Khoa Do December 2, 2019

ECGR 4187-01 Dr. Asis Nasipuri

Fall 2019 Software Project Report

Objective

The overall objective of this project is to apply Dijkstra's algorithm to analyze and evaluate the adaptive link costs and the effect of adaptation rate regarding the stability and the performance of any network. There are two parts to be completed in this project. The first part is to download and use the sample C code from Canvas that implements Dijkstra's algorithm to analyze the network configuration shown in figure 1. The second part is to modify the provided code to adapt to and analyze the network configuration shown in figure 2.

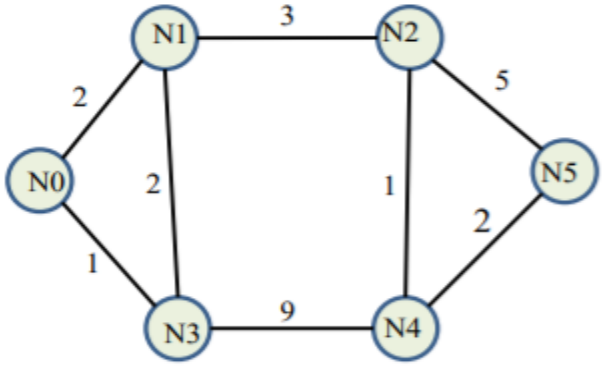


Figure 1. 6-Node Network.

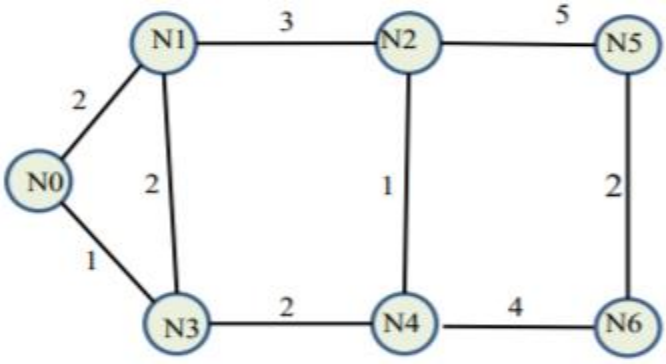


Figure 2. 7-Node Network.

Part I

Part I is to use the program to learn, understand, and analyze the 6-node network shown in figure 1.

a)

This part is to download the sample C code that implements the Dijkstra's algorithm, compile, and run it. The goal is to understand how the code work. The C code was run using the Dev-C++ compiler and the following figure shown the starting window.

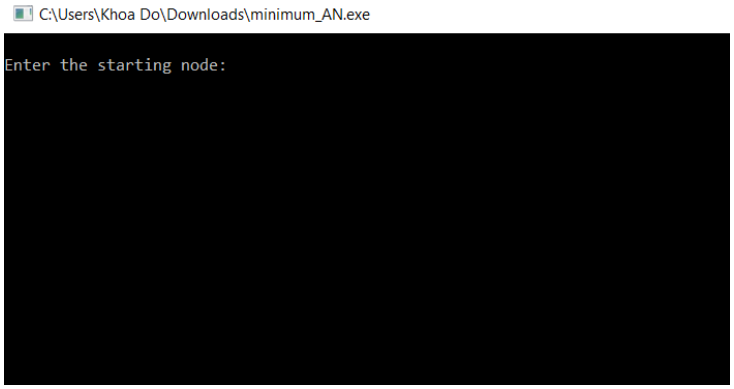


Figure 3. Excuted Sample Code.

The starting node can be any positive interger number. The 6-node network is defined by by 6-by-6 matrix showing all connections between one node and the other nodes with specific values (distance values). If there is no connection between two nodes, the value would be infinite. The six columns in the matrix shows the distacne values for N0, N1, N2, N3, N4, and N5 respectively. The first row of in the matrix shows the starting node is N0, second row will be N1, etc repsectivly.

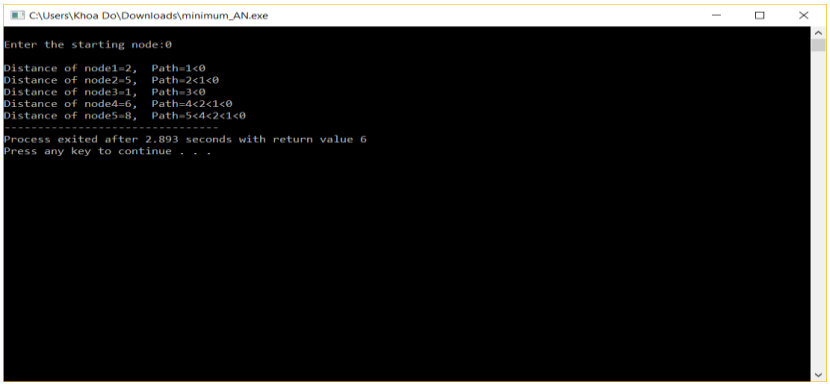
```
3   #define INFINITE 9999
4   #define MAX 6
5
6   void dijkstra(int G[MAX][MAX],int startnode,int *pred, int *distance);
7     1/ 2  void dijkstra (int G[MAX][MAX], int startnode, int *pr
8
9   {
10      int G[MAX][MAX]=
11       {{INFINITE,2,INFINITE,1,INFINITE,INFINITE},
12        {2,INFINITE,3,2,INFINITE,INFINITE},
13        {INFINITE,3,INFINITE,INFINITE,1,5},
14        {1,2,INFINITE,INFINITE,9,INFINITE},
15        {INFINITE,INFINITE,1,9,INFINITE,2},
16        {INFINITE,INFINITE,5,INFINITE,2,INFINITE}};
17
18      int n=6,u;
19      int i,j, pred[MAX]={0}, distance[MAX];
```

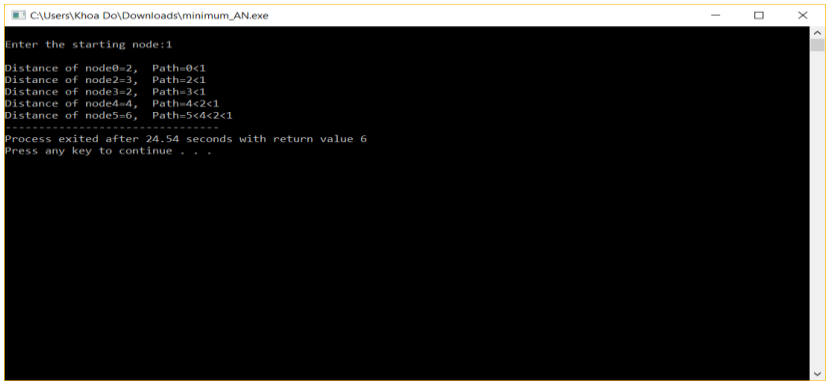
Figure 4. Network's Definition.

