Philip Scuderi

Stanford University

CS246

Winter 2013

Homework 3

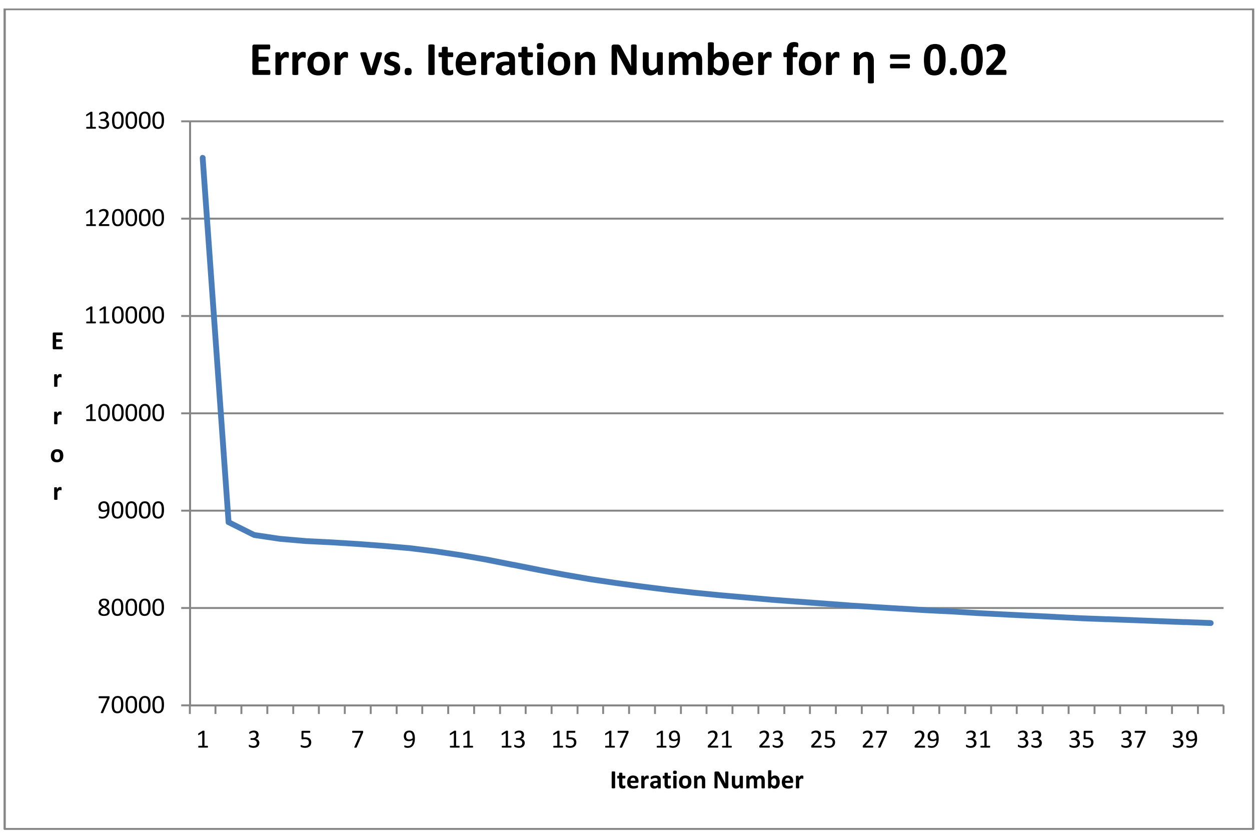
1(a)

$\varepsilon_{iu} = \mathrm{r}_{iu} - q_i \cdot p_u^T$

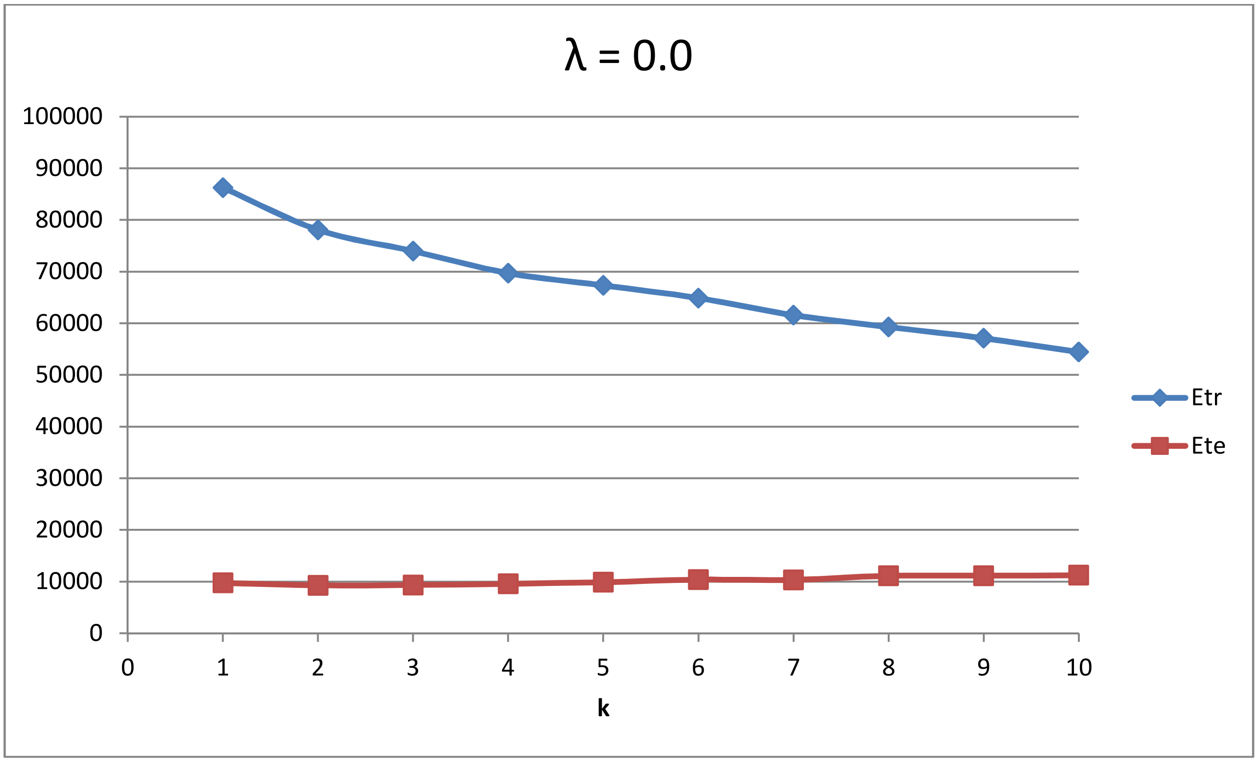
$q_i = q_i + \eta(\varepsilon_{iu} p_u - \lambda q_i)$

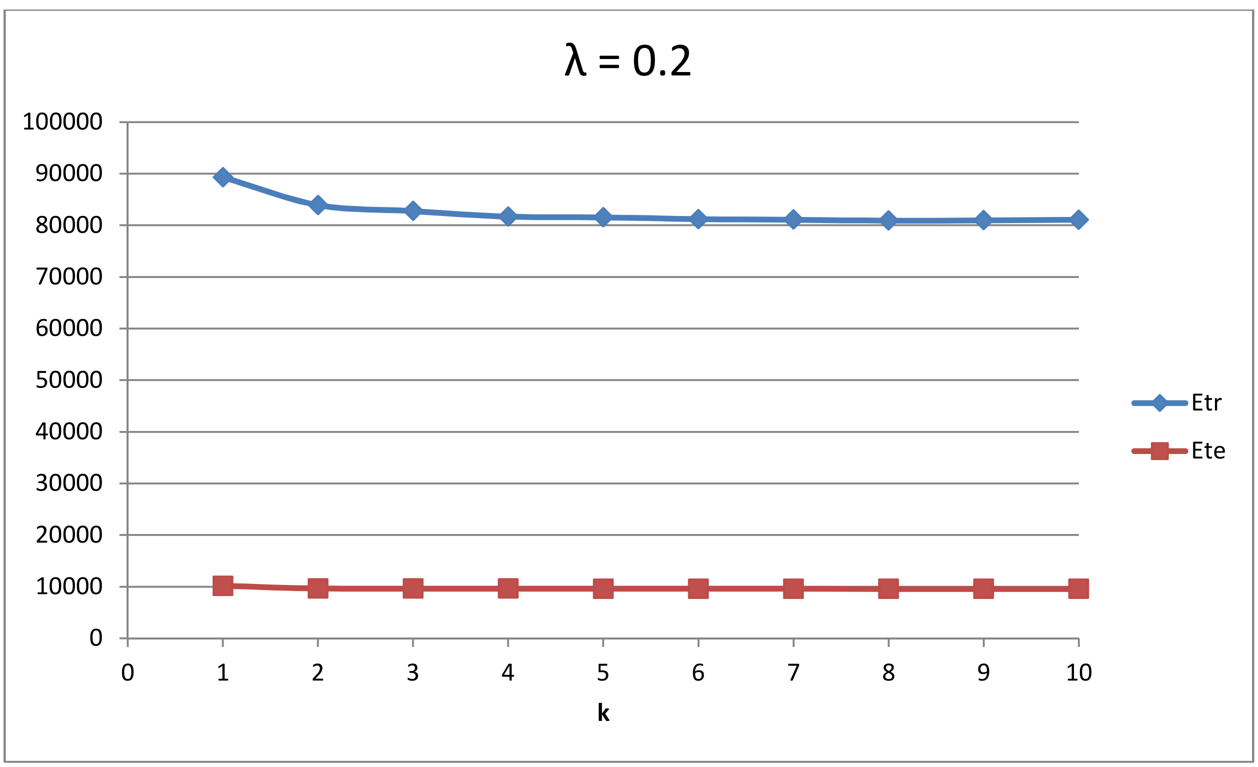
$p_u = p_u + \eta(\varepsilon_{iu} q_i - \lambda p_u)$

1(b)



1(c)


λ = 0.0
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0 1 2 3 4 5 6 7 8 9 10
k
Etr
Ete


λ = 0.2
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0 1 2 3 4 5 6 7 8 9 10
k
Etr
Ete

The following statements are valid (true):

B: Regularization decreases the test error for $\mathrm{k} \geq 5$

D: Regularization increases the training error for all (or almost all) k.

