The Small-World Phenomenon: an Algorithmic Perspective

Abhishek Kamble Rohan Narde

Abstract

Small World Phenomenon is the principle that states that most of the people are linked by short chains of acquaintances. It emphasis on six degrees of separation, that is two unknown people are linked by 6 people of chain in between. Here we are trying to implement this paper by prof. Jon Kleinberg. We have implemented this algorithm in java for p=1,q=1 and for different conditions for each n in {25,50,100,200,300,400} and for each r in {0, .5, .7, 1, 2, 3, 5, 7}. We have calculated number of hops it takes between source and destination for each n and each r for 100 times to get the mean number of hops in each n and r. We are trying to prove here that our algorithm performs according to the analysis mentioned in the paper by estimating value of α.

1 Algorithm:

Step 1 : for each r in {0, .5, .7, 1, 2, 3, 5, 7}

Step 2. for each n in {25,50,100,200,300,400}

Step 2: form n*n grid of nodes.

Step 3: randomly select the source node and randomly select the destination node.

Step 4: Find the relative position between source S and destination d .

Step 5. Consider the shortest hop from source to destination

Step 6 Consider the longest hop from source to destination by consider reachability probability from source.

Step 7. Compare distance of longest hop and shortest hop with respect to destination choose the shortest hop by greedy approach .

Step 8. Increase hop count by 1.

Step 9. Source s = new discovered node

Step 10. Go to step 4 if (s ≠d)

Step 11. Print mean and variance after running 100 times for each n and r combination

2 Calculations:

Run the algorithm for each n in {25,50,100,200,300,400} and for each r in {0, .5, .7, 1, 2, 3, 5, 7} 100 times and calculate the mean and variance for each n and r please find the below observation table.

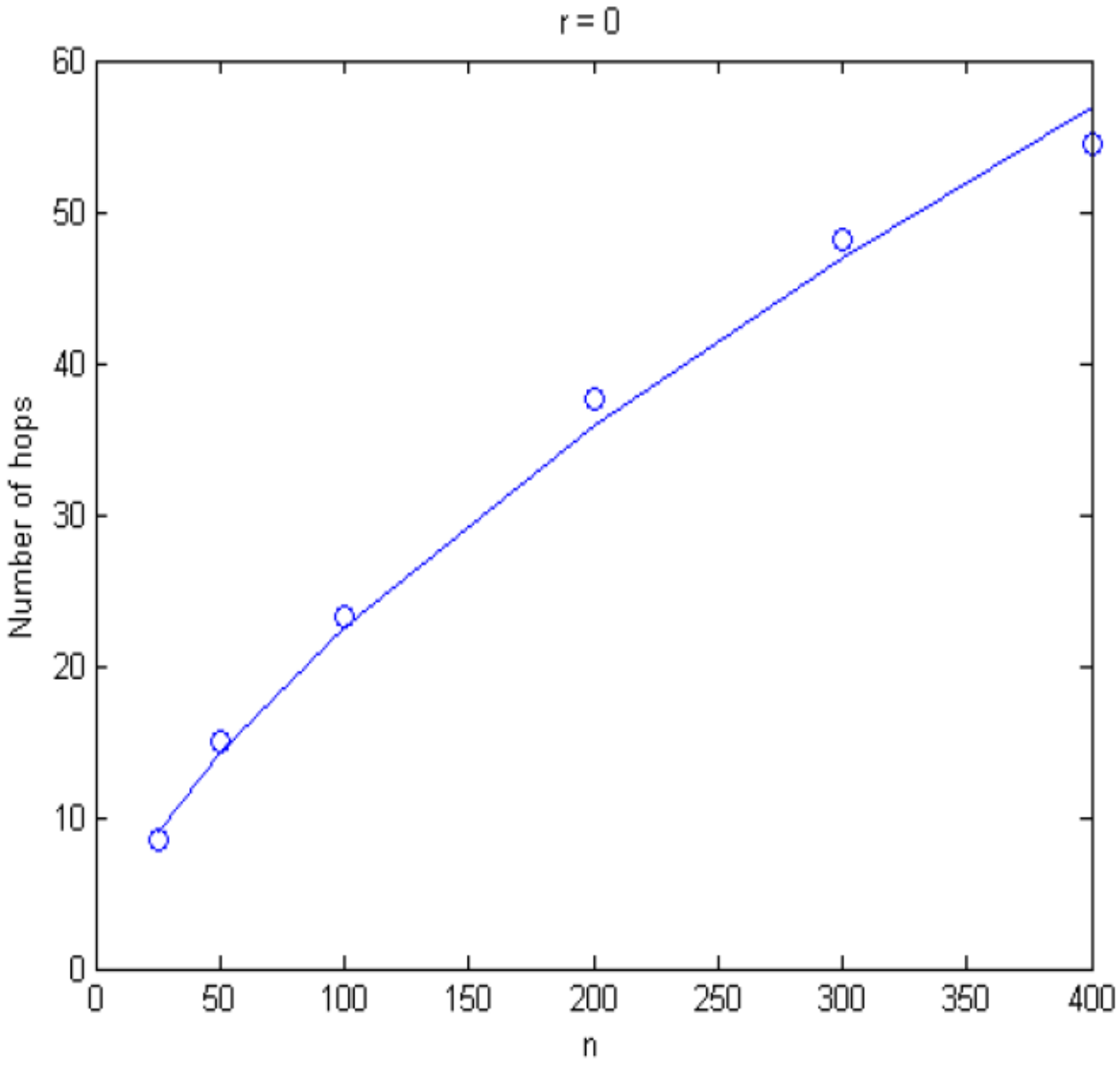
n/r	25	50	100	200	300	400
0	8.44	15.01	23.31	37.6	48.09	54.48
0.5	8.95	14.9	23.69	36.99	45.93	61.16
0.7	9.24	14.79	24.54	38.65	47.94	62.11
1	10.04	16.01	23.66	38.64	56.63	67.43
2	13.44	23.71	43.32	69.03	100.41	112.73
3	17.67	30.86	63.01	126.05	194.48	262.35
5	15.98	33.11	69.67	141.44	198.08	254.03
7	16.9	34.51	66.8	136.96	196.92	279.21

