -1.849)	(0.464, -1.425)	0.84
5	(0.456, -1.483)	-5.891	-7.590	0.92	7	11	(0.140, -2.190)	(0.443, -1.510)	0.76
50	(-0.621, -2.110)	-1.760	-2.883	0.52	5	11	(-1.005, -2.028)	(-0.651, -2.103)	0.52
100	(-1.054, -1.065)	1.573	2.234	0.44	7	11	(-0.803, -0.830)	(-1.032, -1.045)	0.28
149	(-0.489, -0.624)	3.766	5.656	0.4	29	7	(-0.459, -0.611)	(-0.460, -0.630)	0
150	(-0.460, -0.630)	3.777	5.660	0	0	–	–	(-0.460, -0.630)	0.4
151	(-0.460, -0.630)	3.777	5.662	0.4	6	100	(-0.467, -0.636)	(-0.484, -0.648)	0.32
200	(-0.469, -0.630)	3.776	5.663	1.92	0	–	–	(-0.469, -0.630)	2.32
248	(-0.445, -0.626)	3.774	5.661	0.16	6	79	(-0.458, -0.625)	(-0.475, -0.625)	0.08
249	(-0.475, -0.625)	3.773	5.661	0.08	14	10	(-0.470, -0.647)	–	–

glowworm G_{45} is located at $A(45, 0) = (0.4378, -1.4164)$. The objective function value at this location is $J(45, 0) = -5.6816$.

4. The objective function values are used to then compute the luciferin values of each glowworm using (14.1). The luciferin value of G_{45} is $\ell(45, 0) = -0.409$.

5. The neighbors of each glowworm are determined by using (14.1.3). For example, G_{45} has twelve glowworms within an adaptive range $r_d(45, 0) = 1$: G_{86}, G_{91}, G_{34}, G_{31}, G_{77}, G_{15}, G_{54}, G_{11}, G_7, G_{23}, G_{18}, and G_{87}. Their locations are shown in Fig. 14.2(b). Note that a glowworm k is classified as a neighbor for glowworm j, if it is located within a circular neighborhood defined by the current adaptive range $r_d(j, i)$ and if $\ell(k, i) > \ell(j, i)$ as well. The luciferin values of all the twelve glowworms are shown in Table 14.3. Note from the table that except for one glowworm G_7 ($\ell(7, 0) < \ell(45)$), all other eleven glowworms have a luciferin value more than that of G_{45}. Therefore, G_{45} has eleven neighbors at the initial iteration.

6. For each glowworm, the probability of selecting to move toward each of its neighbors is computed by using (14.2). A leader-neighbor toward which movement will be made is determined using these probabilities in a roulette-wheel fashion. For G_{45}, the probabilities computed for all the eleven neighbors found in the previous step are shown in Table 14.3 and G_{31} is selected as a leader. Since the

TABLE 14.3
Luciferin and probability values of glowworms within a adaptive range $r_d(45, 0)$ of 1 unit of glowworm G_{45}'s location at the first iteration. For comparison purpose, the luciferin value G_{45} is provided here: $\ell(45, 0)$ = -0.409.

G_i	G_{86}	G_{91}	G_{34}	G_{31}	G_{77}	G_{15}	G_{54}	G_{11}
$\ell(i, 0)$	3.278	3.459	3.321	3.116	-0.099	1.357	0.188	0.767
$P(i, 0)$	0.149	0.157	0.151	0.143	0.013	0.072	0.024	0.048
G_i	G_7	G_{23}	G_{18}	G_{87}				
$\ell(i, 0)$	-0.735	0.036	0.237	4.512				
$P(i, 0)$	–	0.018	0.026	0.199				

leader selection is probabilistic, the chosen leader was not necessarily the one with highest luciferin, and hence the one with highest probability (G_{87}, $\ell(87, 0) = 4.512$, $P(87, 0) = 0.199$), but G_{31} with a relatively less luciferin ($\ell(31, 0) = 3.116$) and associated probability ($P(31, 0) = 0.143$).