H: Regularization decreases overfitting.

The other statements (A, C, E, F, G, and I) are invalid (false).

2(a)

r and r^k are defined as follows:

$$r = \beta M r + \frac{(1-\beta)}{n} 1^T \tag{2.a.1}$$

$$r^k = \beta M r^{k-1} + \frac{(1-\beta)}{n} 1^T \tag{2.a.2}$$

We will now solve for $\left\|r - r^k\right\|_1$ using equations (2.a.1) and (2.a.2).

$$\left\|r - r^k\right\|_1$$

$$\left\|\left(\beta M r + \frac{(1-\beta)}{n} 1^T\right) - \left(\beta M r^{k-1} + \frac{(1-\beta)}{n} 1^T\right)\right\|_1$$

$$\left\|(\beta M)\left(r - r^{k-1}\right)\right\|_1$$

$$\left\|(\beta M)^k \left(r - \frac{1}{n} 1^T\right)\right\|_1$$

$$\left\|r - r^k\right\|_1 = \beta^k \left\|M^k \left(r - \frac{1}{n} 1^T\right)\right\|_1 \tag{2.a.3}$$

Because we know that r^k is column stochastic $\forall k \geq 0$, we know that:

$$\left\|r^k\right\|_1 = 1, \forall k \geq 0 \tag{2.a.4}$$

If we set $\beta = 1$ in equation (2.a.2), which is within our bounds for β, then we have the following:

$$r^k = Mr^{k-1}$$

$$\|r^k\|_1 = \|Mr^{k-1}\|_1 \tag{2.a.5}$$

It follows from equation (2.a.5) that the matrix multiplication of M times any given r^{k-1} cannot change the 1-norm of r^{k-1}. It follows that:

$$\|M^k r\|_1 = 1 \tag{2.a.6}$$

$\frac{1}{n}1^T$ is by definition column stochastic, therefore:

$$\left\|\frac{1}{n}1^T\right\|_1 = 1$$

By the same reasoning we used to derive equation (2.a.6), it follows that:

$$\left\|M^k \frac{1}{n}1^T\right\|_1 = 1 \tag{2.a.7}$$

From equation (2.a.3), we know:

$$\|r - r^k\|_1 = \beta^k \left\|M^k\left(r - \frac{1}{n}1^T\right)\right\|_1$$

$$\|r - r^k\|_1 = \beta^k \left\|M^k r - M^k \frac{1}{n}1^T\right\|_1 \leq \beta^k\|M^k r\|_1 + \beta^k \left\|M^k \frac{1}{n}1^T\right\|_1 \tag{2.a.8}$$

Substituting equations (2.a.6) and (2.a.7) into (2.a.8), we obtain:

$$\|r - r^k\|_1 \leq \beta^k * 1 + \beta^k * 1 = 2\beta^k$$

$$\|r - r^k\|_1 \leq 2\beta^k$$

<u>2(b)</u>

From part 2(a) we know that the L_1 error is bounded as follows:

$$\|r - r^k\|_1 \leq 2\beta^k$$

$$2\beta^k \leq \delta$$

$$\beta^k \leq \frac{\delta}{2}$$

$$\log_\beta(\beta^k) \leq \log_\beta\left(\frac{\delta}{2}\right)$$

$$k \leq \frac{\log\left(\frac{\delta}{2}\right)}{\log(\beta)}$$

$$k \leq \frac{-\log\left(\frac{\delta}{2}\right)}{-\log(\beta)}$$

$$k \leq \frac{-\log\left(\frac{\delta}{2}\right)}{\log\left(\frac{1}{\beta}\right)}$$

$$k \leq O\left(\frac{1}{\log\left(\frac{1}{\beta}\right)}\right)$$

Thus, the number of power iterations (k) that we need to perform to guarantee the L_1 error is less than a constant δ is on the order of $\frac{1}{\log\left(\frac{1}{\beta}\right)}$.

m is the number of edges in our graph. Therefore m corresponds to the number of non-zero elements in M. This means that each power iteration requires $O(m)$ operations.

So, to guarantee the L_1 error is less than a constant δ the number of operations we need to perform is as follows:

$$O(km) \leq O\left(\frac{m}{\log\left(\frac{1}{\beta}\right)}\right)$$

The running time is on the order of the number of operations. As such the running time to guarantee the L_1 error is less than a constant δ is $O\left(\frac{m}{\log\left(\frac{1}{\beta}\right)}\right)$.

2(c)

Let $\tilde{\mathrm{p}}_j^i$ represent the fraction of random walks that start at node i and end at node j.

We can express the estimation $\tilde{r}_j$ obtained by the MC algorithm as a function of $\tilde{\mathrm{p}}_j^i$ as follows:

$$\tilde{r}_j = \frac{1}{n}\sum_{i=1}^{n} \tilde{\mathrm{p}}_j^i$$

The expected value of $\tilde{r}_j$ is therefore defined as follows:

$$E[\tilde{r}_j] = \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n} \tilde{\mathrm{p}}_j^i\right]$$

$\frac{1}{n}\sum_{i=1}^{n} \tilde{\mathrm{p}}_j^i$ is equivalent to the probability of ending at node j starting from a *random* node. In other words:

$$\frac{1}{n}\sum_{i=1}^{n} \tilde{\mathrm{p}}_j^i = r_j$$

Where r_j is the probability of ending at node j produced by the PageRank algorithm. Therefore:

$$E[\tilde{r}_j] = \mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n} \tilde{\mathrm{p}}_j^i\right] = E[r_j] = r_j$$

<u>2(d)</u>

Let the random variable K represent the number of steps of any of the individual random walks. The probability of a random walk continuing from a given node is β and the probability of a random walk ending at a given node is $1-\beta$. This is simply a sequence of Bernoulli trains with probability of "success" (ending the random walk) of $1-\beta$. Thus we can represent K as follows:

$$P(K = k) = \beta^{k-1}(1-\beta)$$

As you can see, K follows a geometric distribution with $p = 1-\beta$. The expected value of a geometric distribution is well known as $\frac{1}{p}$. It follows that:

$$E[K] = \frac{1}{1-\beta}$$

In the MC algorithm we simulate nR random walks. Therefore the expected running time is as follows:

$$O(nR * E[K]) = O\left(\frac{nR}{1-\beta}\right)$$

2(e)

PageRank

Time Elapsed = 00:00:00.0915033

MC Algorithm

R = 1:
Time Elapsed = 00:00:00.0000252
Average Error (Top 10) = 0.00640663222420862
Average Error (Top 30) = 0.00480184089326871
Average Error (Top 50) = 0.00393012276692224
Average Error (Top 100) = 0.0028437844810428

R = 3:
Time Elapsed = 00:00:00.0001017
Average Error (Top 10) = 0.00407329889087529
Average Error (Top 30) = 0.00230528124835267
Average Error (Top 50) = 0.00185090881125131
Average Error (Top 100) = 0.00149015695379502

R = 5:
Time Elapsed = 00:00:00.0001674
Average Error (Top 10) = 0.00280723868805303
Average Error (Top 30) = 0.00199048241770075
Average Error (Top 50) = 0.00171602213055849
Average Error (Top 100) = 0.00129368788756991

3(a)

```
A0 = eye(3);
B0 = eye(5);

A1 = A0;
A1(1,2) = C1 * (B0(2,2)+B0(2,4)+B0(3,2)+B0(3,4)+B0(5,2)+B0(5,4))  /  (3*2);
A1(2,1) = A1(1,2);
A1(1,3) = C1 * (B0(2,1)+B0(4,1)) / (3*1);
```

```
A1(3,1) = A(1,3);
A1(2,3) = C1 * (B0(2,1)+B0(4,1)) / (2*1);
A1(3,2) = A(2,3);

B1 = B0;
B1(1,2) = C2 * (A0(3,1)+A0(3,2)) / (1*2);
B1(2,1) = B1(1,2);
B1(1,3) = C2 * (A0(3,1)) / (1*1);
B1(3,1) = B1(1,3);
B1(1,4) = C2 * (A0(3,2)) / (1*1);
B1(4,1) = B1(1,4);
B1(1,5) = C2 * (A0(3,1)) / (1*1);
B1(5,1) = B1(1,5);
B1(2,3) = C2 * (A0(1,1)+A0(2,1)) / (2*1);
B1(3,2) = B1(2,3);
B1(2,4) = C2 * (A0(1,2)+A0(2,2)) / (2*1);
B1(4,2) = B1(2,4);
B1(2,5) = C2 * (A0(1,1)+A0(2,1)) / (2*1);
B1(5,2) = B1(2,5);
B1(3,4) = C2 * (A0(1,2)) / (1*1);
B1(4,3) = B1(3,4);
B1(3,5) = C2 * (A0(1,1)) / (1*1);
B1(5,3) = B1(3,5);
B1(4,5) = C2 * (A0(2,1)) / (1*1);
B1(5,4) = B1(4,5);
```