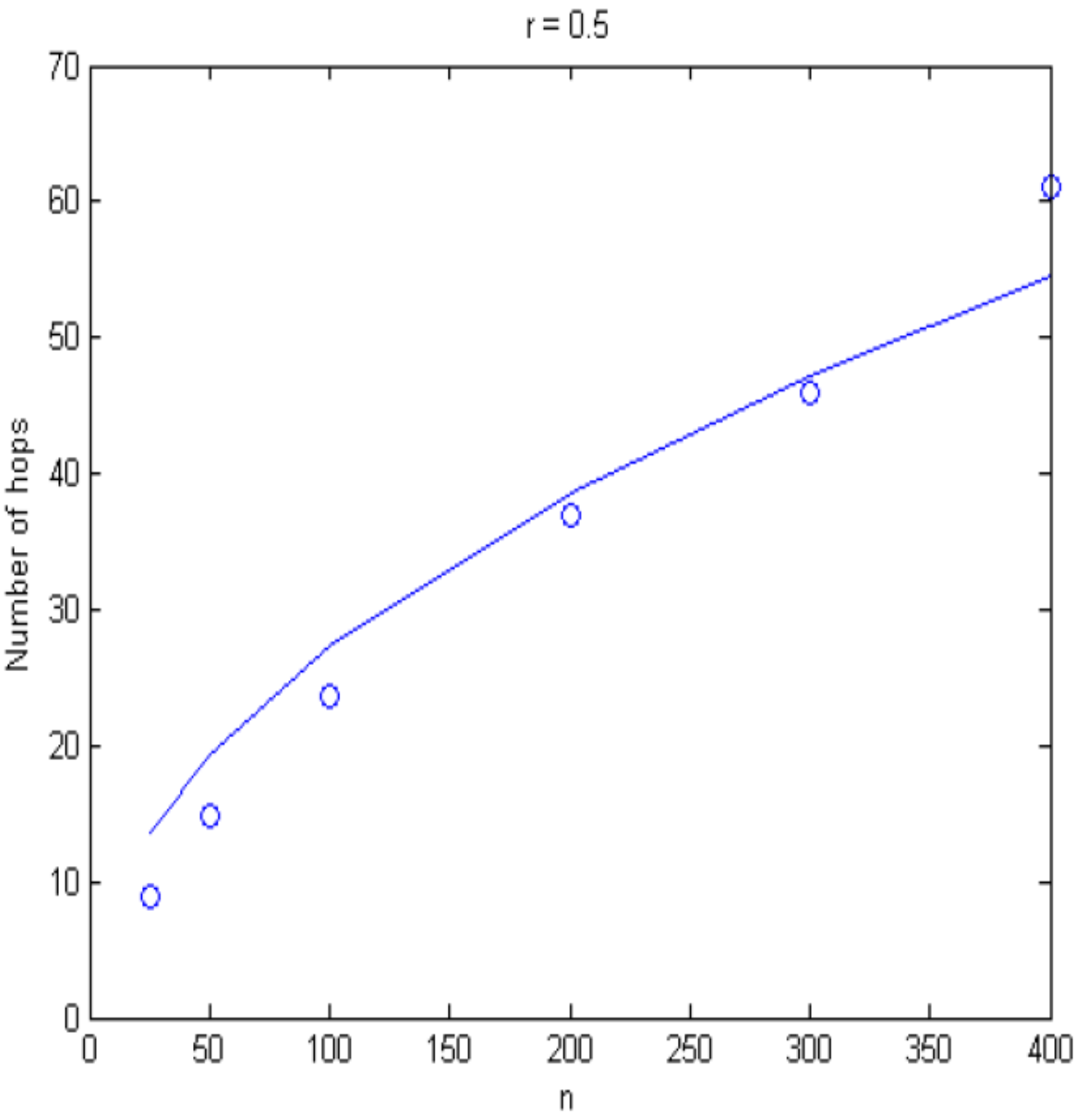
Table 1. Mean number of Hops

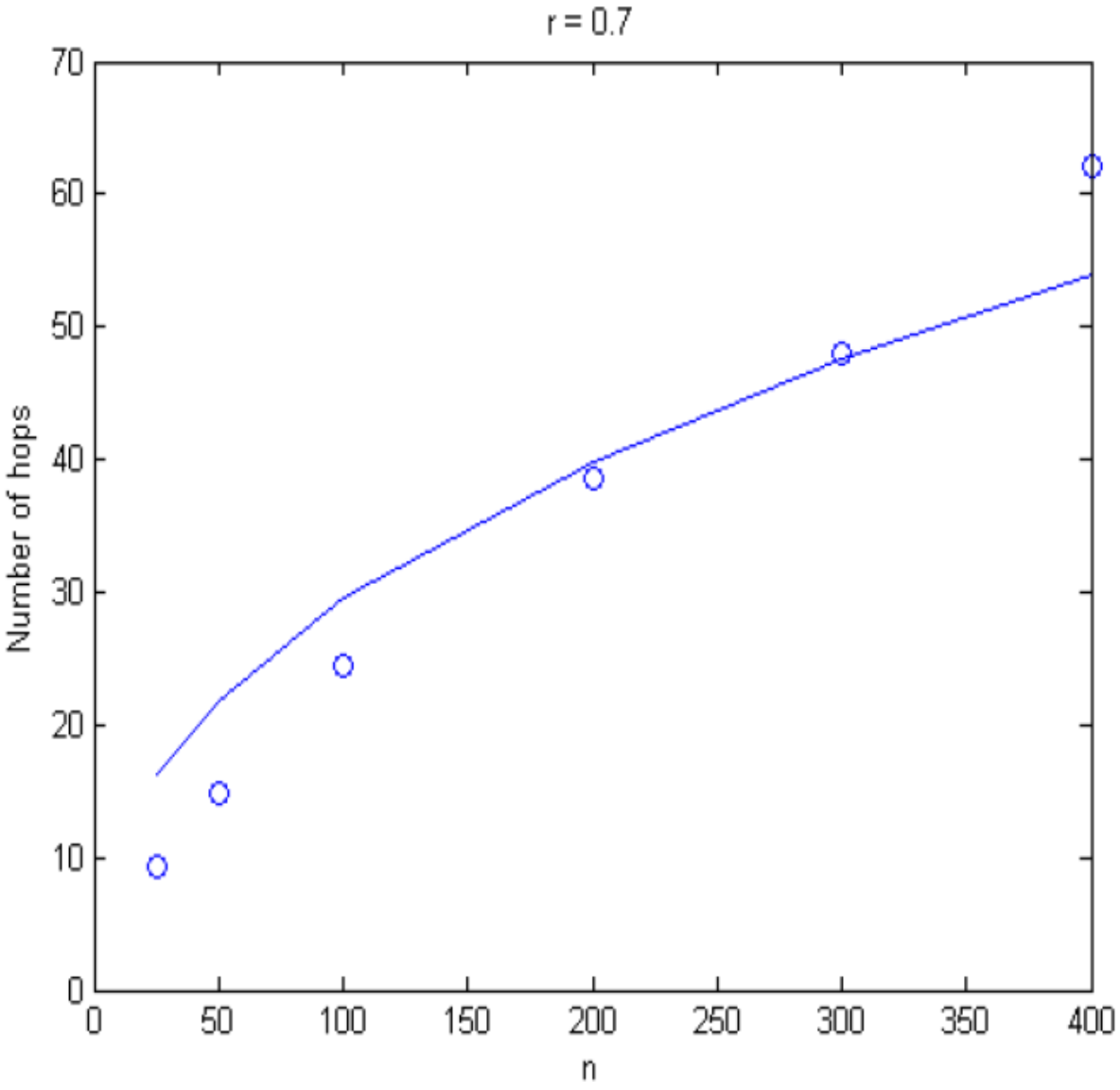
n/r	25	50	100	200	300	400
0	11.06	33.1	80.15	171.6	335.6	525.16
0.5	13.84	34.77	115.37	208.48	277.74	520.71
0.7	18.54	46.82	86.76	233.22	342.73	546.89
1	22.67	51	118.44	330.65	580.5	824.8
2	45.84	126.61	477.59	1412.449	2927.362	3433.8
3	70.84	253.37	1201.13	4218.36	11348.63	15811
5	70.39	321.95	1109.481	4752.286	10291.28	18013.13
7	80.26	276.12	973.04	5092	10467	15318