7. Each glowworm moves toward the selected neighbor by using (14.4). For example, G_{45} makes a movement of step-size $s = 0.03$ toward G_{31} and stops at a new location (0.4596, -1.3958) (Refer to Fig. 14.2(b)).

8. The adaptive range of each glowworm gets updated by using (14.5). For example, since the actual number of neighbors of G_{45} ($Nbr(45, 0) = 11$) is more than the desired number of neighbors for any glowworm ($n_t = 5$), the update rule (14.5) causes the adaptive range of G_{45} to shrink from $r_d(45, 0) = 1$ to $r_d(45, 1) = 0.52$.

9. In the second iteration, all the steps 3 - 8 are executed to update the states of all the glowworms. For example, the objective function value at G_{45}'s new location is $J(45, 1)$ = -5.506. The corresponding luciferin value $\ell(45, 1)$ = -3.549. Figure 14.3 shows the updated locations of all the glowworms and the state of G_{45} at the second iteration. There are two glowworms G_7 and G_{77}) within G_{45}'s current adaptive range of 0.52 units. Their luciferin values are ranked as $\ell(7, 1)(= -4.116) < \ell(45, 1)(= -3.549) < \ell(77)(= -3.117)$. Therefore, G_{77} is the lone neighbor of G_{45} at the second iteration. G_{45} moves 0.03 units toward G_{77} and stops at a new location as shown in Table 14.2. Similarly, Fig. 14.4 shows the updated locations of all the glowworms and the state of G_{45} at iteration $i = 5$.

10. The algorithm continues in this way until the end of iterations ($i_{final} = 249$). Note from Table 14.2 that the final location of G_{45} is $(-0.4749, -0.6246)$, which corresponds to one of the peaks $(-0.46, -0.6292)$ within a tolerance of 0.02 units (the distance between the two locations is 0.0156 units).

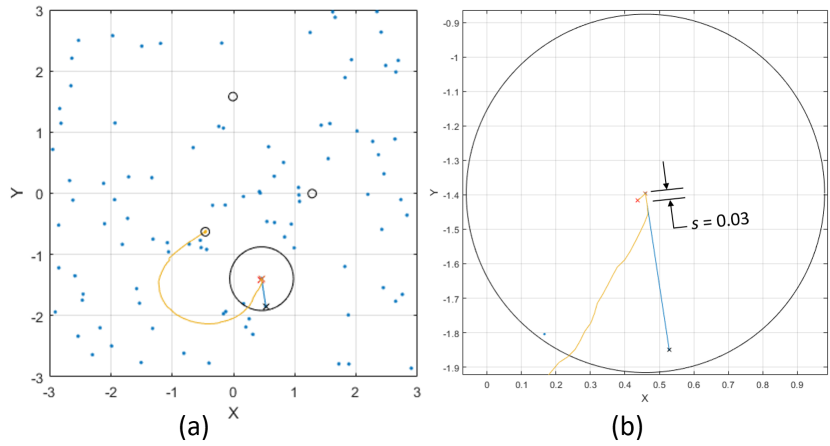

FIGURE 14.3
(a) State of the glowworm G_{45} at the second iteration ($i = 1$): decision range $r_d(45, 0) = 0.52$, number of neighbors $Nbr(45, 0) = 1$, and the newly selected leader glowworm G_{77}. The overall path of G_{45} from its initial location to one of the peaks and the direction of movement toward the selected leader are also shown. (b) Zoom-in view of the state of G_{45} to show its neighbors locating within an adaptive range of $r_d(45, 0) = 0.52$. Note that G_{45} makes a movement of step-size $s = 0.03$ toward its selected leader G_{77}.