<u>Results after iteration 1</u>

```
A1 =

    1.0000    0.1333         0
    0.1333    1.0000         0
         0         0    1.0000


B1 =

    1.0000         0         0         0         0
         0    1.0000    0.4000    0.4000    0.4000
         0    0.4000    1.0000         0    0.8000
         0    0.4000         0    1.0000         0
```

```
         0    0.4000    0.8000         0    1.0000
```

Results after iteration 2

A2 =

```
    1.0000    0.2933         0
    0.2933    1.0000         0
         0         0    1.0000
```

B2 =

```
    1.0000         0         0         0         0
         0    1.0000    0.4533    0.4533    0.4533
         0    0.4533    1.0000    0.1067    0.8000
         0    0.4533    0.1067    1.0000    0.1067
         0    0.4533    0.8000    0.1067    1.0000
```

Results after iteration 3

A3 =

```
    1.0000    0.3431         0
    0.3431    1.0000         0
         0         0    1.0000
```

B3 =

```
    1.0000         0         0         0         0
         0    1.0000    0.5173    0.5173    0.5173
         0    0.5173    1.0000    0.2347    0.8000
         0    0.5173    0.2347    1.0000    0.2347
         0    0.5173    0.8000    0.2347    1.0000
```

Results

The requested results after three iterations are as follows:

$S_A(camera, phone) = A3(1,2) = 0.3431$

$S_A(camera, printer) = A3(1,3) = 0$

3(b)

In the summation, we should multiply the similarities by its weights. We should also normalize by dividing by the sum of the weights.

$$s_A(X,Y) = \frac{C_1}{\sum_{i=1}^{|O(X)|} W(X, O_i(X)) * \sum_{j=1}^{|O(Y)|} W(Y, O_j(Y))} \sum_{i=1}^{|O(X)|} \sum_{j=1}^{|O(Y)|} [W(X, O_i(X)) * W(Y, O_j(Y)) * s_A(O_i(X), O_j(Y))]$$

$$s_B(X,Y) = \frac{C_2}{\sum_{i=1}^{|I(X)|} W(I_i(X), X) * \sum_{j=1}^{|I(Y)|} W(I_j(Y), Y)} \sum_{i=1}^{|I(X)|} \sum_{j=1}^{|I(Y)|} [W(X, I_i(X)) * W(Y, I_j(Y)) * s_A(I_i(X), I_j(Y))]$$

3(c)

Details for Iteration 1 of $K_{2,1}$

```
A0 = eye(2);
B0 = eye(1);

A1 = A0;
A1(1,2) = C1 * (B0(1,1)) / (1*1);
A1(2,1) = C1 * (B0(1,1)) / (1*1);

B1 = B0;
```

Similarity Scores for $K_{2,1}$

```
A1 =

    1.0000    0.8000
    0.8000    1.0000


B1 =

     1
```

```
A2 =

    1.0000    0.8000
    0.8000    1.0000


B2 =

     1


A3 =

    1.0000    0.8000
    0.8000    1.0000


B3 =

     1
```

<u>Details for Iteration 1 of $K_{2,2}$</u>

```
A0 = eye(2);
B0 = eye(2);

A1 = A0;
A1(1,2) = C1 * (B0(1,2)+B0(2,1)) / (2*2);
A1(2,1) = A1(1,2);

B1 = B0;
B1(1,2) = C2 * (A0(1,2)+A0(2,1)) / (2*2);
B1(2,1) = B1(1,2);
```

<u>Similarity Scores for $K_{2,2}$</u>

```
A1 =

    1.0000    0.4000
```

0.4000 1.0000

B1 =

1.0000	0.4000
0.4000	1.0000

A2 =

1.0000	0.5600
0.5600	1.0000

B2 =

1.0000	0.5600
0.5600	1.0000

A3 =

1.0000	0.6240
0.6240	1.0000

B3 =

1.0000	0.6240
0.6240	1.0000

<u>How the Similarity Scores for $K_{2,1}$ and $K_{2,2}$ compare after 3 Iterations</u>

The similarity score A3(1,2) for $K_{2,1}$ is higher than the similarity A3(1,2) score for $K_{2,2}$ by 0.176.

3(d)

We use the same formulas we came up with in part 3(b), but change the weigh formula W(X, Y) to count the number of neighbors that nodes X and Y share.

We know that this similarity score would never be greater than 1 because we normalize in part 3(b) by dividing by the sum of the weights.

3(e)

$s_A(x, y)$ is the probability of the two random walkers starting at x and y ending at the same node.

4(a)(i)

We assume the following:

$$|A(S)| < \frac{\epsilon}{1+\epsilon}|S| \tag{4.a.1}$$

$$|S \setminus A(S)| = |S| - |A(S)| \tag{4.a.2}$$

Applying equation (4.a.1) to equation (4.a.2), we obtain:

$$|S \setminus A(S)| = |S| - |A(S)| \geq |S| - \frac{\epsilon}{1+\epsilon}|S| = |S|\left(1 - \frac{\epsilon}{1+\epsilon}\right) = |S|\left(\frac{1}{1+\epsilon}\right)$$

$$|S \setminus A(S)| \geq |S|\left(\frac{1}{1+\epsilon}\right) \tag{4.a.3}$$

Remember that $A(S)$ is defined as follows:

$$A(S) = \{i \in S \mid deg_S(i) \leq 2(1+\epsilon)\rho(S)\}$$

Therefore $S \setminus A(S)$ contains the elements in S that are not in A(S), specifically:

$$S \setminus A(S) = \{i \in S \mid deg_S(i) > 2(1+\epsilon)\rho(S)\} \tag{4.a.4}$$

We can calculate the density $\rho(S)$ as follows. Note that by summing the degree of each node we are double counting each edge, which is why the denominator below contains $2|S|$.

$$\rho(S) = \frac{\sum_{i\in S} deg_S(i)}{2|S|} = \frac{\sum_{i\in A(S)} deg_S(i) + \sum_{i\in S\setminus A(S)} deg_S(i)}{2|S|}$$

$$\rho(S) = \frac{\sum_{i \in A(S)} deg_S(i)}{2|S|} + \frac{\sum_{i \in S \setminus A(S)} deg_S(i)}{2|S|} \tag{4.a.5}$$

Applying equations (4.a.3) and (4.a.4) to equation (4.a.5), we obtain:

$$\rho(S) = \frac{\sum_{i \in A(S)} deg_S(i)}{2|S|} + \frac{\sum_{i \in S \setminus A(S)} deg_S(i)}{2|S|} > \frac{\sum_{i \in A(S)} deg_S(i)}{2|S|} + \frac{|S|\left(\frac{1}{1+\epsilon}\right)2(1+\epsilon)\rho(S)}{2|S|}$$

$$\rho(S) > \frac{\sum_{i \in A(S)} deg_S(i)}{2|S|} + \rho(S)$$

Clearly, $\rho(S)$ cannot be larger than $\rho(S)$ plus a positive constant. Therefore, our assumption in equation (4.a.1) must be wrong. We therefore conclude the following:

$$|A(S)| \geq \frac{\epsilon}{1+\epsilon}\,|S|$$

4(a)(ii)

$$|S \setminus A(S)| \;=\; |S| - |A(S)| \;\geq\; |S| - \frac{\epsilon}{1+\epsilon}\,|S| = \;|S|\left(1 - \frac{\epsilon}{1+\epsilon}\right) = \;|S|\left(\frac{1}{1+\epsilon}\right)$$

It follows that at every iteration, the result gets smaller by at least $\frac{1}{1+\epsilon}$.

Therefore, in the worst case the algorithm will terminate in at most $log_{1+\epsilon}(n)$ iterations.

4(b)(i)

We assume the following:

Let node $x \in S$ such that $deg_{S*}(x) \;<\; \rho^*(G)$ (4.b.1)

$$\rho(S^* \backslash \{x\}) = \frac{|E[S^*]| - deg_{S*}(x)}{|S^*|-1} \tag{4.b.2}$$

Because $deg_{S*}(x) \;<\; \rho^*(G)$ it follows that we can rewrite equation (4.b.1) as:

$$\rho(S^* \backslash \{x\}) = \frac{|E[S^*]| - deg_{S*}(x)}{|S^*|-1} > \frac{|E[S^*]| - \rho^*(G)}{|S^*|-1} \tag{4.b.3}$$

We know by definition that $\rho^*(G) = \max_{S \subseteq V}\{\rho(S)\}$. Because S^* is the densest subgraph of G it follows that:

$$\rho^*(G) = \rho(S^*) \tag{4.b.4}$$

Applying equation (4.b.4) to equation (4.b.3), we obtain:

$$\rho(S^* \backslash \{x\}) > \frac{|E[S^*]| - \rho(S^*)}{|S^*| - 1} = \frac{|E[S^*]| - \frac{|E[S^*]|}{|S^*|}}{|S^*| - 1} = \frac{|E[S^*]|}{|S^*|} = \rho^*(G)$$

However, $\rho(S^* \backslash \{x\})$ cannot be greater than $\rho^*(G)$ because $\rho^*(G) = \max_{S \subseteq V}\{\rho(S)\}$. As such, our assumption is wrong and we have proved the following:

$$deg_{S*}(v) \geq \rho^*(G), \forall v \in S$$

4(b)(ii)