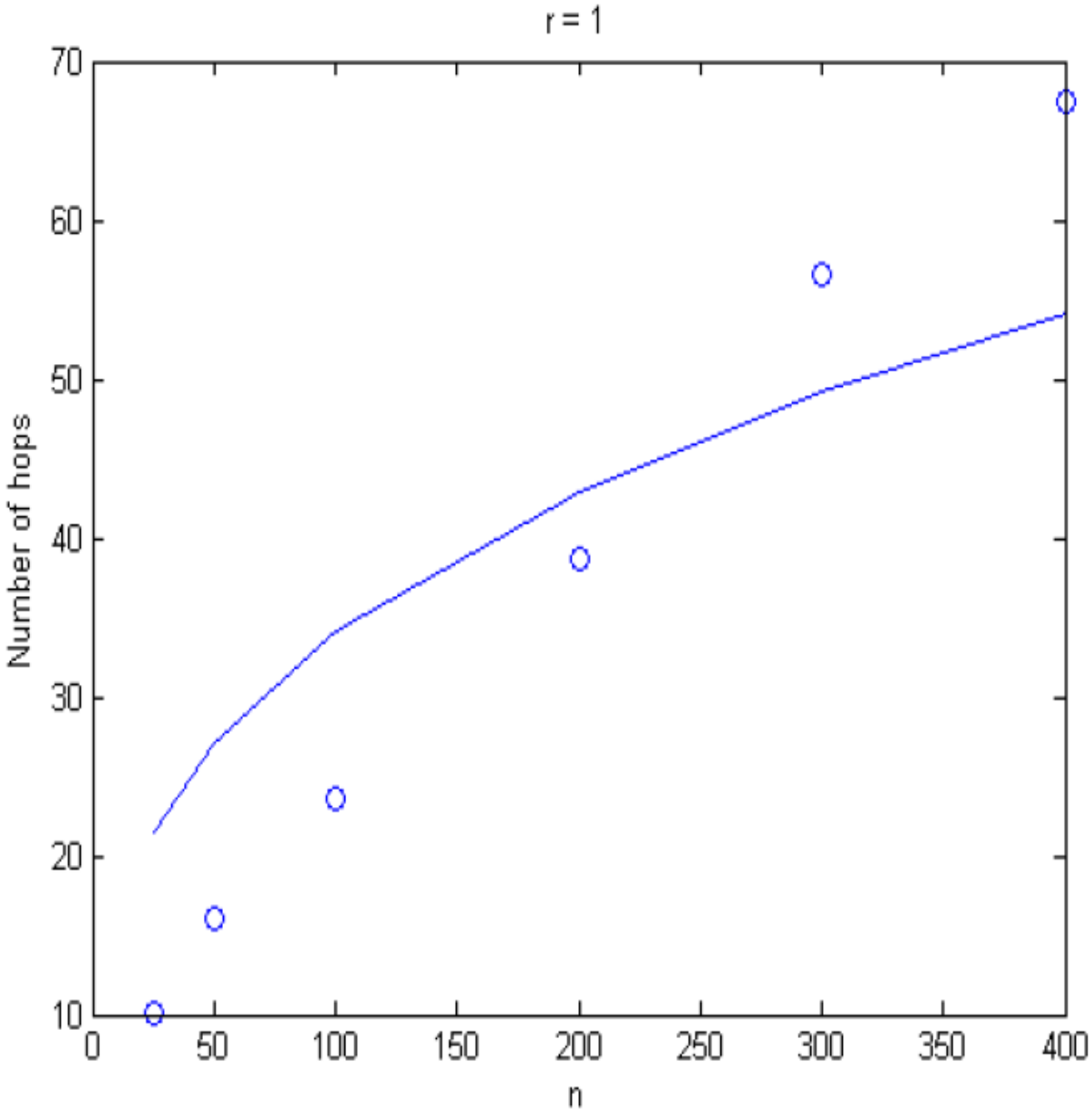
Table 2. Variance for number of Hops

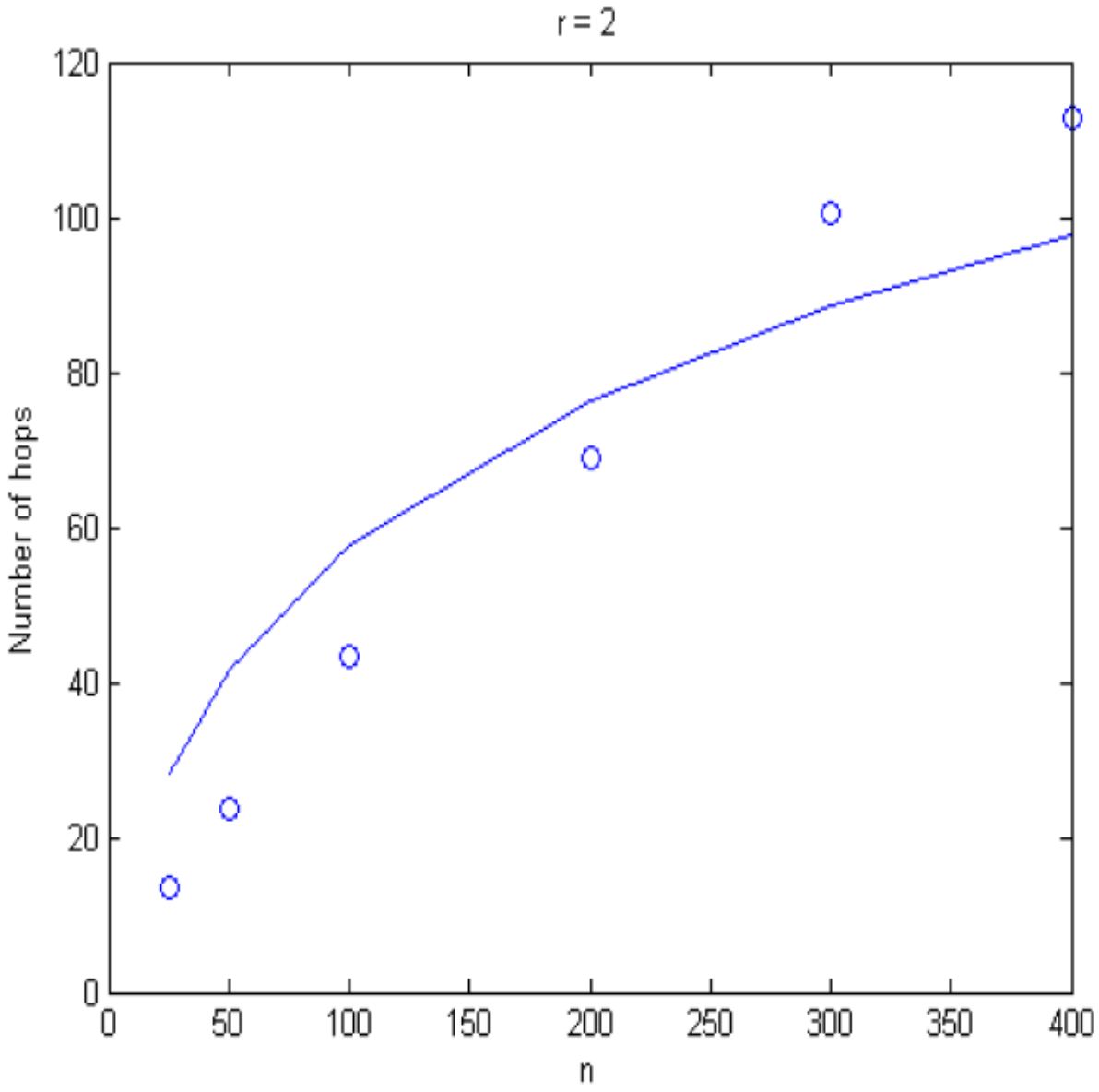
3. Charts, one for each choice of *r*, with *n* on the x-axis and the mean number of hops on the y-axis.