For example, there are six node total in figure 1, then line 4 and 18 should be six. Starting at node 0 (N0, first row of the matrix), N0 is connected N1 and N3 with the distance values of 2 and 1. The conenction between N0 and itself is INF and there are not any connection between N0 and N2/N4/N5, the values are INF. Therefore, the first row's vslues for six columns should be {INF,2,INF,1,INF,INF}, which shown in figure 4. The rest of the code runs loops to calculate the shortest path between the starting node and the rest of the nodes. The matrix and the values in line 4 and 8 will need to be modified for different network configuretions.

b)

After running the code for the starting nodes from N0 to N5. The following results were recorded.





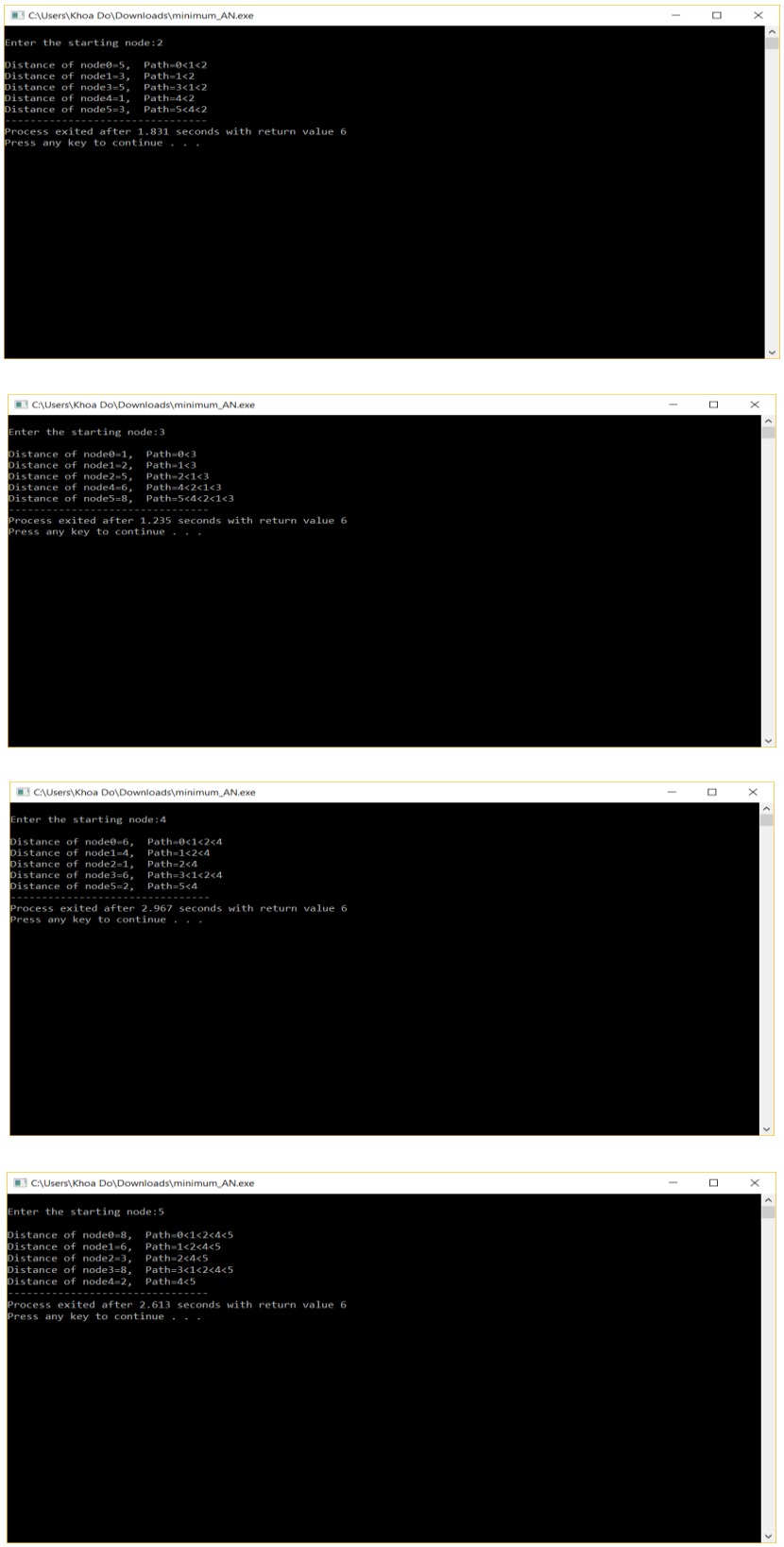


Figure 5. Results I(a) for N0-N5.

From the results above, this following table was obtained.

		To					
		0	**1**	**2**	**3**	**4**	**5**
From	**0**	-	0	1	0	2	4
	1	1	-	1	1	2	4
	2	1	2	-	1	2	4
	3	3	3	1	-	2	4
	4	1	2	4	1	-	4
	5	1	2	4	1	5	-

Table 1. Centralized Routing Table Corresponding to the Least Costs.

The results above were compared to the expected reuslts obtained by hand and they were the same.