We are given $v \in S^* \cap A(S)$

Recall that the definition of A(S) is:

$$A(S) = \{i \in S \mid deg_S(i) \leq 2(1+\epsilon)\rho(S)\}$$

Because $v \in A(S)$, we know that:

$$deg_S(v) \leq 2(1+\epsilon)\rho(S) \tag{2.b.5}$$

We showed in 4(b)(i) that:

$$deg_S(v) < \rho^*(G) \tag{2.b.6}$$

Combining equations (2.b.5) and (2.b.6) we obtain:

$$2(1+\epsilon)\rho(S) \geq deg_S(v) \geq \rho^*(G)$$

$$2(1+\epsilon)\rho(S) \geq \rho^*(G)$$

4(b)(iii)

In part 4(b)(ii) we showed:

$2(1+\epsilon)\rho(S) \geq \rho^*(G)$

$\rho(S) \geq \frac{\rho^*(G)}{2(1+\epsilon)}$ (2.b.7)

The algorithm is designed to find an induced subgraph $\tilde{S}$ of G whose density isn't much smaller than $\rho^*(G)$. Thus:

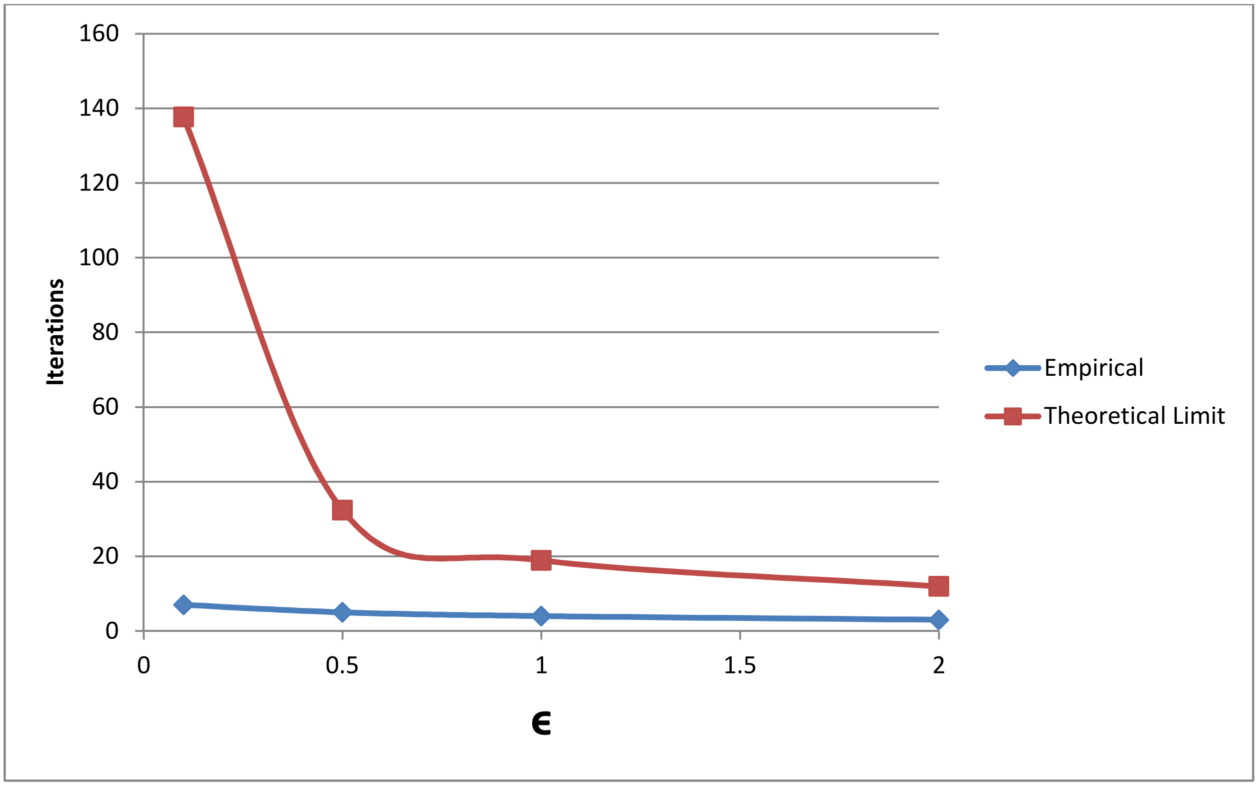
$\rho(\tilde{S}) \geq \rho(S).$ (2.b.8)

Combining equations (2.b.7) and (2.b.8), we obtain:

$$\rho(\tilde{S}) \geq \rho(S) \geq \frac{\rho^*(G)}{2(1+\epsilon)}$$

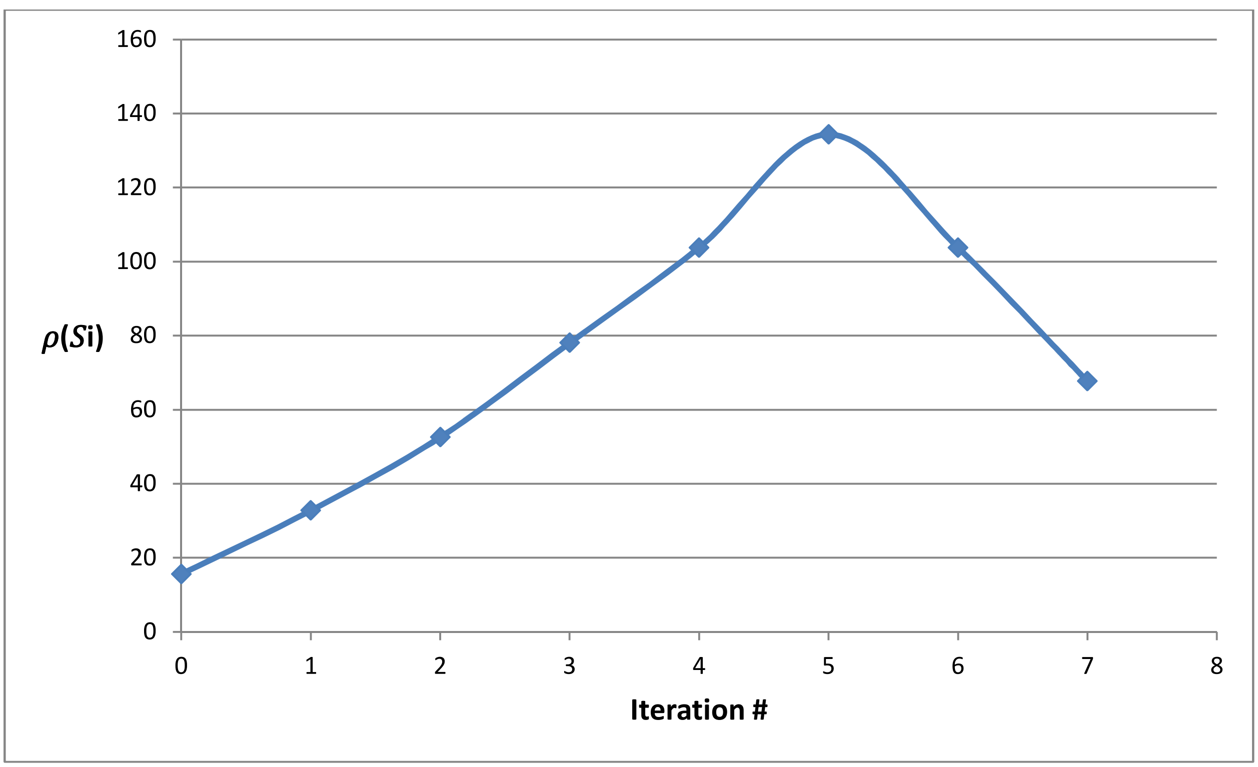
$$\rho(\tilde{S}) \geq \frac{\rho^*(G)}{2(1+\epsilon)}$$

4(c)(i)