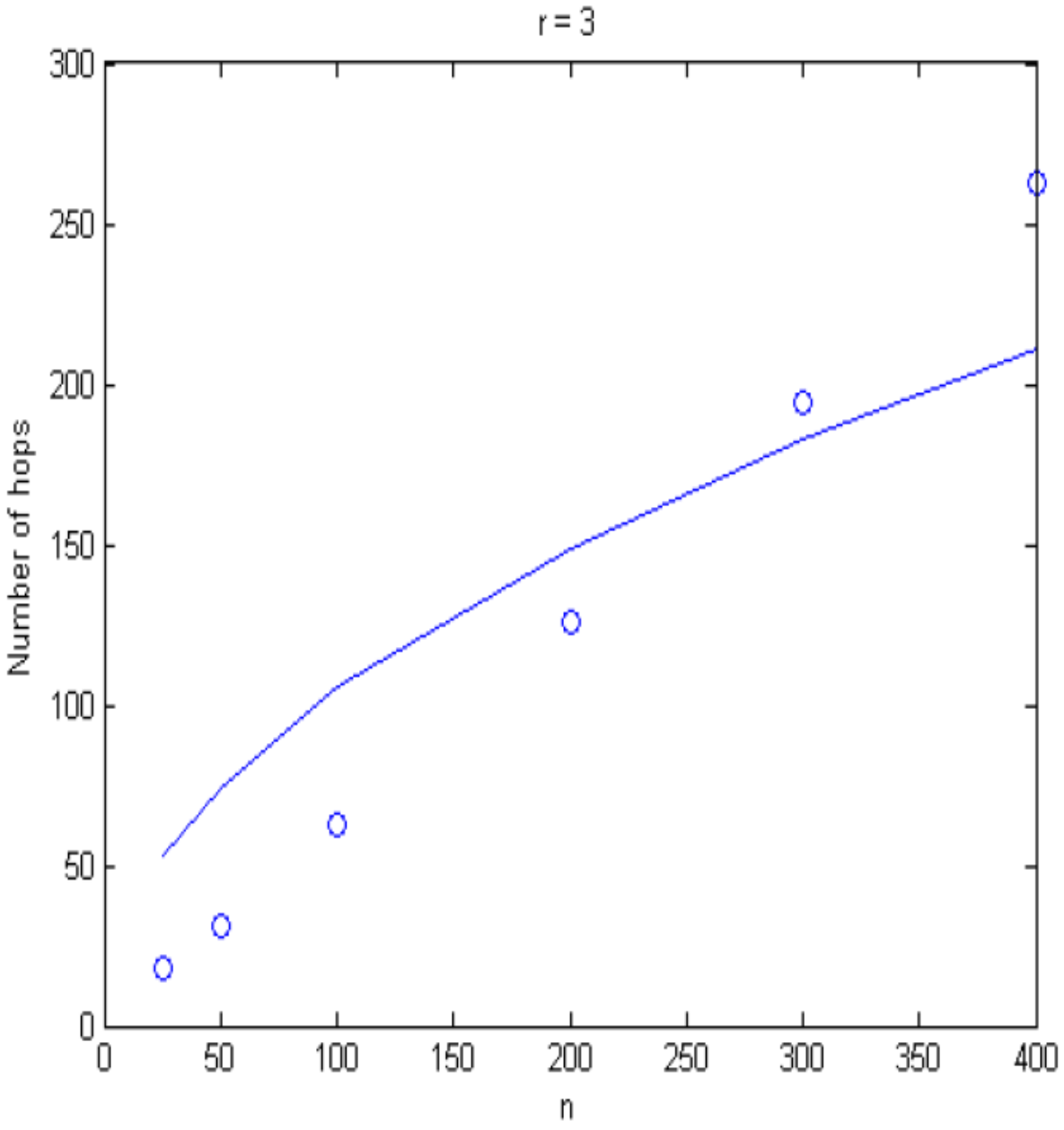


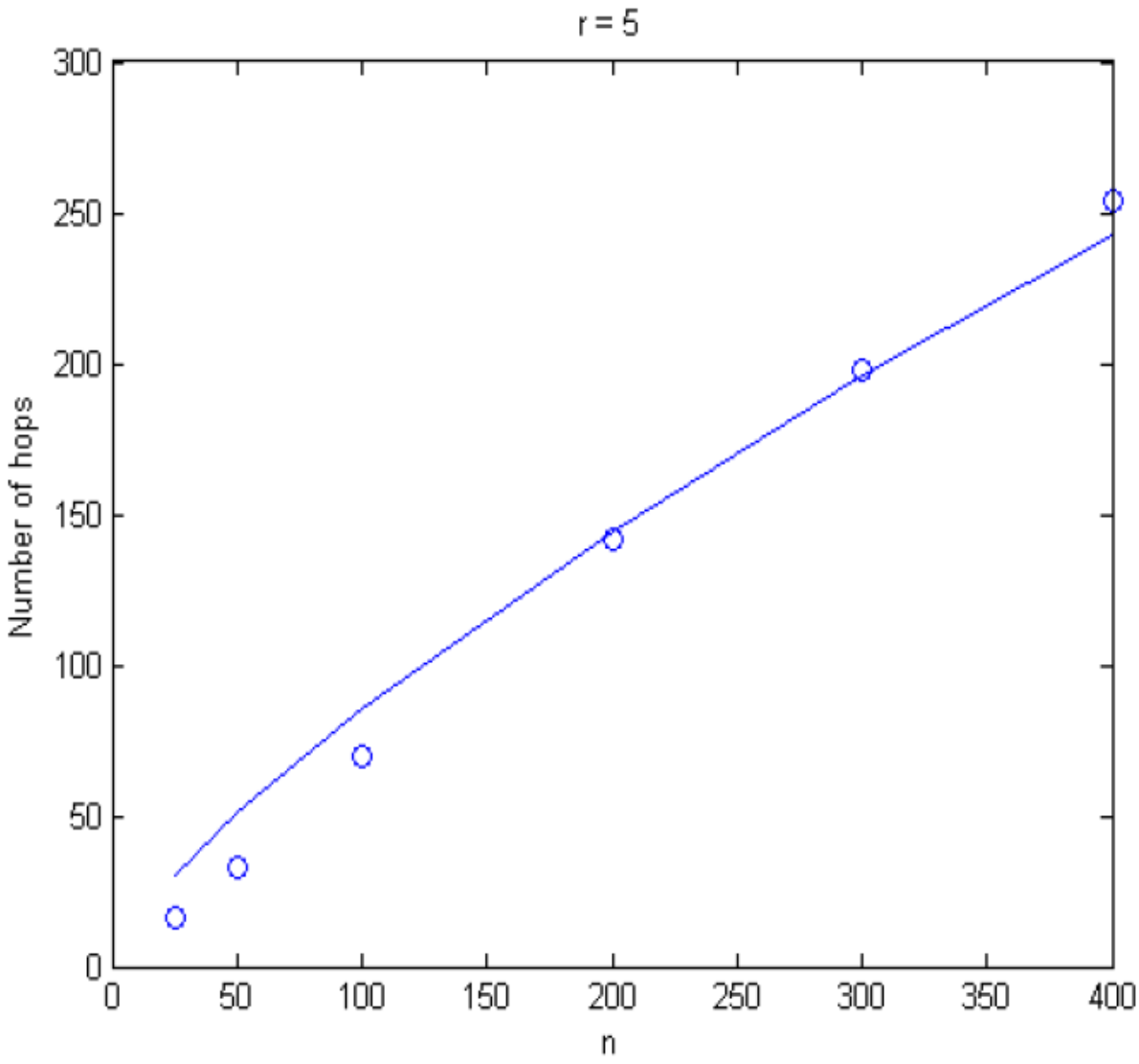

r = 0.5
Number of hops
n
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300
350
400