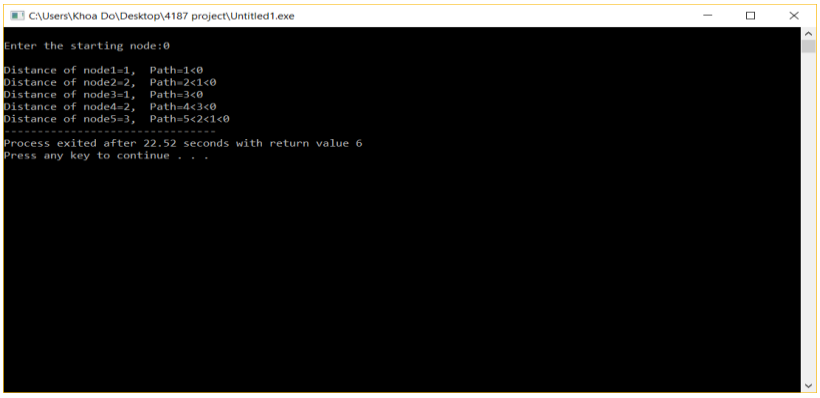
c)

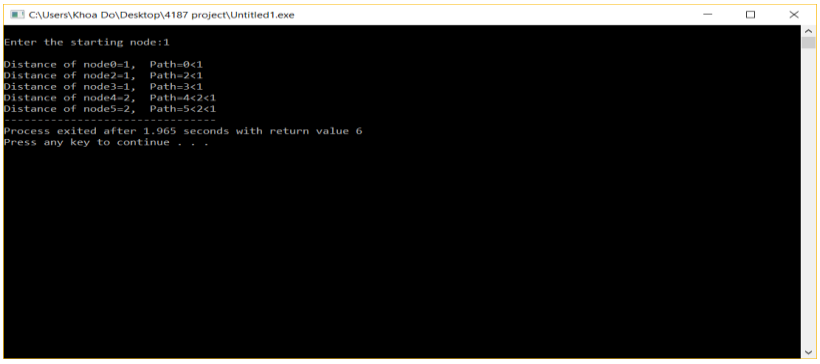
Part c is similar to part b, which is to obtained the centralized routing table but the only difference is that it has to follow the least-hop routes. The least-hop routes mean the least numer of transmissions (intermediates) taken to send data from the source to the destination. The algorithm basically finds the shortest traveling path without worrying about the least cost. Therefore, the simplest solution to this is to change all the distance (cost) values in the matrix to be the same, in this case, 1's; in other words, all the numbers shown in figure 1 will be 1's. The INF's will be kept the same if there are not any connection between two nodes. With that, the new matrix was obtained and shown below.

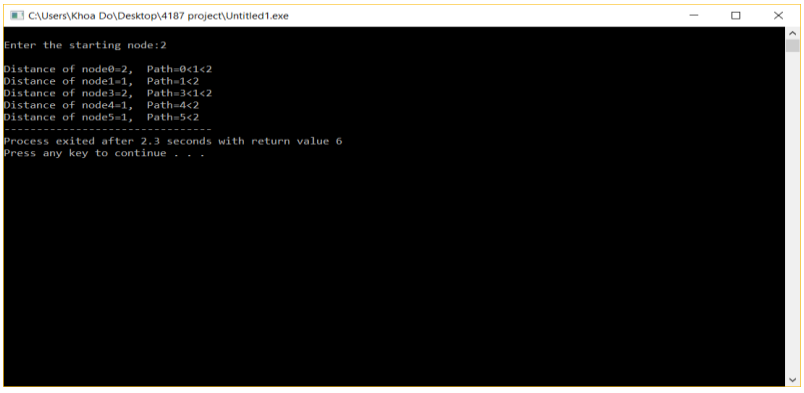
```
10      int G[MAX][MAX]=
11        {{INFINITE,1,INFINITE,1,INFINITE,INFINITE},
12         {1,INFINITE,1,1,INFINITE,INFINITE},
13         {INFINITE,1,INFINITE,INFINITE,1,1},
14         {1,1,INFINITE,INFINITE,1,INFINITE},
15         {INFINITE,INFINITE,1,1,INFINITE,1},
16         {INFINITE,INFINITE,1,INFINITE,1,INFINITE}};
17
```

Figure 6. New Matrix for Part I (b).

Running the same code with the new matrix, the following results were obtained.