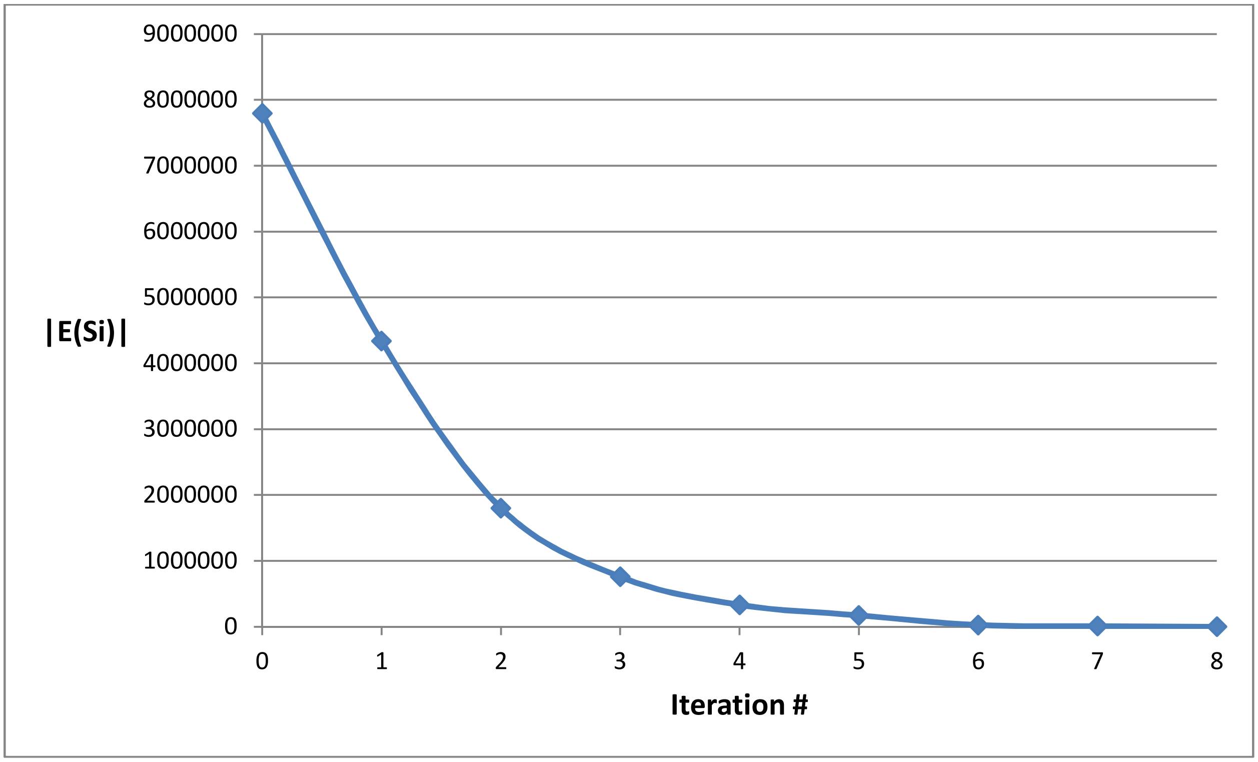


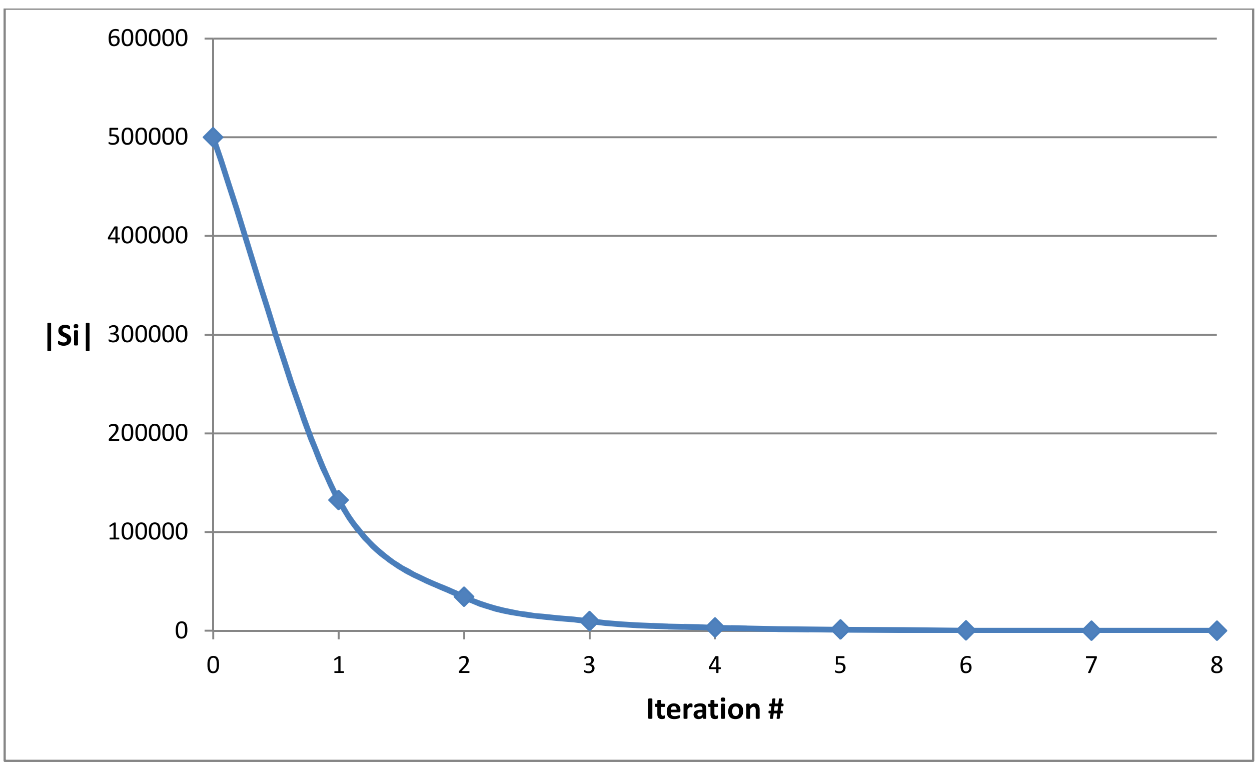
160
140
120
100
80
60
40
20
0
Iterations
0
0.5
1
1.5
2
ϵ
Empirical
Theoretical Limit

4(c)(ii)

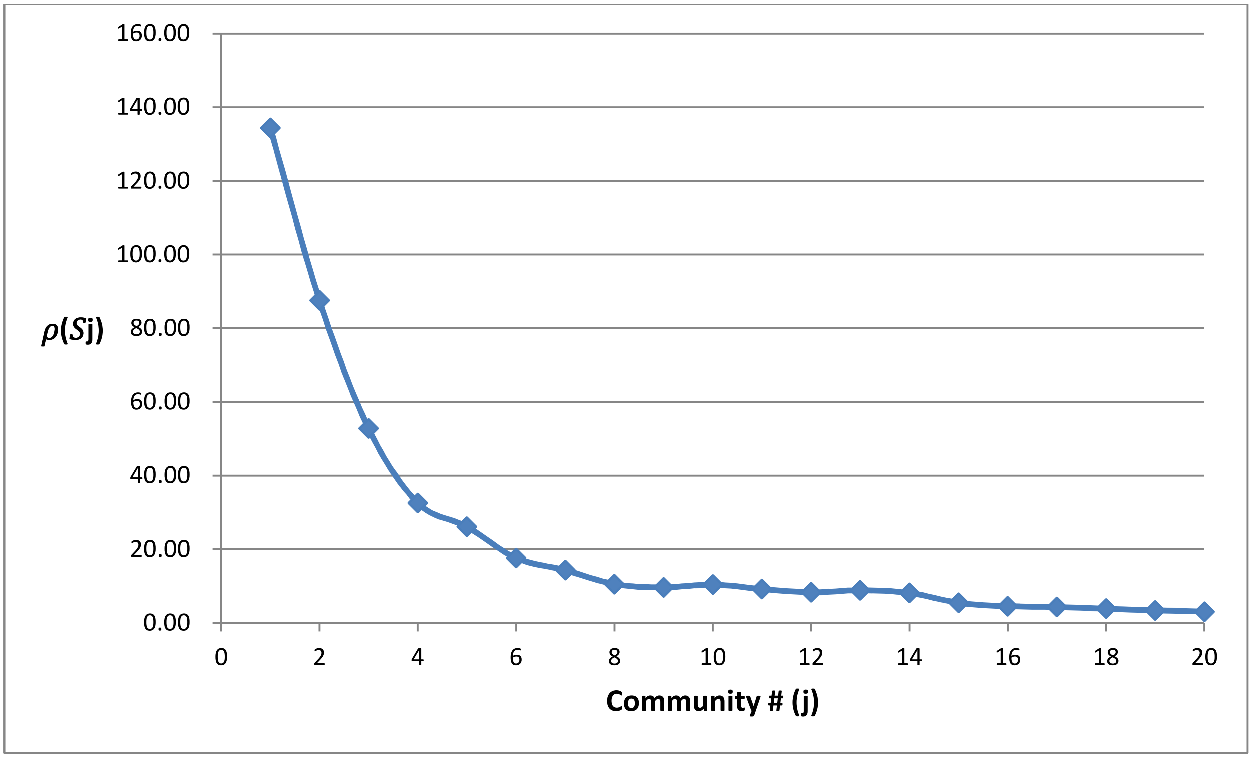


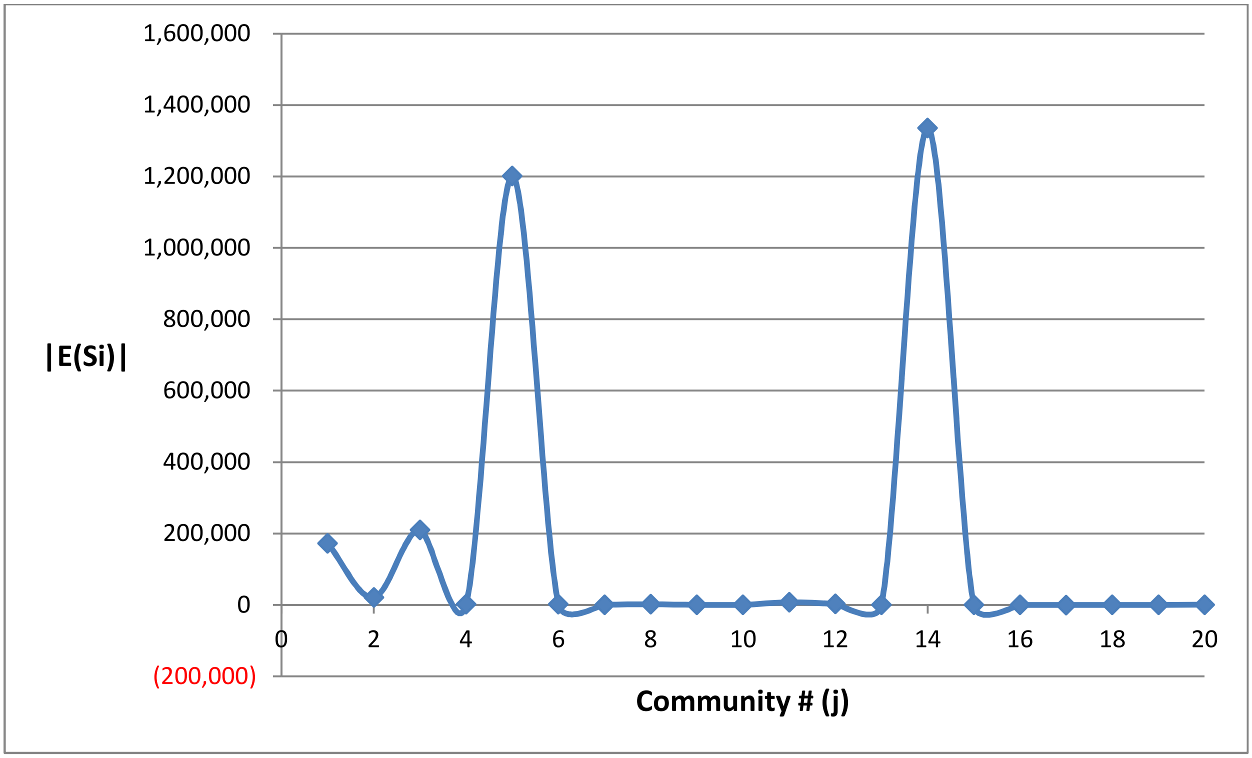
160
140
120
100
80
60
40
20
0
ρ(Si)
0
1
2
3
4
5
6
7
8
Iteration #