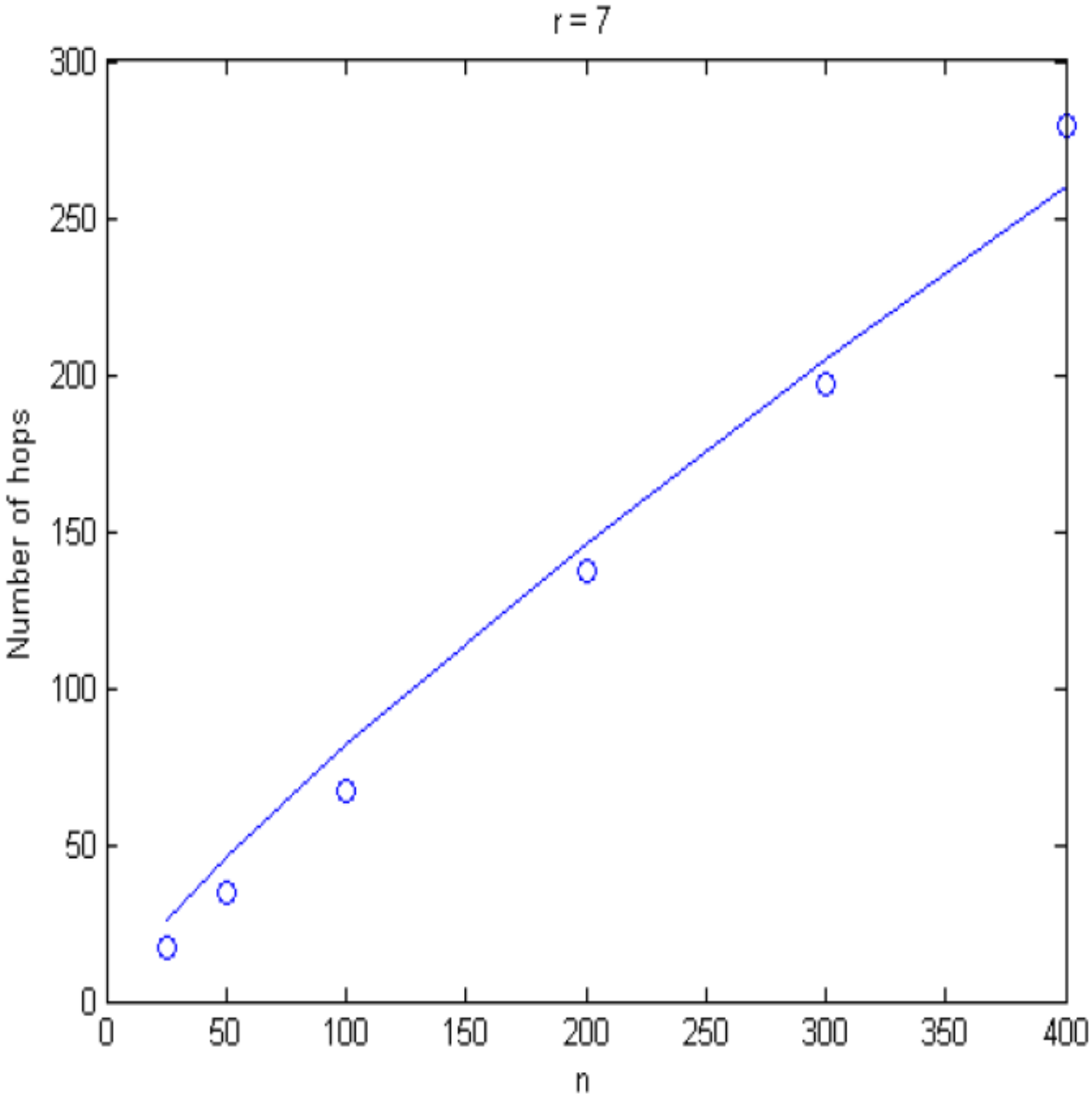

r = 0.7
Number of hops
n
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300
350
400