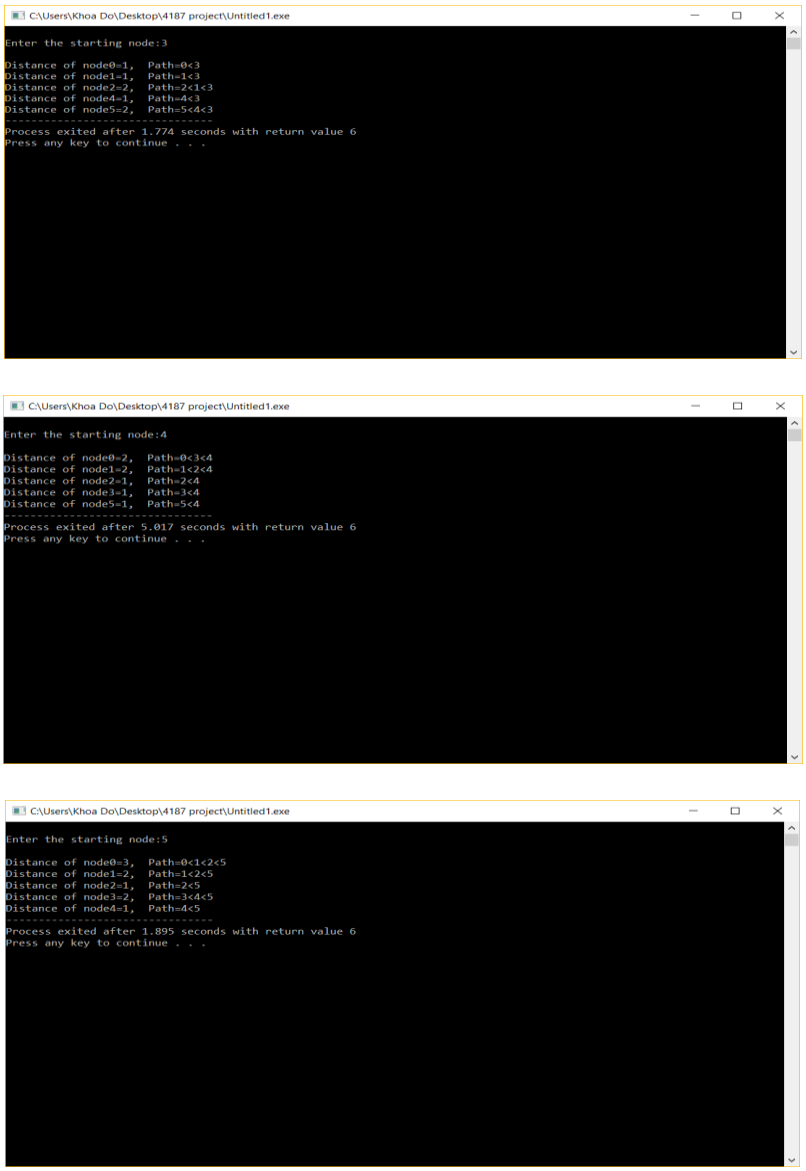


Figure 7. Results I(c) for N0-N5.

From the results above, this following table was obtained.

		To					
		0	**1**	**2**	**3**	**4**	**5**
From	**0**	-	0	1	0	3	2
	1	1	-	1	1	2	2
	2	1	2	-	1	2	2

	3	3	3	1	-	3	4
	4	3	2	4	4	-	4
	5	1	2	5	4	5	-

Table 2. Centralized Routing Table Corresponding to the Least-Hop.

From N0 to N5, the path can be either 0-1-2-5 or 0-3-4-5 because these two paths have the same amount of "hop." According to table 2, it took the path 0-1-2-5. The same thing happened to the path from N5 to N0.

Part II

Part II is to modifly the provided code to adapt to the new network confiuration shown in figure 2. In addtion, an algorithm will be added on to the code to perform the link load computation and cost adaption, knowing that N0-N6 are sources and N0 is the destination.

a)

The first part is to initialize the link cost matrix, which basically modifying the matrix like it has been done in part I(b).

```
55      float p=0.0;
56      int answer;
57      int w[MAX][MAX] =
58
59      {{INFINITE,2,INFINITE,1,INFINITE,INFINITE,INFINITE},
60       {2,INFINITE,3, 2,INFINITE,INFINITE,INFINITE},
61       {INFINITE,3,INFINITE,INFINITE,1,5,INFINITE},
62        {1,2,INFINITE,INFINITE,2,INFINITE,INFINITE},
63        {INFINITE,INFINITE,1,2,INFINITE,INFINITE,4 },
64        {INFINITE,INFINITE,5,INFINITE,INFINITE,INFINITE,2},
65        {INFINITE,INFINITE,INFINITE,INFINITE,4,2,INFINITE}};
66
```

Figure 8. Maxtrix of the New Network in Figure 2.

The new network has seven nodes and new connections. The 7-by-7 matrix is built. Seven columns are the cost values for N0 to N6 respectively and seven rows are starting node N0 to N6. If there are not sny connections between any nodes, the column's value will be INF.

b)

Part b is use Dijkstra's again with the new matrix (part a) that adapts to the new network configuration to obtain the least-cost routes from all nodes to N0. There are some major modifications made to the code. After modifying the matrix, a nested for loop is added right after to make the cost array identical to the matrix.

```
float cost[MAX][MAX];
float L[MAX][MAX];
int j,k;

for ( j=0; j<MAX; j++ )
    for (k=0; k<MAX; k++ )
        cost[j][k]= w[j][k];
```

Figure 9. Nested for loop for Cost.

The "void dijkstra" is modified and renamed as "shortpath" fucntion. This function calculates all least-cost routes from all other nodes to N0.

```
80      for (i=0; i<MAX; i++)
81          shortpath(cost,preced, distance);
```

Figure 10. Shortpath Function.

To ouput the least-cost routes, the following block of code is added.

```
82  for (i=0; i<MAX; i++ ) {
83      int j=i ;
84      printf("Least cost route from %d = %d ", i+1, i+1);
85      do {
86          printf("- %d " , preced[j]+1);
87          j=preced[j];
88      } while (j!= 0);
89      printf("\n");
```

Figure 11. Least-Cost Ouput Code.

The loop above first outputs the source node and then enters the do-while loop, which outputs least-cost routes located inside the preced array until N0 is reached. It then exits the do-while loop, iterates the for loop, and repeats this process for all nodes in the network. The

following table was obtained as the result of all least-cost routes from all other nodes to N0 shown in figure 2.

		From						
		0	**1**	**2**	**3**	**4**	**5**	**6**
To	**0**	-	1,0	2,4,3,0	3,0	4,3,0	5,2,4,3,0	6,4,3,0

Table 3. Least-Cost Routes to N0 in Figure 2.

c)

This part is to develop an algorithm that computes the link load – number of transmissions on links (2,1) and (4-3) specified in part e. N0 will be the destination and other nodes will be sources.

```
91  for (i=0; i<MAX; i++)
92  {
93      int x=i;
94      if (distance[i]==0);
95      else
96      do{
97          L[x][preced[x]]=L[x][preced[x]]+1;
98          x=preced[x];
99      }while (x!=0);
```

Figure 12. Link Load Algorithm.

First, it checks if the route cost is 0 or not; if not, it will execute the link load algorithm in the do-while loop because there is no need to calculate the cost of N0 going to itself. The Variable x that has been initialized on line 93 is used again in the do-while loop to go through all the nodes in the network untill it gets to N0. As the same time, the number of transimissions is incremented as x moves from one node to another.

d)

Next, an algorithm is developed to adapt link costs on links (2,1) and (4,3) specified in part e based on the number of transmissions, intial cost, and a constant p. The formula is given at G(i,j) = G(i,j) +pL(I,j).