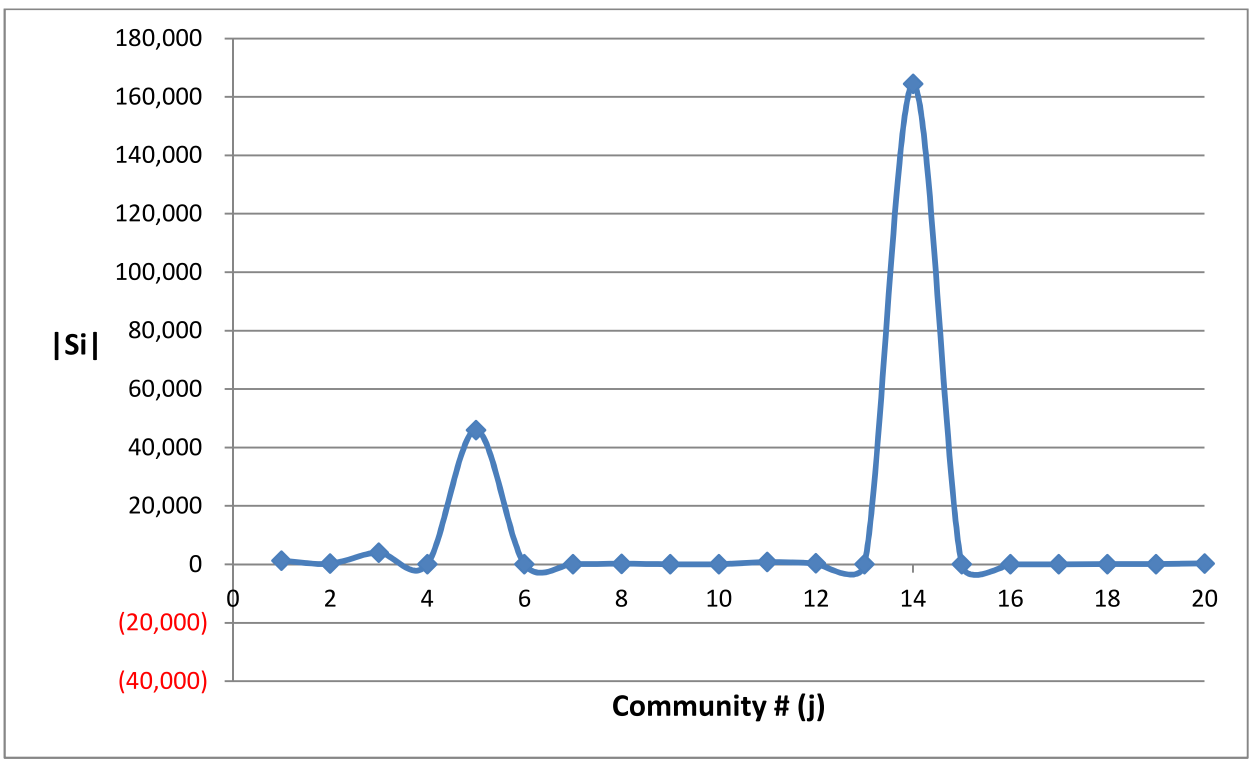

9000000
8000000
7000000
6000000
5000000
4000000
3000000
2000000
1000000
0
|E(Si)|
0
1
2
3
4
5
6
7
8
Iteration #


600000
500000
400000
300000
200000
100000
0
|Si|
0
1
2
3
4
5
6
7
8
Iteration #

4(c)(iii)


160.00
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
ρ(Sj)
0
2
4
6
8
10
12
14
16
18
20
Community # (j)


1,600,000
1,400,000
1,200,000
1,000,000
800,000
600,000
400,000
200,000
0
(200,000)
|E(Si)|
0
2
4
6
8
10
12
14
16
18
20
Community # (j)


180,000
160,000
140,000
120,000
100,000
80,000
60,000
40,000
20,000
0
(20,000)
(40,000)
|Si|
0
2
4
6
8
10
12
14
16
18
20
Community # (j)

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 3
 * Question 1
 */

package HW3_Q1c;

import java.io.*;
import java.util.*;
import Jama.Matrix;

public class StochasticGradientDescent
{
   protected String rTrainingPath, rTestPath;
   protected double lambda, eta, trainingError, regularizationAdjustment;
   protected Matrix p, q;
   protected int m, n, k;

   public static void main(String[] args) throws IOException
   {
      String path = "C:\\Dropbox\\Private\\Stanford\\CS246\\Homeworks\\HW3\\Q1\\";
```

```
        String rTrainingFile = "ratings.train.txt";
        String rTestFile = "ratings.val.txt";
        String rTrainingPath = path + rTrainingFile;
        String rTestPath = path + rTestFile;



        double eta = 0.03;
        int iterations = 40;



        double lambda = 0.0;


        System.out.println("lambda = " + lambda + ":");
        for (int k = 1; k <= 10; k++)
        {
            StochasticGradientDescent sgd = new StochasticGradientDescent(eta, lambda, k, rTrainingPath,
rTestPath);
            sgd.Iterate(iterations);
            System.out.println(k + "\t" + sgd.GetTrainingError() + "\t" + sgd.GetTestError());
        }
        System.out.println();



        lambda = 0.2;


        System.out.println("lambda = " + lambda + ":");
```

```
        for (int k = 1; k <= 10; k++)
        {
            StochasticGradientDescent sgd = new StochasticGradientDescent(eta, lambda, k, rTrainingPath,
rTestPath);
            sgd.Iterate(iterations);
            System.out.println(k + "\t" + sgd.GetTrainingError() + "\t" + sgd.GetTestError());
        }
        System.out.println();
    }

    public StochasticGradientDescent(double eta, double lambda, int k, String rTrainingPath, String
rTestPath) throws FileNotFoundException, IOException
    {
        trainingError = regularizationAdjustment = Double.NaN;

        this.lambda = lambda;
        this.k = k;
        this.eta = eta;
        this.rTrainingPath = rTrainingPath;
        this.rTestPath = rTestPath;

        // initialize m and n
        BufferedReader reader = new BufferedReader(new FileReader(rTrainingPath));
        String line;
        while ( (line = reader.readLine()) != null)
        {
```

```
    StringTokenizer tokenizer = new StringTokenizer(line);

    int u = Integer.parseInt(tokenizer.nextToken());
    int i = Integer.parseInt(tokenizer.nextToken());

    if (u > n)
        n = u;

    if (i > m)
        m = i;
}
reader.close();

Random r = new Random();

// initialize p
p = new Matrix(n, k);
for (int row = 0; row < n; row++)
    for (int col = 0; col < k; col++)
        p.set(row, col, r.nextDouble() * Math.pow(5.0 / (double)k, 0.5));

// initialize q
q = new Matrix(m, k);
for (int row = 0; row < m; row++)
    for (int col = 0; col < k; col++)
```

```
            q.set(row, col, r.nextDouble() * Math.pow(5.0 / (double)k, 0.5));
    }

    public void Iterate(int numIterations) throws FileNotFoundException, IOException
    {
      for (int a = 0; a < numIterations; a++)
      {
        trainingError = 0.0;

        BufferedReader reader = new BufferedReader(new FileReader(rTrainingPath));
        String line;
        while ( (line = reader.readLine()) != null)
        {
          StringTokenizer tokenizer = new StringTokenizer(line);

          int u = Integer.parseInt(tokenizer.nextToken());
          int i = Integer.parseInt(tokenizer.nextToken());
          int Riu = Integer.parseInt(tokenizer.nextToken());

          Matrix Qi = GetRow(q, i-1);
          Matrix Pu = GetRow(p, u-1);

          double Eiu = Riu - DotProduct(Qi, Pu);

          Matrix nextQi = Qi.plus(Pu.times(Eiu).minus(Qi.times(lambda)).times(eta));
```

```
            Matrix nextPu = Pu.plus(Qi.times(Eiu).minus(Pu.times(lambda)).times(eta));