r = 1
Number of hops
n
70
60
50
40
30
20
10
0
50
100
150
200
250
300
350
400

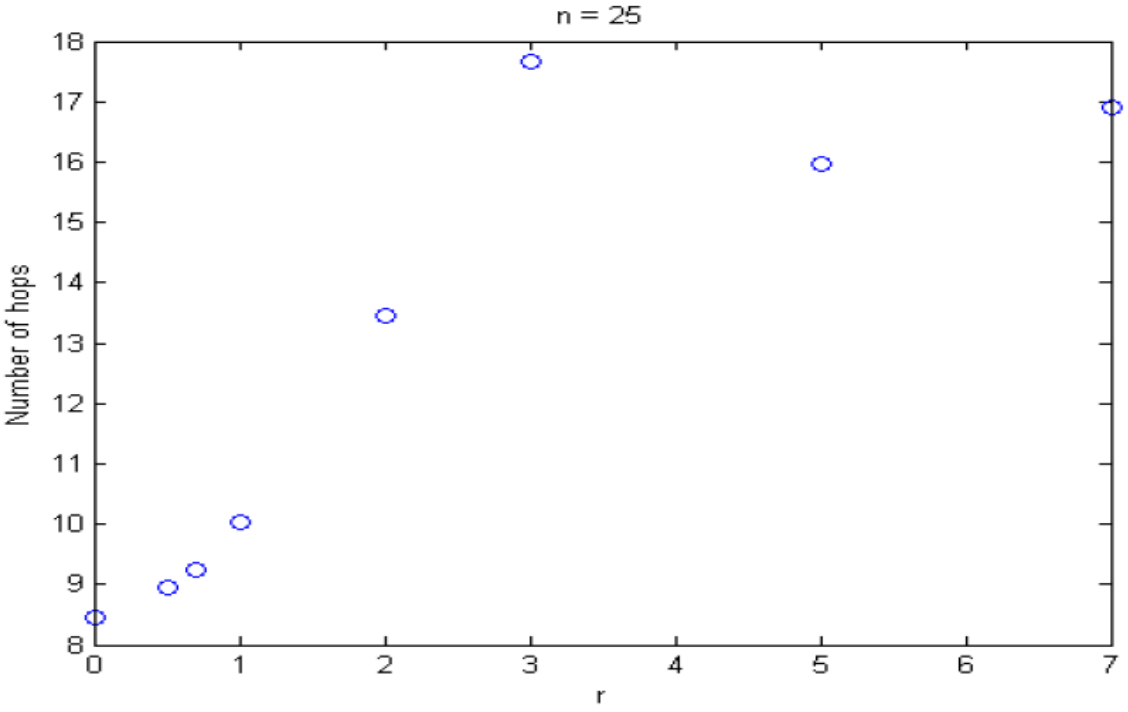

r = 2
Number of hops
n
0
20
40
60
80
100
120
0
50
100
150
200
250
300
350
400

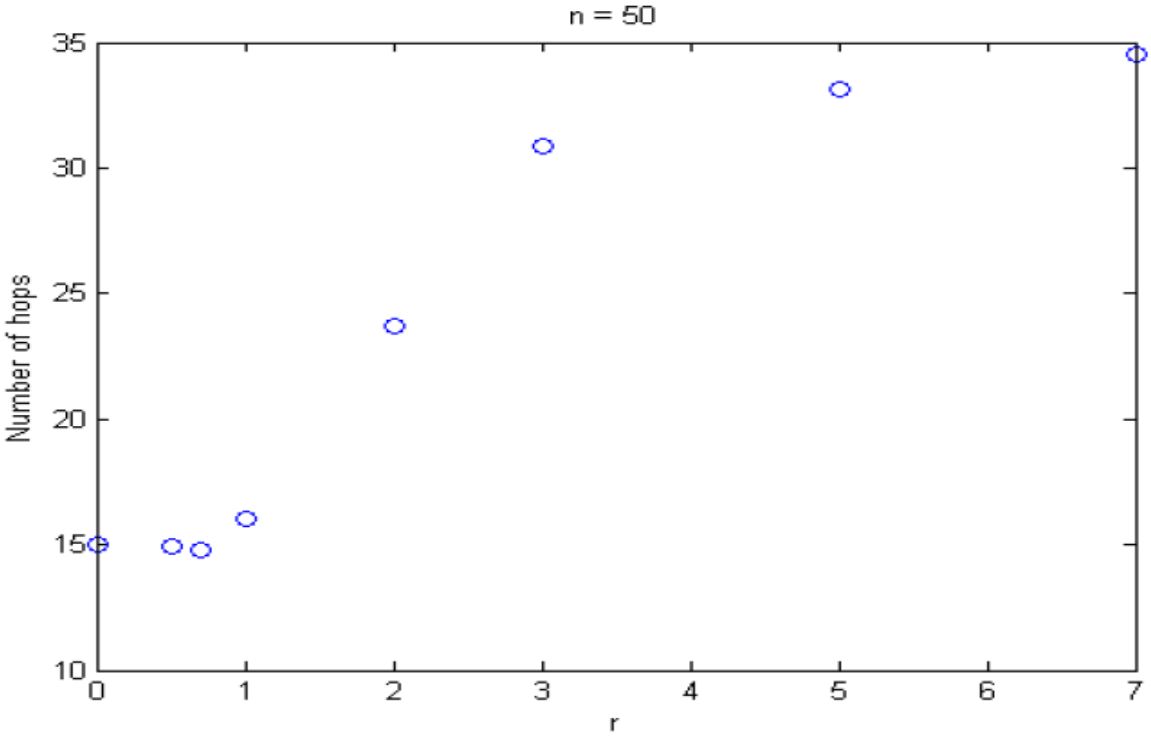

r = 3
Number of hops
n
0
50
100
150
200
250
300
350
400

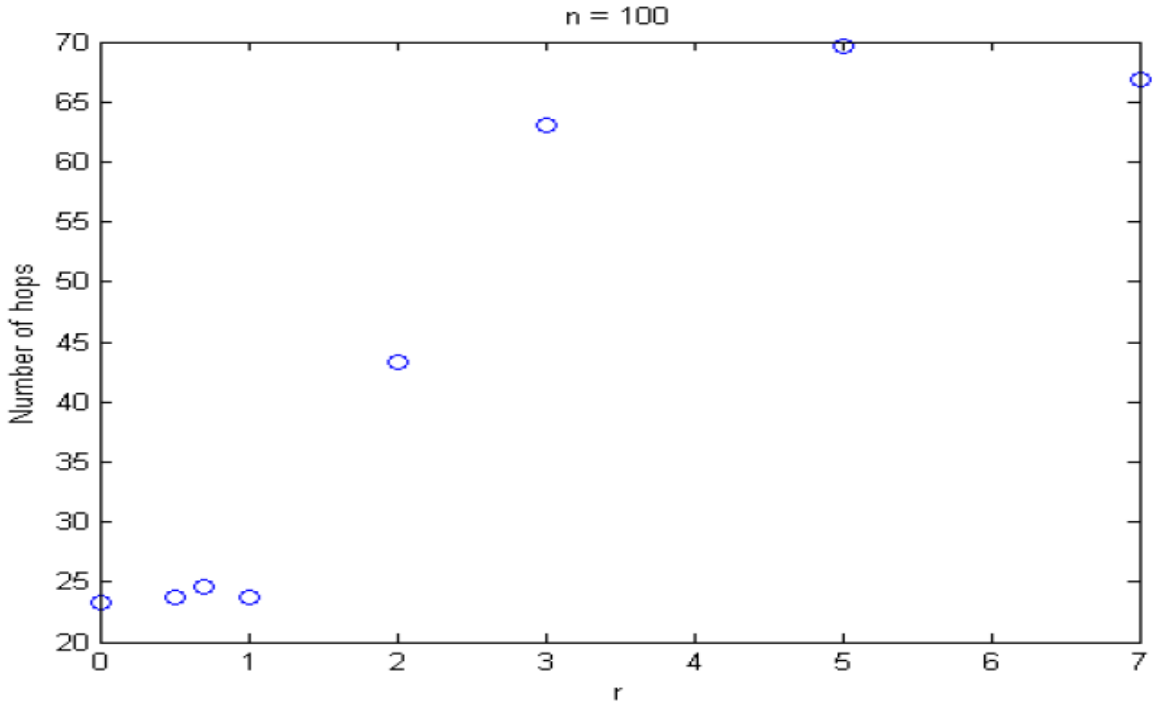

r = 5
Number of hops
n
0
50
100
150
200
250
300
350
400

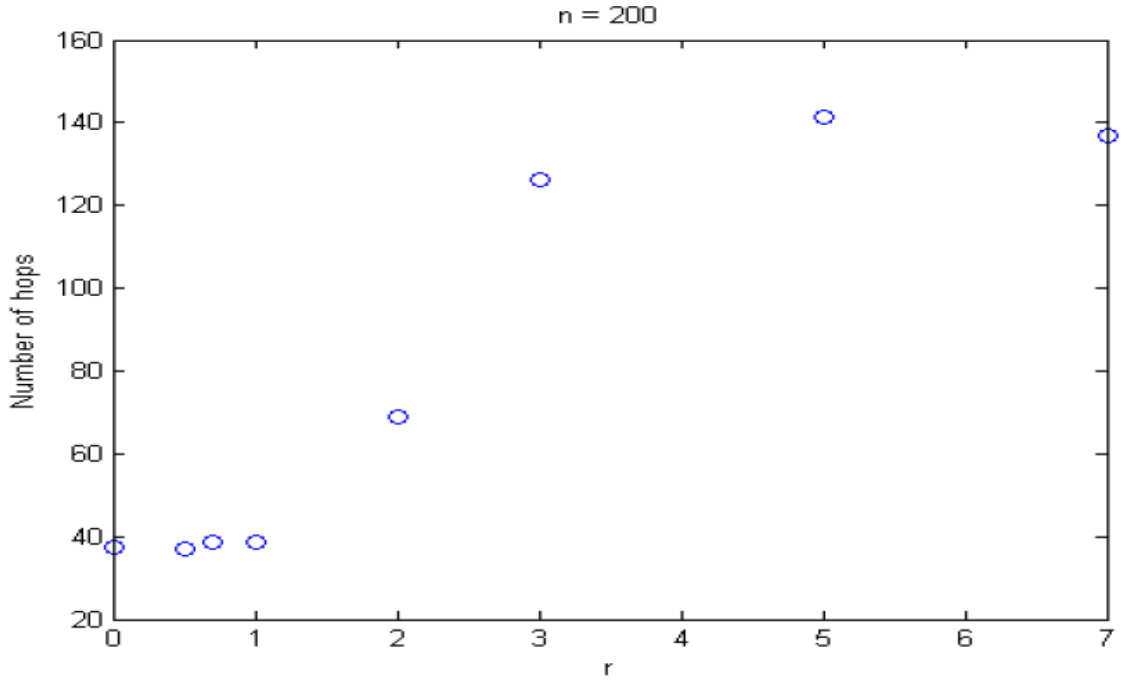

r = 7
Number of hops
n
0
50
100
150
200
250
300
350
400

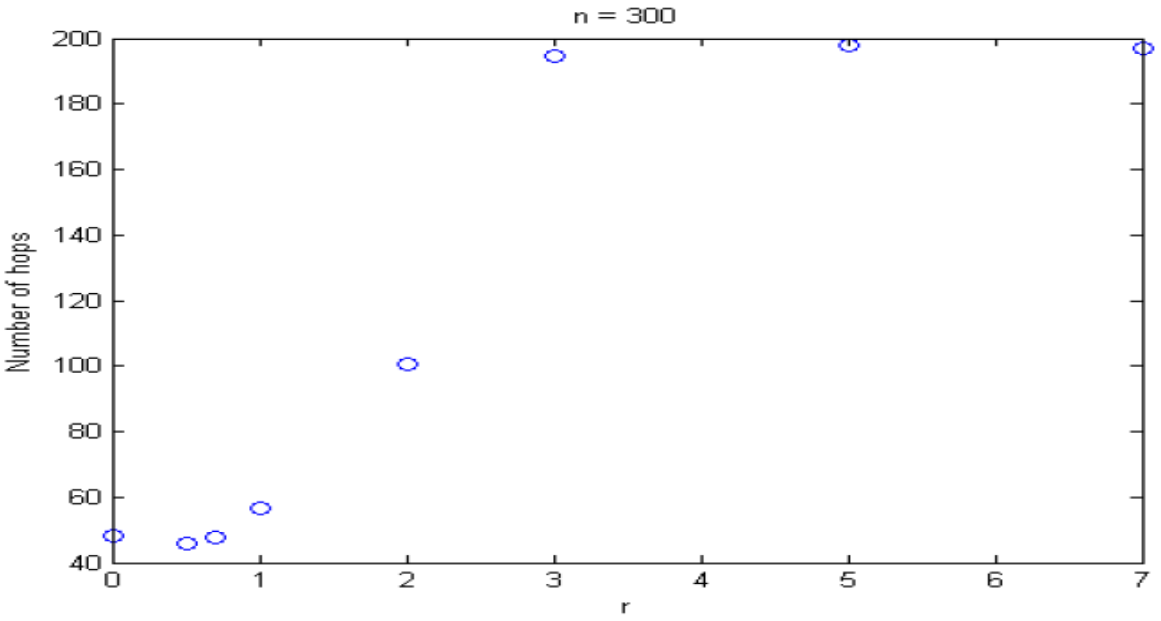
4. charts, one for each choice of *n*, with *r* on the x-axis and the mean number of hops on the y-axis.

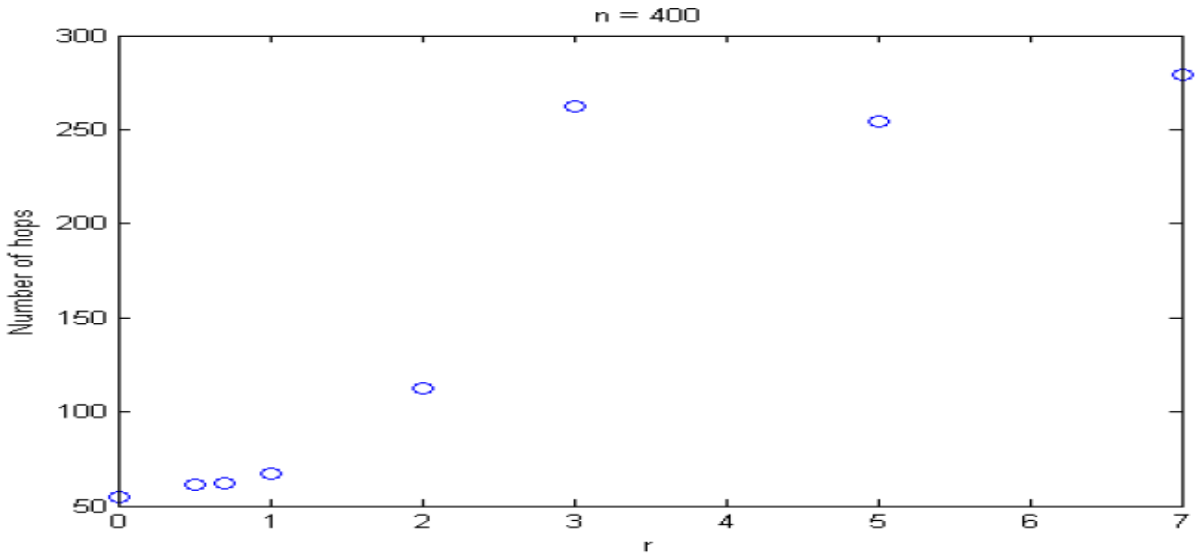





n = 100
Number of hops
r
70
65
60
55
50
45
40
35
30
25
20
0
1
2
3
4
5
6
7


n = 200
Number of hops
r
160
140
120
100
80
60
40
20
0
1
2
3
4
5
6
7


n = 300
Number of hops
r
200
180
160
140
120
100
80
60
40
0
1
2
3
4
5
6
7


n = 400
Number of hops
r
300
250
200
150
100
50
0
1
2
3
4
5
6
7

5. Matlab Source Code For Plotting Graphs and estimation of α by partial differentiation method:

```
% Using partial differentiation

clc; close all; clear all;

load('three_variables.mat');
% For r = {0,0.5,0.7,1,2,35,7} perform curve fitting for each
% n={25,50,100,200,300,400} on X- axis and number of hops on Y- axis