A nested for loop is used to assign all elements in the a two-dimensional cost array (i and j) following the above formula. With routes being bidirectional, another two-dimensional cost array (j and i) is set equal the first cost array. Then, the algorithm above is put in a do-while loop so that it will be iterated until the user decides to quit the program. The two processes above are shown in figure 13 and 14.

```
113     for (i=0; i<MAX; i++)
114         for ( j=0; j<MAX; j++){
115         cost[i][j]= w[i][j]+p*(L[i][j]);
116         cost[j][i]= cost[i][j];}
```

Figure 13. Cost Adaption Algorithm.

```
127     printf("Would you like to repeat the iteration with the new cost? (1 Y / 0 N) \n");
128     scanf("%d",&choice);
129
130     }while (choice ==1);
131         return 0;
```

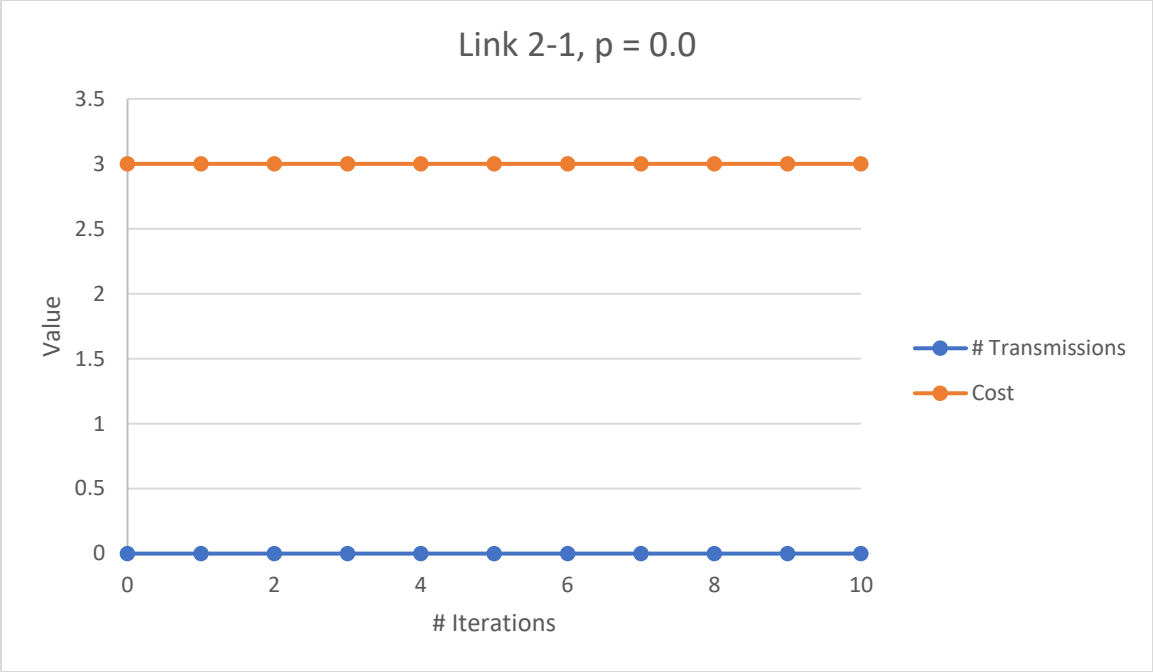
Figure 14. Program Iteration.

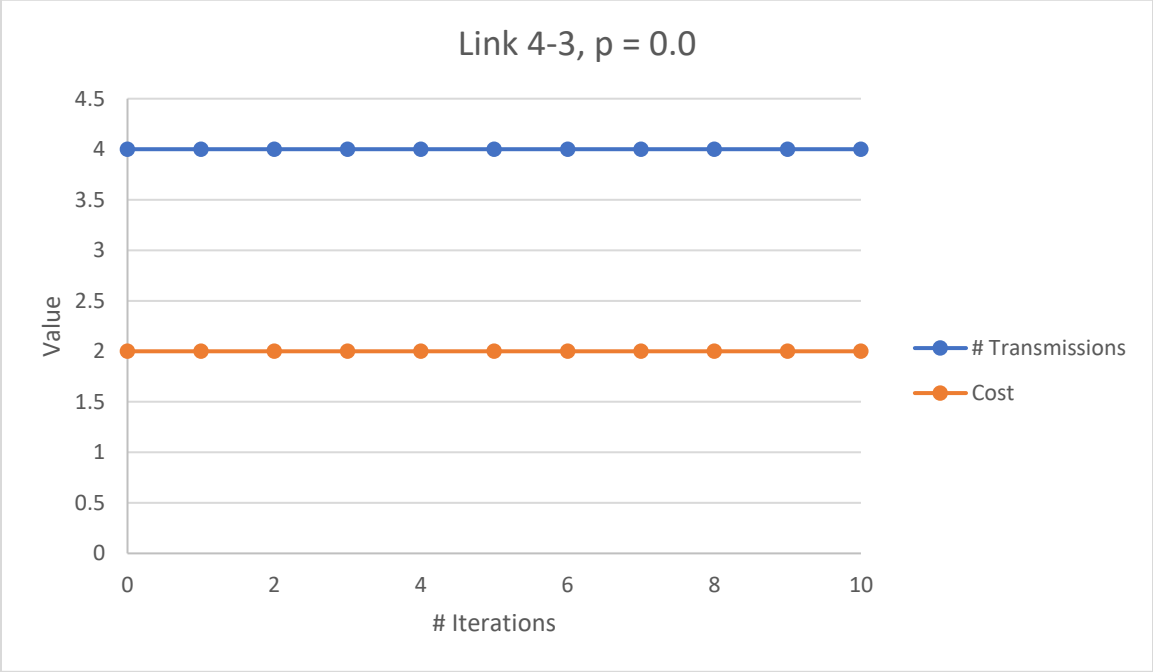
e)