            SetRow(q, i-1, nextQi);
            SetRow(p, u-1, nextPu);

            trainingError += Math.pow(Eiu, 2.0);
        }
        reader.close();


        regularizationAdjustment = 0.0;

        // finish determing the error (determine right side of the equation)
        for (int i = 0; i < m; i++)
            regularizationAdjustment += lambda * Math.pow(GetRow(q, i).norm2(), 2.0);

        for (int u = 0; u < n; u++)
            regularizationAdjustment += lambda * Math.pow(GetRow(p, u).norm2(), 2.0);
    }
}

public double GetError()
{
    return trainingError + regularizationAdjustment;
}
```

```
public double GetTrainingError()
{
   return trainingError;
}

public double GetTestError() throws FileNotFoundException, IOException
{
   double testError = 0.0;

   BufferedReader reader = new BufferedReader(new FileReader(rTestPath));
   String line;
   while ( (line = reader.readLine()) != null)
   {
      StringTokenizer tokenizer = new StringTokenizer(line);

      int u = Integer.parseInt(tokenizer.nextToken());
      int i = Integer.parseInt(tokenizer.nextToken());
      int Riu = Integer.parseInt(tokenizer.nextToken());

      Matrix Qi = GetRow(q, i-1);
      Matrix Pu = GetRow(p, u-1);

      double Eiu = Riu - DotProduct(Qi, Pu);
```

```
        testError += Math.pow(Eiu, 2.0);
    }
    reader.close();

    return testError;
}

// only for same size vectors (1 row matricies of the same length)
protected static double DotProduct(Matrix m1, Matrix m2)
{
    double dotProduct = 0.0;

    for (int i = 0; i < m1.getColumnDimension(); i++)
        dotProduct += m1.get(0, i) * m2.get(0, i);

    return dotProduct;
}

protected static void SetRow(Matrix m, int row, Matrix val)
{
    m.setMatrix(row, row, 0, m.getColumnDimension()-1, val);
}

protected static void SetColumn(Matrix m, int column, Matrix val)
{
```

```
        m.setMatrix(0, m.getColumnDimension()-1, column, column, val);
    }

    protected static Matrix GetRow(Matrix m, int row)
    {
        return m.getMatrix(row, row, 0, m.getColumnDimension()-1);
    }