for i = 1:length(r)
    figure;
    plot(n,hops(i,:),'o');
    title(['r = ',num2str(r(i))]); xlabel('n');ylabel('Number of hops');
    hold on;
    rr = r(i);
    % for r =0 perform curve fitting by calculating alpha for SSE
    if (rr == 0)
        a(i) = (sum((n.^(2/3)).*hops(i,:)))/(sum(n.^(4/3)));
        y = a(i)*(n.^(2/3));
    % for r < 2  perform curve fitting by calculating alpha for SSE
    elseif (0< rr <2)
        a(i) = sum((n.^((2-rr)/3)).*hops(i,:)) / sum(n.^((4-2*rr)/3));
        y = a(i)*(n.^((2-rr)/3));
    end

   % for r = 2  perform curve fitting by calculating alpha for SSE
    if (rr == 2)
        a(i) = 1/((sum((log2(n)).^4))/(sum(((log2(n)).^2).*hops(i,:))));
        y = a(i)*((log2(n)).^2);
   % for r > 2  perform curve fitting by calculating alpha for SSE
    elseif (rr > 2)
        w = (rr-2)/(rr-1);
        a(i) = sum((n.^w).*hops(i,:)) / sum(n.^(2*w));
        y = a(i)*(n.^w);
    end
    plot(n,y,'b');
end



% For each n={25,50,100,200,300,400} draw graph r = {0,0.5,0.7,1,2,35,7}
% X- axis and number of hops on Y- axis

for j = 1:length(n)
    figure;
    plot(r,hops(:,j),'o');
    title(['n = ',num2str(n(j))]); xlabel('r');ylabel('Number of hops');
end

disp(a);
```

```
    1.0482    2.7245    4.0144    7.3463    1.3059   10.5440    2.7153    1.7660
```