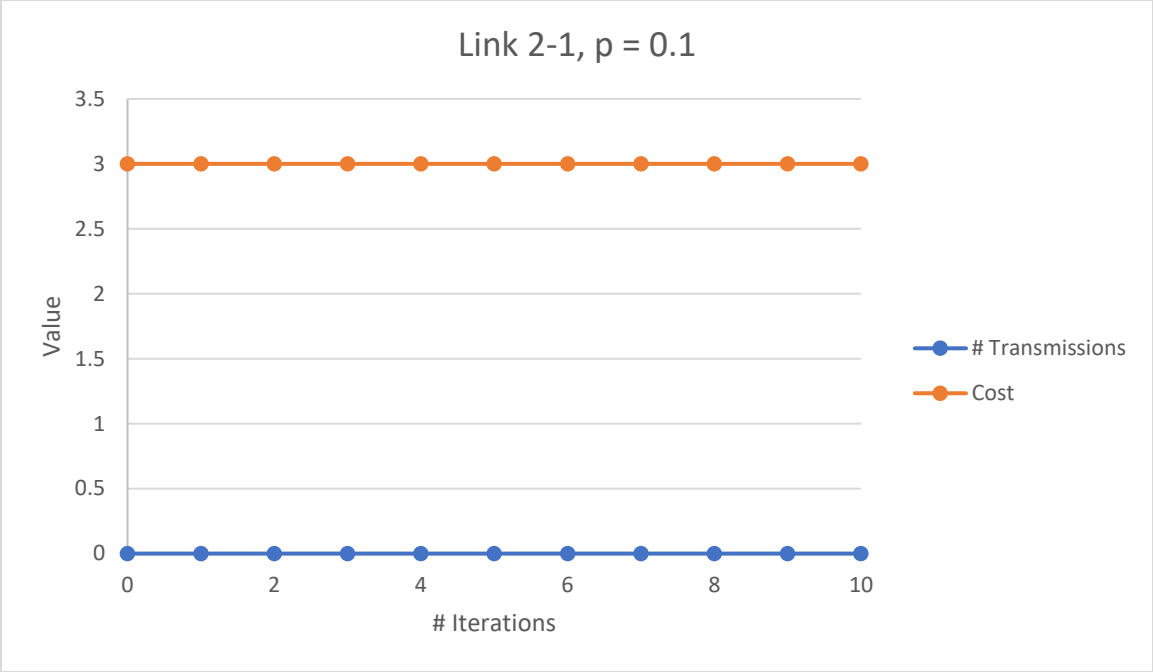
This part is to repeat b-d with values of p = 0.0, 0.1, 0.2, 0.3, 0.4, and 0.5 on links (2,1) and (4,3) with an iterations of 10.

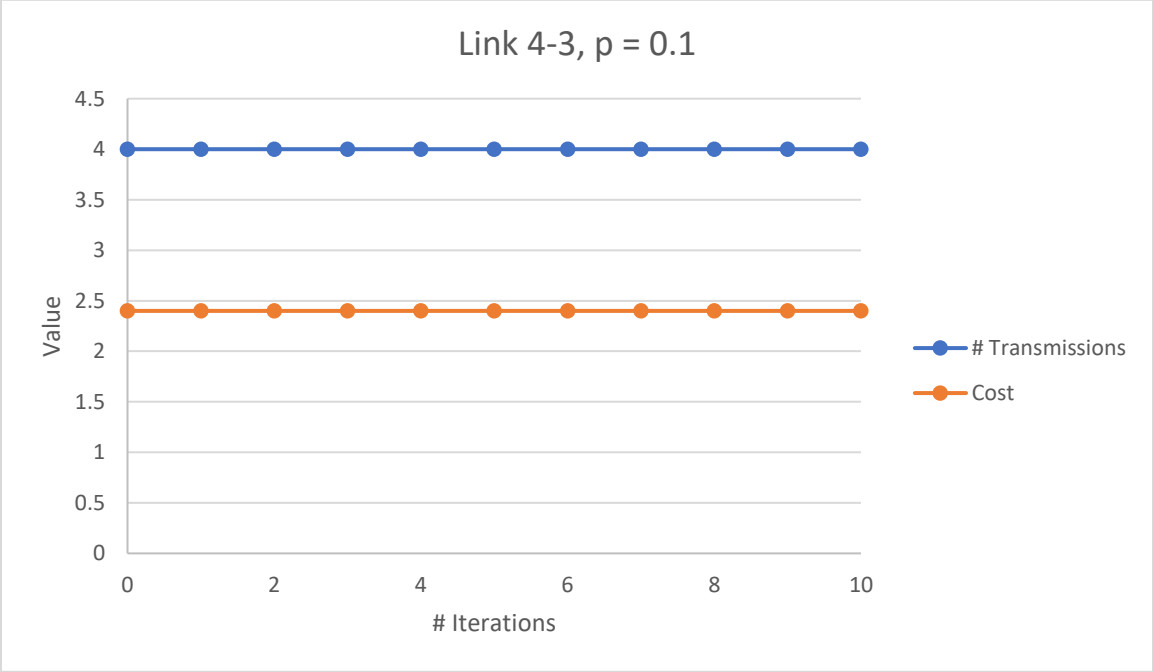
f)