    protected static Matrix GetColumn(Matrix m, int column)
    {
        return m.getMatrix(0, m.getRowDimension()-1, column, column);
    }
}
```

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 3
 * Question 2
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;
using Extreme.Mathematics;
using Extreme.Mathematics.LinearAlgebra;

namespace HW3_Q2
{
    class Program
    {
        const string graphPath = "../../graph.txt";
        const int n = 100;
        const int m = 1024;
        const double beta = 0.8;

        static void Main(string[] args)
        {
            // 2(e)
            Console.WriteLine("2(e): PageRank\n--------------\n");
            Tuple<Vector, KeyValuePair<int, double>[], TimeSpan> pageRankResults =
PageRank();
            Console.WriteLine("r = " + pageRankResults.Item1);
            Console.WriteLine();
            Console.WriteLine("Time Elapsed = " + pageRankResults.Item3);
            Console.WriteLine();
            Console.WriteLine();

            // 2(f)
            Console.WriteLine("2(f): MC Algorithm\n------------------\n");
            Tuple<Vector, KeyValuePair<int, double>[], TimeSpan> rw1 = RandomWalk(1);
            Tuple<Vector, KeyValuePair<int, double>[], TimeSpan> rw3 = RandomWalk(3);
            Tuple<Vector, KeyValuePair<int, double>[], TimeSpan> rw5 = RandomWalk(5);

            Console.WriteLine("R = 1:");
            Console.WriteLine("Time Elapsed = " + rw1.Item3);
            Console.WriteLine("Average Error (Top  10) = " +
AverageErrorTopK(pageRankResults.Item2, rw1.Item1, 10));
            Console.WriteLine("Average Error (Top  30) = " +
AverageErrorTopK(pageRankResults.Item2, rw1.Item1, 30));
            Console.WriteLine("Average Error (Top  50) = " +
AverageErrorTopK(pageRankResults.Item2, rw1.Item1, 50));
            Console.WriteLine("Average Error (Top 100) = " +
AverageErrorTopK(pageRankResults.Item2, rw1.Item1, 100));
            Console.WriteLine();

            Console.WriteLine("R = 3:");
```

```
            Console.WriteLine("Time Elapsed = " + rw3.Item3);
            Console.WriteLine("Average Error (Top  10) = " +
AverageErrorTopK(pageRankResults.Item2, rw3.Item1, 10));
            Console.WriteLine("Average Error (Top  30) = " +
AverageErrorTopK(pageRankResults.Item2, rw3.Item1, 30));
            Console.WriteLine("Average Error (Top  50) = " +
AverageErrorTopK(pageRankResults.Item2, rw3.Item1, 50));
            Console.WriteLine("Average Error (Top 100) = " +
AverageErrorTopK(pageRankResults.Item2, rw3.Item1, 100));
            Console.WriteLine();

            Console.WriteLine("R = 5:");
            Console.WriteLine("Time Elapsed = " + rw5.Item3);
            Console.WriteLine("Average Error (Top  10) = " +
AverageErrorTopK(pageRankResults.Item2, rw5.Item1, 10));
            Console.WriteLine("Average Error (Top  30) = " +
AverageErrorTopK(pageRankResults.Item2, rw5.Item1, 30));
            Console.WriteLine("Average Error (Top  50) = " +
AverageErrorTopK(pageRankResults.Item2, rw5.Item1, 50));
            Console.WriteLine("Average Error (Top 100) = " +
AverageErrorTopK(pageRankResults.Item2, rw5.Item1, 100));
            Console.WriteLine();

            Console.WriteLine();
            Console.WriteLine("Press any key to continue...");
            Console.ReadKey();
        }

        private static double AverageErrorTopK(KeyValuePair<int, double>[]
truePageRankDesc, Vector estimatedPageRank, int k)
        {
            double totalError = 0.0;

            for (int i = 0; i < k; i++)
                totalError += Math.Abs(truePageRankDesc[i].Value -
estimatedPageRank[truePageRankDesc[i].Key]);

            return totalError / ((double)k);
        }

        private static KeyValuePair<int, double>[] ToSortedArrayDesc(double[] d)
        {
            Dictionary<int, double> dictionary = new Dictionary<int, double>(d.Length);

            for (int i = 0; i < d.Length; i++)
                dictionary.Add(i, d[i]);

            return dictionary.OrderByDescending(kvp => kvp.Value).ToArray();
        }

        private static Tuple<Vector, KeyValuePair<int, double>[], TimeSpan>
RandomWalk(int r)
        {
            // for each source node, stores a list of destination nodes
            List<int>[] graph = new List<int>[n];
            for (int i = 0; i < n; i++)
                graph[i] = new List<int>();
```

```
            // populate the graph data structure
            using (StreamReader reader = new StreamReader(graphPath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    string[] elements = line.Split();

                    int source = int.Parse(elements[0]);
                    int destination = int.Parse(elements[1]);

                    graph[source - 1].Add(destination);
                }
            }

            // stores how many times each node is visted in the MC algorithm
            int[] visits = new int[n];
            for (int i = 0; i < n; i++)
                visits[i] = 0;

            Random random = new Random();

            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            // perform the random walk r times
            for (int i = 0; i < r; i++)
            {
                // from each node
                for (int j = 0; j < n; j++)
                {
                    int currentNode = j + 1;

                    do
                    {
                        // increment the # of times we've visited the current node
                        visits[currentNode - 1]++;

                        // travel to the next node with P(beta) and exit the random walk
with P(1-beta)
                        if (random.NextDouble() < beta)
                        {
                            // get the next node in the random walk
                            if (graph[currentNode - 1].Count == 0)
                                // exit if we hit a dangling node
                                break;
                            else
                                // select the next node uniformly at random
                                currentNode = graph[currentNode -
1][random.Next(graph[currentNode - 1].Count)];
                        }
                        else
                            break;
                    } while (true);
                }
            }

            stopwatch.Stop();
```

```
            // calculate Rj, the estimated PageRank
            double[] rj = new double[n];
            for (int i = 0; i < n; i++)
                rj[i] = ((double)visits[i]) * ((1.0 - beta) / ((double)(n * r)));

            // return the results
            return new Tuple<Vector, KeyValuePair<int, double>[], 
TimeSpan>(Vector.Create(rj), ToSortedArrayDesc(rj), stopwatch.Elapsed);
        }

        private static Tuple<Vector, KeyValuePair<int, double>[], TimeSpan> PageRank()
        {
            // create M
            Matrix M = Matrix.Create(n, n);

            // populate M with 1.0 for each edge
            using (StreamReader reader = new StreamReader(graphPath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    string[] elements = line.Split();

                    int source = int.Parse(elements[0]);
                    int destination = int.Parse(elements[1]);

                    // increment Mji
                    M[destination-1, source-1] += 1.0;
                }
            }

            // make M column stochastic
            for (int i = 0; i < n; i++)
            {
                Vector column = M.GetColumn(i);
                double sum = column.Sum();

                for (int j = 0; j < n; j++)
                    column[j] /= sum;
            }

            // create and initialize teleport vector S
            Vector S = Vector.Create(n);
            for (int i = 0; i < n; i++)
                S[i] = 1.0 / (double)n;

            // create and initialize r (r0)
            Vector r = Vector.Create(n);
            for (int i = 0; i < n; i++)
                r[i] = 1.0 / (double)n;


            // time 40 power iterations
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            for (int i = 0; i < 40; i++)
```

```
                r = (beta * (M * r)) + ((1 - beta) * S);

            stopwatch.Stop();

            return new Tuple<Vector, KeyValuePair<int, double>[], TimeSpan>(r,
ToSortedArrayDesc(r.ToArray()), stopwatch.Elapsed);
        }
    }
}
```

```
% Philip Scuderi
% Stanford University
% Winter 2013
% CS 246
% Homework 3
% Question 3

function [] = Q3
    M = [0 1 1 0 1; 0 1 0 1 0; 1 0 0 0 0];
    [M_A1,M_B1] = Iterate(M,1);
    M_A1 %#ok<*NOPRT>
    M_B1
    [M_A2,M_B2] = Iterate(M,2);
    M_A2
    M_B2
    [M_A3,M_B3] = Iterate(M,3);
    M_A3
    M_B3

    K21 = [1; 1];
    [K21_A1,K21_B1] = Iterate(K21,1);
    K21_A1
    K21_B1
    [K21_A2,K21_B2] = Iterate(K21,2);
    K21_A2
    K21_B2
    [K21_A3,K21_B3] = Iterate(K21,3);
    K21_A3
    K21_B3

    K22 = [1 1; 1 1];
    [K22_A1,K22_B1] = Iterate(K22,1);
    K22_A1
    K22_B1
    [K22_A2,K22_B2] = Iterate(K22,2);
    K22_A2
    K22_B2
    [K22_A3,K22_B3] = Iterate(K22,3);
    K22_A3
    K22_B3
end

function [A,B]=Iterate(M,k)
    A = eye(size(M,1));
    B = eye(size(M,2));
    for i=1:k
        nextA = NextA(M,A,B);
        nextB = NextB(M,A,B);
        A = nextA;
        B = nextB;
    end
end

function A1=NextA(M,A0,B0)
    A1 = A0;
    for i=1:length(A0)
```

```
        for j=1:length(A0)
            if i~=j
                A1(i,j) = Sa(M,i,j,B0);
                A1(j,i) = A1(i,j);
            end
        end
    end
end

function B1=NextB(M,A0,B0)
    B1 = B0;
    for i=1:length(B0)
        for j=1:length(B0)
            if i~=j
                B1(i,j) = Sb(M,i,j,A0);
                B1(j,i) = B1(i,j);
            end
        end
    end
end

function a=Sa(M,X,Y,B)
    C1 = 0.8;
    a = 0.0;
    for i=1:sum(O(M,X))
        for j=1:sum(O(M,Y))
            a = a + B(IdxIthNonZeroElement(i, O(M,X)), IdxIthNonZeroElement(j,
O(M,Y)));
        end
    end
    a = a * (C1 / (sum(O(M,X)) * sum(O(M,Y))));
end

function b=Sb(M,X,Y,A)
    C2 = 0.8;
    b = 0.0;
    for i=1:sum(I(M,X))
        for j=1:sum(I(M,Y))
            b = b + A(IdxIthNonZeroElement(i, I(M,X)), IdxIthNonZeroElement(j,
I(M,Y)));
        end
    end
    b = b * (C2 / (sum(I(M,X)) * sum(I(M,Y))));
end

function O=O(M,X)
    O = M(X,:);
end

function I=I(M,X)
    I = M(:,X);
end

function IdxIthNonZeroElement=IdxIthNonZeroElement(i, OX)
    IdxIthNonZeroElement = 0;
    count = 0;
```

```
    for j=1:length(OX)
        if OX(j)~=0
            count = count + 1;
            if count == i
                IdxIthNonZeroElement = j;
            end
        end
    end
end
```

```
/*
 * Philip Scuderi
 * Stanford University
 * CS246
 * Winter 2013
 * Homework 3
 * Question 4
 */

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using System.Diagnostics;

namespace HW3_Q4
{
    class Program
    {
        static void Main(string[] args)
        {
            Stopwatch stopwatch = new Stopwatch();
            stopwatch.Start();

            Q4 q4 = new Q4("../../livejournal-undirected.txt");

            // 4(c)(i)
            Console.WriteLine("epsilon\ti");
            foreach (double epsilon in (new double[] { 0.1, 0.5, 1.0, 2.0 }))
                Console.WriteLine("{0}\t{1}", epsilon, q4.FindInducedSubgraph(epsilon,
true).Item2.Count - 1);

            // 4(c)(ii)
            Console.WriteLine("\ni\tp(Si)\t|E(Si)|\t|Si|");
            int i = 0;
            foreach (Tuple<double, int, int> r in q4.FindInducedSubgraph(0.05,
true).Item2)
                Console.WriteLine("{0}\t{1}\t{2}\t{3}", i++, r.Item1, r.Item2, r.Item3);

            // 4(c)(iii)
            List<Tuple<ISet<int>, IList<Tuple<double, int, int>>>> results = new
List<Tuple<ISet<int>, IList<Tuple<double, int, int>>>>(20);
            for (int j = 0; j < 20; j++)
            {
                var resultJ = q4.FindInducedSubgraph(0.05);
                results.Add(resultJ);
                q4.DeleteVerticies(resultJ.Item1);
            }

            Console.WriteLine("\n\nj\tp(Si)\t|E(Si)|\t|Si|");
            for (int j = 0; j < 20; j++)
            {
                int numInducedEdges = q4.NumInducedEdges(results[j].Item1);
                int sizeOfS = results[j].Item1.Count;
                double rho = ((double)numInducedEdges) / ((double)sizeOfS);
                Console.WriteLine("{0}\t{1}\t{2}\t{3}", j+1, rho, numInducedEdges,
sizeOfS);
```

```
            }

            stopwatch.Stop();

            Console.WriteLine("\n\nTotal Time = {0}\nPress any key to continue...",
stopwatch.Elapsed);
            Console.ReadKey();
        }
    }

    public class Q4
    {
        protected HashSet<int> V;
        protected string graphPath;

        public Q4(string graphPath)
        {
            this.graphPath = graphPath;
            V = new HashSet<int>();

            using (StreamReader reader = new StreamReader(graphPath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    string[] elements = line.Split();

                    int v1 = int.Parse(elements[0]);
                    int v2 = int.Parse(elements[1]);

                    V.Add(v1);
                    V.Add(v2);
                }
            }
        }

        public void DeleteVerticies(ISet<int> S)
        {
            V.ExceptWith(S);
        }

        public Tuple<ISet<int>, IList<Tuple<double, int, int>>>
FindInducedSubgraph(double epsilon, bool resultsPerIteration = false)
        {
            // the results at each iteration: p(Si), |E(Si)|, and |Si|
            List<Tuple<double, int, int>> iterativeResults = null;

            if (resultsPerIteration)
                iterativeResults = new List<Tuple<double, int, int>>();

            HashSet<int> S = new HashSet<int>(V);
            HashSet<int> tildeS = new HashSet<int>(V);

            while (S.Count != 0)
            {
                if (resultsPerIteration)
                {
                    if (iterativeResults.Count == 0)
```

```
                        // track iterative results at iteration 0, i.e., before we begin
                        iterativeResults.Add(new Tuple<double, int, int>(Rho(S), 
NumInducedEdges(S), S.Count));
                }

                // A(S) := {i ∈ | degS(i) <= 2(1+ϵ)p(S)}
                HashSet<int> AS = A(S, epsilon);

                // S <- S \ A(S)
                S.ExceptWith(AS);

                // if p(S) > p(~S)
                if (Rho(S) > Rho(tildeS))
                {
                    // ~S <- S
                    tildeS = new HashSet<int>(S);
                }

                if (resultsPerIteration)
                {
                    // track the results at the end of each iteration (i = 1, 2, ...)
                    iterativeResults.Add(new Tuple<double, int, int>(Rho(S), 
NumInducedEdges(S), S.Count));
                }
            }

            return new Tuple<ISet<int>, IList<Tuple<double, int, int>>>(tildeS, 
iterativeResults);
        }

        protected HashSet<int> A(HashSet<int> S, double epsilon)
        {
            HashSet<int> A = new HashSet<int>();

            Dictionary<int, int> degS = deg(S);
            double rhoS = Rho(S);

            foreach (int i in S)
                if (degS[i] <= 2.0 * (1.0 + epsilon) * rhoS)
                    A.Add(i);

            return A;
        }

        protected Dictionary<int, int> deg(HashSet<int> S)
        {
            Dictionary<int, int> degS = new Dictionary<int, int>(S.Count);

            foreach (int i in S)
                degS.Add(i, 0);

            using (StreamReader reader = new StreamReader(graphPath))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    string[] elements = line.Split();
```

```
                int v1 = int.Parse(elements[0]);
                int v2 = int.Parse(elements[1]);

                if (S.Contains(v1) && S.Contains(v2))
                {
                    degS[v1]++;
                    degS[v2]++;
                }
            }
        }

        return degS;
    }

    public int NumInducedEdges(ISet<int> S)
    {
        int inducedEdgeCount = 0;

        using (StreamReader reader = new StreamReader(graphPath))
        {
            string line;
            while ((line = reader.ReadLine()) != null)
            {
                string[] elements = line.Split();

                int v1 = int.Parse(elements[0]);
                int v2 = int.Parse(elements[1]);

                if (S.Contains(v1) && S.Contains(v2))
                    ++inducedEdgeCount;
            }
        }

        return inducedEdgeCount;
    }

    public double Rho(ISet<int> S)
    {
        return ((double)NumInducedEdges(S)) / ((double)S.Count);
    }
  }
}
```