6. Matlab Source Code for estimating α using the least-squares method of linear regression:

```
% Using linear curves

clc; close all; clear all;

load('three_variables.mat');
%ftype = fittype('poly1');

for i = 1:length(r)
    figure;
    plot(n,hops(i,:),'o');
    title(['r = ',num2str(r(i))]); xlabel('n');ylabel('Number of hops');
    hold on;
    if r(i) == 0
        a(i) = (length(n)*sum((n.^(2/3)).*hops(i,:)) - (sum(n.^(2/3)))*(sum(hops(i
        y = a(i)*(n.^(2/3));
    elseif 0< r(i) <2
        a(i) = (length(n)*sum((n.^((2-r(i))/3)).*hops(i,:)) - (sum(n.^((2-r(i))/3)
        y = a(i)*(n.^((2-r(i))/3));
    end
    if r(i) == 2
        a(i) = (length(n)*sum((log2(n).^2).*hops(i,:)) - (sum(log2(n).^2))*(sum(ho
        y = a(i)*((log2(n)).^2);
    elseif r(i) > 2
        a(i) = (length(n)*sum((n.^((r(i)-2)/(r(i)-1))).*hops(i,:)) - (sum(n.^((r(i
        y = a(i)*(n.^((r(i)-2)/(r(i)-1)));
    end
    plot(n,y,'b');
end

disp(a);
```

```
    1.0223    3.3397    5.4982   13.0049    1.9267   16.2500    3.0813    1.9417
```

For r=0 estimation of α :

According to paper for r=0 and particular p, q independent of n the expected delivery time of any decentralized algorithm is at least α n^(2/3).

We can find partial derivative of the above equation y= α n^(2/3).

SSE = (y- α n^(2/3))^2

Thus partially differentiating with respect to α. We get

α= ∑i (x_i ^(2/3). y_i)/ ∑ x_i^(4/3);

Substituting all values of x_i= n in {25,50,100,200,300,400} and y_i=number of hops calculated .

α = 1.0482 for r=0 by partial derivation

or we can approximate the curve by least square methods of linear regression.

$$a = \frac{n\sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n\sum_i x_i^2 - (\sum_i x_i)^2},$$

Substituting x = f(x)= x^(2/3) for approximation

α = (n ∑i (x_i^(2/3) .y_i) - (∑i x_i^(2/3)) (∑i y_i)) / (n ∑i x_i^(4/3) - (∑i x^(2/3))^2)

thus by substituting value of x_i and y_i we get

α = 1.0223

For r=2 estimation of α :

According to paper for r=2 and particular p, q independent of n the expected delivery time of any decentralized algorithm is at least α (log n)^2.

We can find partial derivative of the equation y= α (log n)^2

SSE= (y - α (log n)^2)^2

Partially differentiating with α we get

α = (∑i (log x_i)^2 * y_i) / (∑i (log x_i)^4)

by substituting values of x_i and y_i we get α = 1.3059

or we can approximate the curve by least square methods of linear regression.

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2},$$

Substituting x = f(x)= (log x)^2 for approximation

α = (n ∑i ((log x_i)^2 .y_i) - (∑i (log x_i)^2) (∑i y_i)) / (n ∑i (log x_i)^2) - (∑i (log x_i)^2)^2)

α = 1.9267

For 0<r<2 estimation of α :

0 < r < 2. There is a constant α depending on p, q, r, but independent of n, so that the expected delivery time of any decentralized algorithm is at least α n^((2-r)/3).

We can find partial derivative of the equation y= α n^((2-r)/3).

SSE= (y - α n^((2-r)/3))^2

Partially differentiating with α we get

α = (∑i x_i^ ((2-r)/3) *y_i) / (∑i x_i^((4-2r)/3))

substituting values of x_i and y_i we get

α = 2.7245 for r=0.5

α = 4.0144 for r=0.7

α = 7.3463 for r=1

or we can approximate the curve by least square methods of linear regression.

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2},$$

Substituting x = f(x)= x^ ((2-r)/3) for approximation

α = (n ∑i (x_i^ ((2-r)/3).y_i) - (∑i x_i^ ((2-r)/3)) (∑i y_i)) / (n (∑i(x_i^ ((2-r)/3)) ^2) - (∑i x^ ((2-r)/3)))^2

by substituting values of x_i and y_i we get

α = 3.3397 for r=0.5

α = 5.4982 for r=0.7

α = 13.0049 for r=1

For r > 2 estimation of α :

For r > 2. There is a constant α, depending on p, q, r, but independent of n, so that the expected delivery time of any decentralized algorithm is at least α n^((r-2)/(r-1)).

Let w = (r-2)/(r-1)

We can find partial derivative of the equation y= α n^(w).

SSE = (y - α n^(w))^2.

Partially differentiating with α we get

α = (∑i (x_i)^w . y_i)/ (∑i (x_i)^2w)

substituting values of x_i and y_i we get

α = 10.544 for r=3

α = 2.7153 for r=5

α = 1.7660 for r=7

or we can approximate the curve by least square methods of linear regression.

$$a = \frac{n \sum_i x_i y_i - (\sum_i x_i)(\sum_i y_i)}{n \sum_i x_i^2 - (\sum_i x_i)^2},$$

Substituting x = f(x)= x^ w for approximation

α = (n ∑i ((x_i)^w . y_i) - (∑i x_i^ (w) (∑i y_i)) / (n ∑i(x_i^ (w) ^2 - ((∑i x^ (w))^2)

by substituting values of x_i and y_i we get

α = 16.25 for r=3

α = 3.0813 for r=5

α = 1.9417 for r=7

Conclusion :

According to paper we have following conditions

(i) for r=0 and particular p, q independent of n the expected delivery time of any decentralized algorithm is at least $\alpha\, n^{(2/3)}$.

(ii) for r=2 and particular p, q independent of n the expected delivery time of any decentralized algorithm is at least $\alpha\, (\log n)^2$.

(iii) for $0 < r < 2$. There is a constant α depending on p, q, r, but independent of n, so that the expected delivery time of any decentralized algorithm is at least $\alpha\, n^{((2-r)/3)}$.

(iv) $r > 2$. There is a constant α, depending on p, q, r, but independent of n, so that the expected delivery time of any decentralized algorithm is at least $\alpha\, n^{((r-2)/(r-1))}$.

For all these above values of r and corresponding equation we have estimated α and we perform curve fitting.

for r=2 it tries to cover points as it is unique exponent at which node's long range contacts are nearly uniformly distributed over all distance scales. Paper states that If we divide the grid into set A(0),A(1),…. ,A(log n) then for r=2 long range hops are nearly equally like to be any of the above sets, for $r<2$ there is bias for A(j) for greater distance and for $r > 2$ there is a bias for smaller distance. Thus we have implemented the algorithm and we have calculated the number of hops for different values n and r and we trying to see that the observations can be actually fitted into the expected curve giving the expected results.