AUSTRALIAN RESEARCH COUNCIL CENTRE OF EXCELLENCE FOR MATHEMATICAL AND STATISTICAL FRONTIERS

Advanced Sampling and Exploration Matlab Competition

Dirk P. Kroese and Slava Vaisman

School of Mathematics and Physics
The University of Queensland
Australia
http://acems.smp.uq.edu.au/

1 The Problem

Imagine you are located in the center of a unit disc that contains obstacles. You cannot see what is inside the disc, but you can activate a sensor (oracle) and learn if an obstacle exists at any given coordinates.

The challenge is to find a feasible path from the center to the outside boundary using the least number of oracle probes.

The "landscape" is defined as follows. Each obstacle is a disc with a fixed radius $r < 1$ and the centers of these discs form a homogeneous Poisson process on the unit disc. This process, which is defined by two parameters, namely $n \in \mathbb{N}$ and r, generates $N \sim \mathsf{Poiss}(n)$ discs having radius $r < 1$ uniformly at random on the unit disk. *However, the process generation is conditioned on the existence of a feasible path from the center to the disc boundary.* Figure 1 shows a typical outcome of the stochastic process (for $n = 100$ and $r = 0.1$) described above. The corresponding generation Matlab code is given in Section 5.1 (`landscapegeneration.m`).

The competition is targeted at all students associated with the *ARC Centre of Excellence for Mathematical and Statistical Frontiers* (ACEMS), but any interested student or group of students outside the centre can submit as well. The winner of the competition and the winning ACEMS entry will be announced at the

International Workshop on Monte Carlo Methods for Spatial Stochastic Systems

21-23 July 2015, Brisbane, Australia

The winning ACEMS entry will receive a $250 award. The deadline for submissions is 13 July, 2015. Please send your submission to `kroese@maths.uq.edu.au`. Accompanying Matlab code is available at `http://acems.smp.uq.edu.au/`.

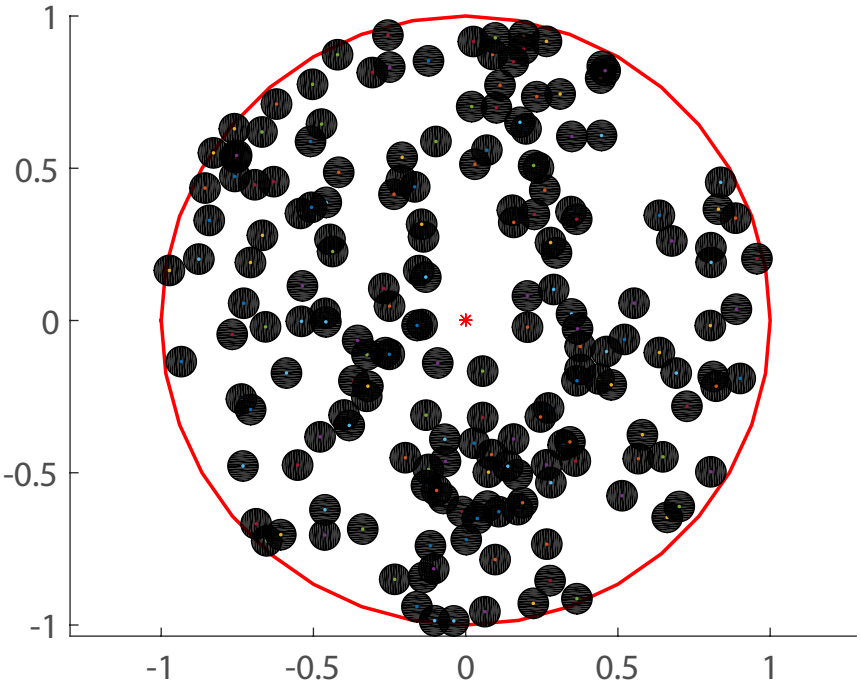


Figure 1: A random landscape on the unit disc.

2 The Challenge Rules

The participant will provide a single Matlab function that implements the path sampling algorithm. Please give your function a distinctive name. This function should have the following format:

$$\texttt{path} = \texttt{functionname}(\texttt{oracle}).$$

The function outputs a path that starts from the origin (0,0) and ends at or outside the unit circle, and takes as an input argument an oracle that is provided by us. Your `path` should be some $k \times 2$ matrix of coordinates. For example, `path = [0,0; 0.5,0.5; 1,0]` constitutes a path.