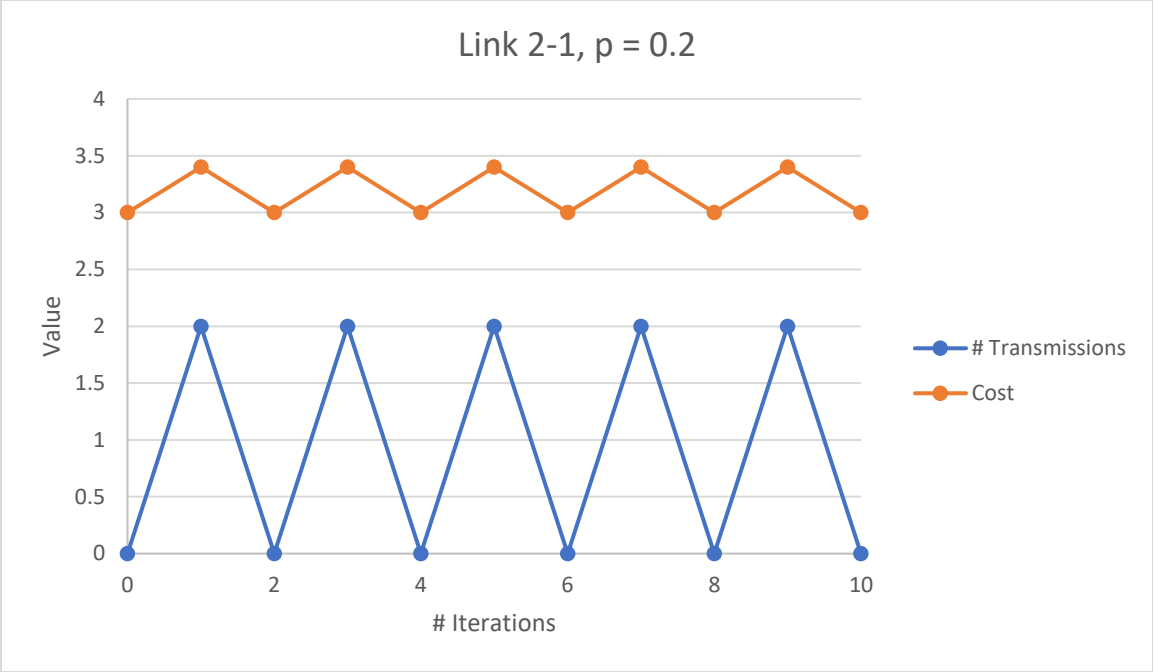
Going though steps b-d, the following plots were obtained.

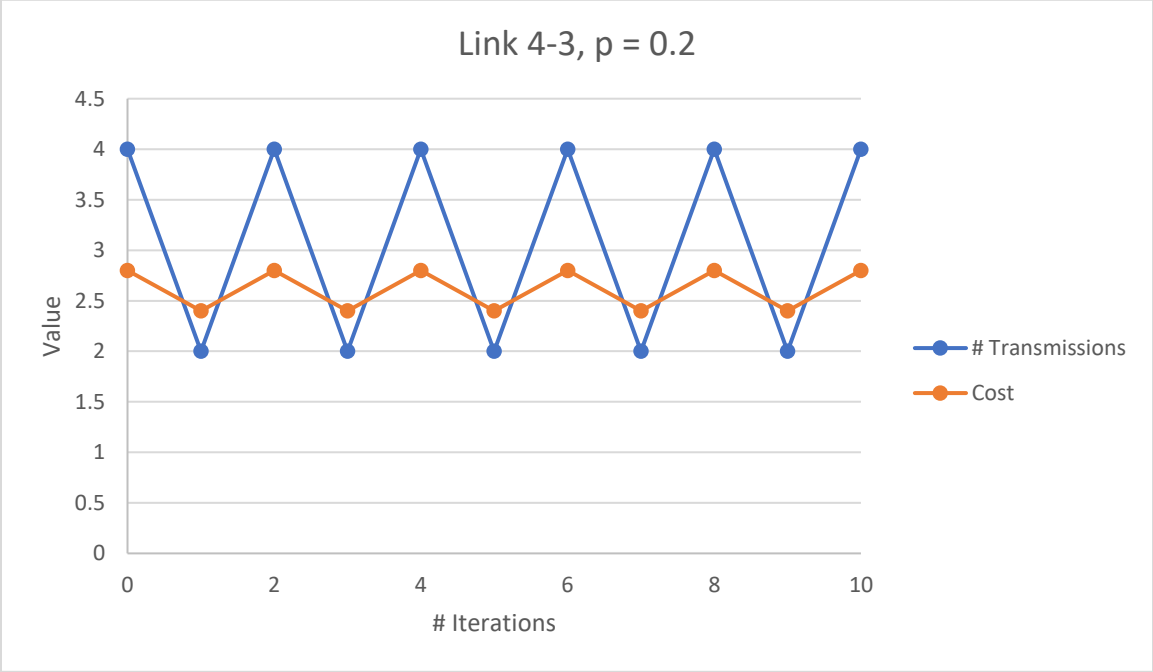

Link 2-1, p = 0.0
3.5
3
2.5
2
1.5
1
0.5
0
Value
0
2
4
6
8
10
Iterations
Transmissions
Cost

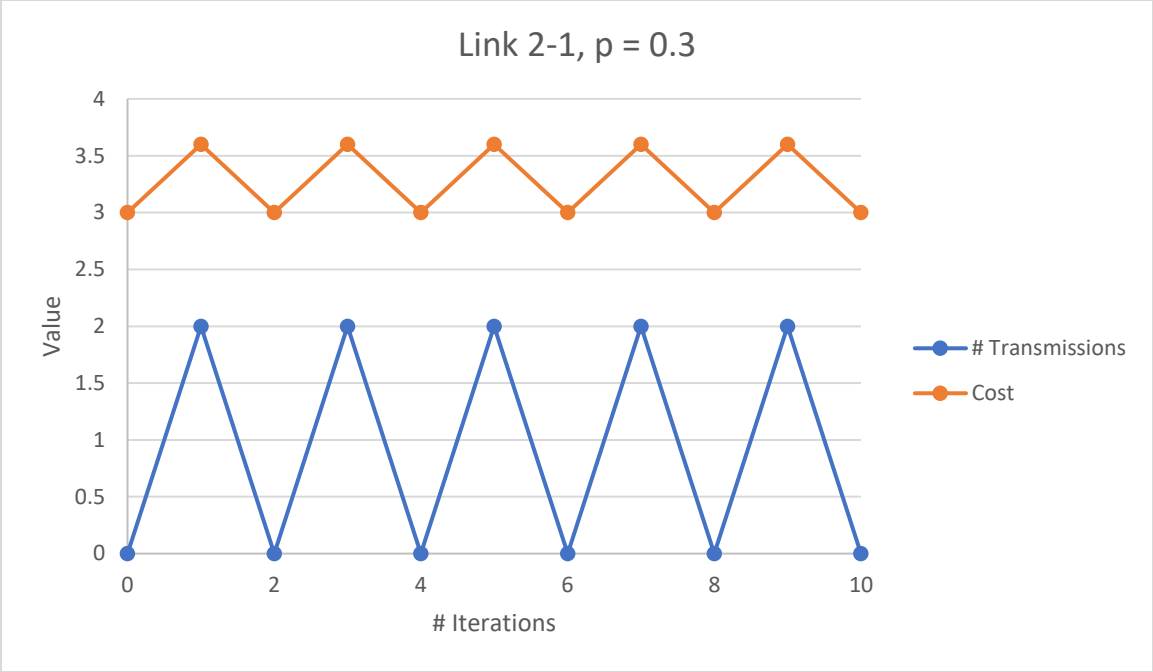

Link 4-3, p = 0.0
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Value
0
2
4
6
8
10
Iterations
Transmissions
Cost