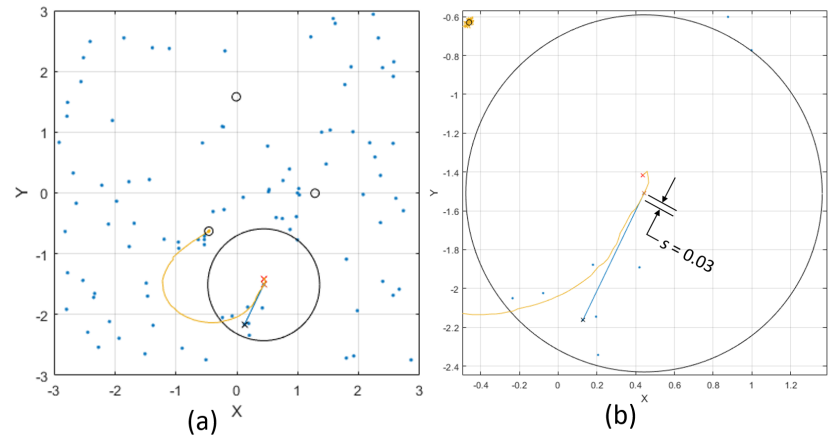

FIGURE 14.4
(a) State of the glowworm G_{45} at iteration $i = 5$: decision range $r_d(45, 0) = 0.92$, number of neighbors $Nbr(45, 0) = 7$, and the selected leader glowworm G_{11}. The overall path of G_{45} from its initial location to one of the peaks and the direction of movement toward the selected leader are also shown. (b) Zoom-in view of the state of G_{45} at iteration $i = 5$. Note that G_{45} makes a movement of step-size $s = 0.03$ toward its selected leader G_{11}.

A few observations are made to provide an insight into situations when a glowworm becomes leaderless. Note from Table 14.2 that the glowworm G_{45} has no leaders at iteration $i = 150$. This can be attributed to the fact that its adaptive range $r_d(45, 150)$ is zero, and hence it has zero neighbors. In order to find out why the range became zero, the previous state of G_{45} was analyzed. Note that the glowworm has a larger number of neighbors than the desired number of neighbors ($29 \gg n_t = 5$) at iteration $i = 149$ within an adaptive range $r_d(45, 149)$ of 0.4 units. This causes the range update rule (14.5) to shrink the adaptive range steeply. The reason for G_{45} having a high number of neighbors is due to the fact that it is already very close to the peak location $(-0.46, -0.6292)$ at iteration $i = 149$, where glowworms get crowded as increasingly more numbers of them move toward one of the three peaks. Similarly, G_{45} becomes leaderless at iteration $i = 200$, although it has a non-zero adaptive range of $r_d(45, 200) = 1.92$. This can be attributed to the fact that G_{45} has the highest within the adaptive range, and hence has no neighbors.

14.5 Conclusions

This chapter presented glowworm swarm optimization (GSO) that achieves simultaneous capture of multiple optima of multimodal functions and shows its special suitability for multiple signal source localization tasks. The underlying ideas behind the GSO technique were presented and the steps of the algorithm were illustrated using a numerical example involving simultaneous computation of multiple peaks of a multimodal function.

References

1. K.N. Kaipa and D. Ghose. Detection of multiple source locations using a glowworm metaphor with applications to collective robotics. In *Proceedings of IEEE Swarm Intelligence Symposium*, Pasadena, California, June 8−10, 2005, pp. 84−91.
2. K.N. Kaipa and D. Ghose. Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. *Swarm Intelligence*, Vol. 3, No. 2, pp. 87−124, 2009.
3. K. N. Kaipa and D. Ghose. *Glowworm Swarm Optimization: Theory, Algorithms, and Applications, Studies in Computational Intelligence*, Vol. 698, Springer-Verlag, 2017.

4. X. S. Yang. Firefly algorithms for multimodal optimization, In: Watanabe O., Zeugmann T. (eds) *Stochastic Algorithms: Foundations and Applications. SAGA 2009. Lecture Notes in Computer Science*, Vol. 5792. Springer, Berlin, Heidelberg
5. R. Sikora and M.J. Shaw. A double-layered learning approach to acquiring rules for classification: Integrating genetic algorithms with similarity-based learning. *ORSA Journal on Computing*, Vol. 6, No. 2, pp. 174−187, 1994.
6. J. Horn, D.E Goldberg, and K. Deb. Implicit niching in a learning classifier system: Nature's way. *Evolutionary Computation*, Vol. 2, No. 1, pp. 37−66, 1994.
7. K. McGill and S. Taylor. Robot algorithms for localization of multiple emission sources. *ACM Computing Surveys*, 43, 3, Article 15, 25 pages, 2011.