Remark 2.1 (The Oracle) In your function testing should use the oracle that is provided in Section 5.2. However, use it as an external function, that is, do not copy the oracle into your function (`path = functionname(oracle)` file).

2.1 The Score

Having in mind that the required algorithm outputs a path from the center of the unit disk to the outside boundary, the score of the algorithm will be calculated as follows. The check program will calculate (1) the **path feasibility**, (2) the number of oracle calls used, and (3) the path length. An infeasible path will

get a zero score. An implementation that will provide a feasible path and will use the smallest number of oracle calls will receive a better score. An additional bonus will be given to the implementation that produces (for a specific n and r) on average the shortest feasible path on multiple test instances. See Section 4 for more details on the scoring.

3 Basic Algorithms

We next provide two basic example algorithms for solving the above problem. These methods essentially implement a random walk on the feasible space. Algorithm 3.1 generates a random direction $\mathbf{v}$ at each step and performs an ε sized move (in $\mathbf{v}$ direction). Algorithm 3.2 chooses a random direction and continues until an obstacle is met without changing it. Both algorithms continue until the boundary of the unit disk is reached.

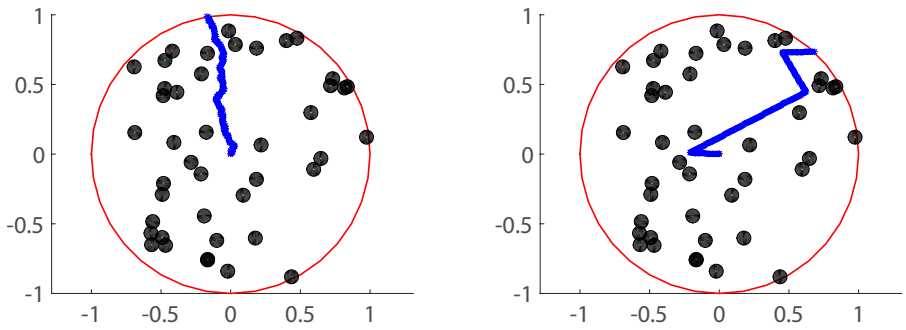
Algorithm 3.1 (Basic Algorithm 1) *Given an* `oracle(x)` *function that outputs* 1 *if* $\mathbf{x}$ *is feasible and* 0 *otherwise, execute the following steps.*

1. *Set* $\mathbf{X} \leftarrow (0,0)$, $\varepsilon \leftarrow 0.1$, *and* $\texttt{path} \leftarrow \{\mathbf{X}\}$.
2. *While* $\mathbf{X} \in$ *unit disk, do:*
   (a) *Generate a random direction* $\mathbf{v} = (v_x, v_y)$.
   (b) *Set* $\mathbf{X}' \leftarrow \mathbf{X} + \varepsilon\,\mathbf{v}$.
   (c) *If* $\texttt{oracle}(\mathbf{X}') = 1$, *set* $\mathbf{X} \leftarrow \mathbf{X}'$ *and* $\texttt{path} \leftarrow \texttt{path} \cup \mathbf{X}$.
3. *Output* `path`.

Algorithm 3.2 (Basic Algorithm 2) *Given an* `oracle(x)` *function that outputs* 1 *if* $\mathbf{x}$ *is feasible and* 0 *otherwise, execute the following steps.*

1. *Set* $\mathbf{X} \leftarrow (0,0)$, $\varepsilon \leftarrow 0.1$, *and* $\texttt{path} \leftarrow \{\mathbf{X}\}$.
2. *Generate a random direction* $\mathbf{v} = (v_x, v_y)$.
3. *While* $\mathbf{X} \in$ *unit disk, do:*
   (a) *Set* $\mathbf{X}' \leftarrow \mathbf{X} + \varepsilon\,\mathbf{v}$.
   (b) *If* $\texttt{oracle}(\mathbf{X}') = 0$, *generate a new random direction* $\mathbf{v}$, *else set* $\mathbf{X} \leftarrow \mathbf{X}'$ *and* $\texttt{path} \leftarrow \texttt{path} \cup \mathbf{X}$.
4. *Output* `path`.

The Matlab code of Algorithms 3.1 and 3.2 is given in Section 5.3, (`BasicAlgorithm1.m` and `BasicAlgorithm2.m`). Typical paths are given in Figure 2.



(a) Basic Algorithm 1 (b) Basic Algorithm 2

Figure 2: Typical runs of Algorithms 3.1 and 3.2

4 Testing and Scoring

We provide the test program (`testalgorithm.m`) that will be used for algorithm evaluation; see Section 5.4 for the code. The test program checks an ε-path feasibility in the following sense. Each segment of the path will be divided into ε-sized intervals. Then, the oracle will be applied to the endpoints of these intervals to determine their feasibility. For the precise definition of the ε-path feasibility, please see the code.

In addition to `testalgorithm.m` we provide four example landscapes. These landscapes will not be used in the final evaluation, but similar ones will be generated using the `landscapegeneration.m` function (conditional on there being a feasible path).

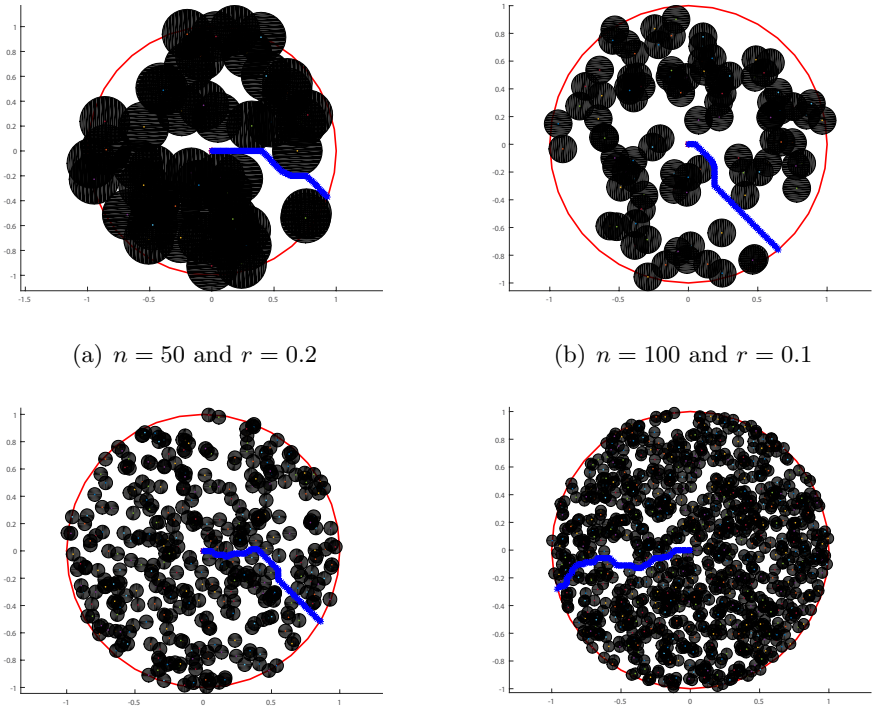
Below we present the result of a typical run of the test program on Algorithms 3.1 and 3.2.

```
>> testalgorithm
Basic Algorithm 1: Feasibility: 1 oracle calls: 202, path length: 1.75
Basic Algorithm 2: Feasibility: 1 oracle calls: 106, path length: 1.05
```

In the actual test we will use 5 instances for each of the four parameter choices in Figure 3.

Details of Scoring

For each of the 20 test landscapes, the algorithms will be ranked by number of oracle calls. The best 10 will be awarded points according to the "Eurovision" convention: 12, 10, 8, 7, ..., 1. For each of the 4 parameter choices there will be a bonus of 10 points awarded to the algorithm that provides on average the smallest feasible path length over the 5 instances for that parameter choice. The maximum possible total score is thus $20 \times 12 + 40 = 280$.



(a) $n = 50$ and $r = 0.2$

(b) $n = 100$ and $r = 0.1$

(c) $n = 400$ and $r = 0.05$

(d) $n = 1000$ and $r = 0.04$

Figure 3: Example of feasible paths for different models.

5 The Matlab Code

5.1 Landscape Generation

```
function [n rad N z] = landscapegeneration(n, rad)

N  = poissrnd(n);
r = randn(N,2);
z = r./repmat(sqrt(sum(r.^2,2)),1,2);
found = 0;
while ~found
    u = sqrt(rand(N,1));
    found = ~(sum(u < rad));
end
z = z.*repmat(u,1,2);
fig = figure;
hold on
plot([0,0],[0,0],'r*')
rectangle('Position',[-1,-1,2,2],'Curvature',[1,1], ...
'EdgeColor','r', 'LineWidth',2)
for i=1:N
    rectangle('Position',[z(i,1) - rad ,z(i,2) - rad,  2*rad,   2*rad
        ], ...
        'Curvature',[1,1],'FaceColor','k');
    plot(z(i,1),z(i,2),'.')
```

```
end
axis equal
hold off

end
```

5.2 The Oracle

```
function hit = oracle(x)

    % global oracle ussage counter0
    global call_to_oracle

    % load data
    M = dlmread('20_0.1.txt',',');
    rad = M(1,1);
    z =  M(2:size(M,1),:);

    N = size(z,1);
    s = z - repmat(x,N,1);
    norms = sqrt(sum(s.^2,2));
    hit = logical(sum(norms < rad));

    call_to_oracle = call_to_oracle+1;

end
```

5.3 Basic Random Walk Algorithms

```
function path = BasicAlgorithm1(oracle)
    path = [0 0];
    x = 0; y=0;

    epsilon = 0.1;

    while(IsOutOfDisk(x,y)==0)
       % generate next coordinate
       [v,u] = GetRandUnitVector();
       x_try = x+v*epsilon;
       y_try = y + u*epsilon;
       if(oracle([x_try,y_try])==0)
            % no hit
             path = [path; x_try y_try ];
             x = x_try; y = y_try;
       end
    end
end

function res = IsOutOfDisk(x,y)
    res = (x)^2 + (y)^2 >= 1;
end

function [x,y] = GetRandUnitVector()
    ang = rand*pi;
```

```
    x = cos(ang);
    y = sin(ang);
end
```

```
function path = BasicAlgorithm2(oracle)
    path = [0 0];
    x = 0; y=0;
    epsilon = 0.1;

    [v,u] = GetRandUnitVector();
    while(IsOutOfDisk(x,y)==0)
       % generate next coordinate
       x_try = x+v*epsilon;
       y_try = y + u*epsilon;
       if(oracle([x_try,y_try])==0)
            % no hit
             path = [path; x_try y_try ];
             x = x_try; y = y_try;
       else
           % change direction
            [v,u] = GetRandUnitVector();
       end
    end
end

function res = IsOutOfDisk(x,y)
    res = (x)^2 + (y)^2 >= 1;
end

function [x,y] = GetRandUnitVector()
    ang = rand*pi;
    x = cos(ang);
    y = sin(ang);
end
```

5.4 Algorithm Test Program

```
function testalgorithm

    global call_to_oracle

    eps = 0.01;

    alg1 = @BasicAlgorithm1;
    alg2 = @BasicAlgorithm2;

    call_to_oracle = 0;
    path1 = alg1(@oracle);
    [is_feasible1, total_length1] = CheckPathFeasibility(path1, eps);
    alg1_oracle_calls = call_to_oracle;

    call_to_oracle = 0;
    path2 = alg2(@oracle);
    [is_feasible2, total_length2] = CheckPathFeasibility(path2, eps);
    alg2_oracle_calls = call_to_oracle;
```

```
19
20      % print statistics for the solved problems
21      fprintf('Basic Algorithm 1: Feasibility: %d oracle calls: %d,
            path length: %.2f \n',is_feasible1, alg1_oracle_calls,
            total_length1);
22      fprintf('Basic Algorithm 2: Feasibility: %d oracle calls: %d,
            path length: %.2f \n',is_feasible2, alg2_oracle_calls,
            total_length2);
23  end
24
25  function [is_feasible, total_length] = CheckPathFeasibility(path, eps
        )
26      n = size(path,1);
27      total_length = Inf;
28      % verify that the path is outside the disk
29      if(0 == IsOutOfDisk(path(n,1),path(n,2)))
30          is_feasible = 0;
31      else
32          % verify that the path is epsilon feasible
33          x0 = 0; y0=0;
34          total_length = 0;
35          for i=2:n
36              x1 = path(i,1); y1 = path(i,2);
37              tmp = sqrt(  (x0-x1)^2 + (y0-y1)^2  );
38              total_length = total_length + tmp;
39
40              delta = floor(tmp/eps);
41              v_x = (x1-x0); v_y = (y1-y0);
42              v_x = v_x/sqrt(v_x^2+v_y^2); v_y = v_y/sqrt(v_x^2+v_y^2);
43              for j=0:delta-1
44                  tmp_x = x0+delta*eps*(v_x);
45                  tmp_y = y0+delta*eps*(v_y);
46                  if(1==oracle([tmp_x,tmp_y]))
47                      total_length = Inf;
48                      is_feasible = 0;
49                      return;
50                  end
51              end
52              x0 = x1; y0=y1;
53          end
54          is_feasible = 1;
55      end
56  end
57
58  function res = IsOutOfDisk(x,y)
59      res = (x)^2 + (y)^2 >= 1;
60  end
```