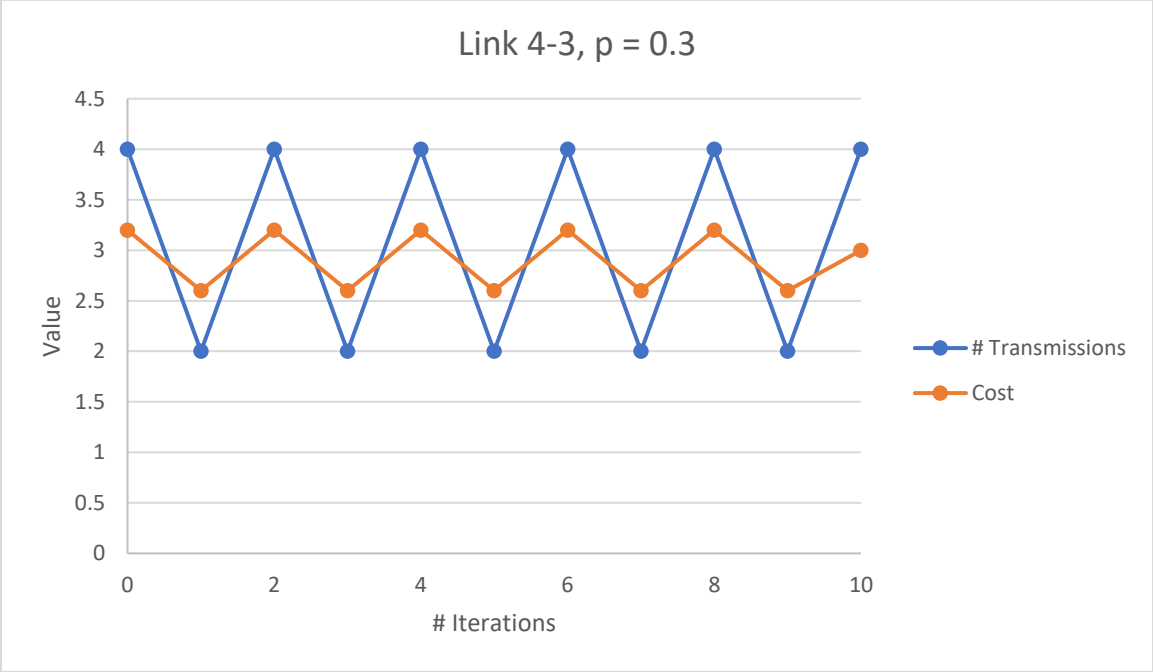

Link 2-1, p = 0.1
3.5
3
2.5
2
1.5
1
0.5
0
Value
0
2
4
6
8
10
Iterations
Transmissions
Cost

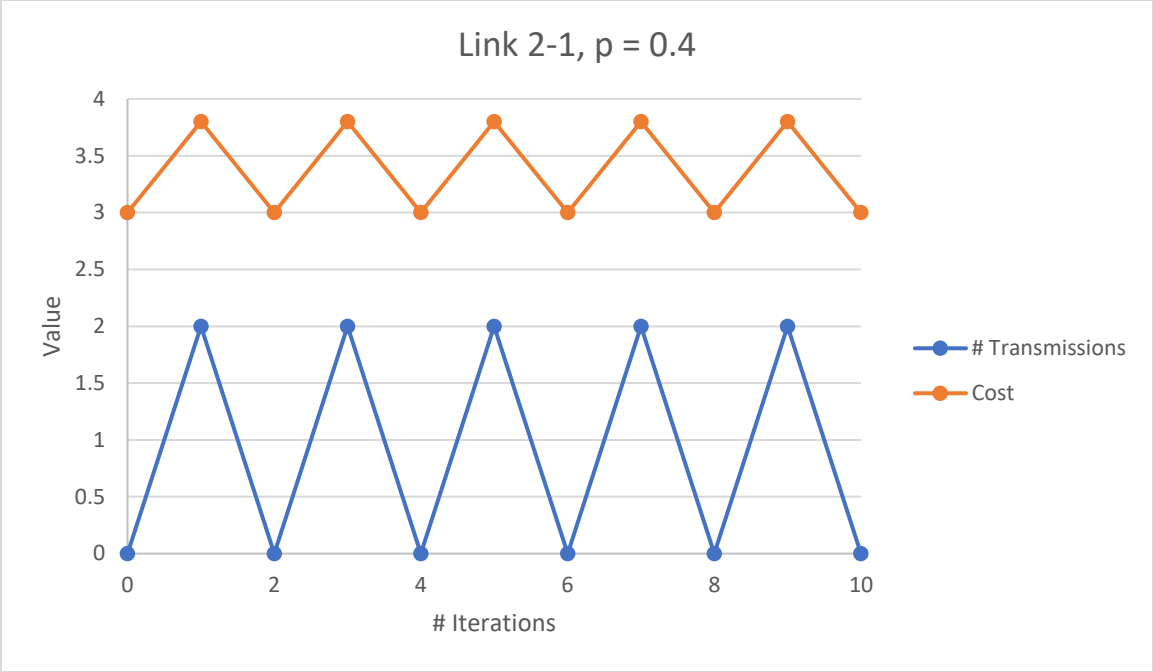

Link 4-3, p = 0.1
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
Value
0
2
4
6
8
10
Iterations
Transmissions
Cost

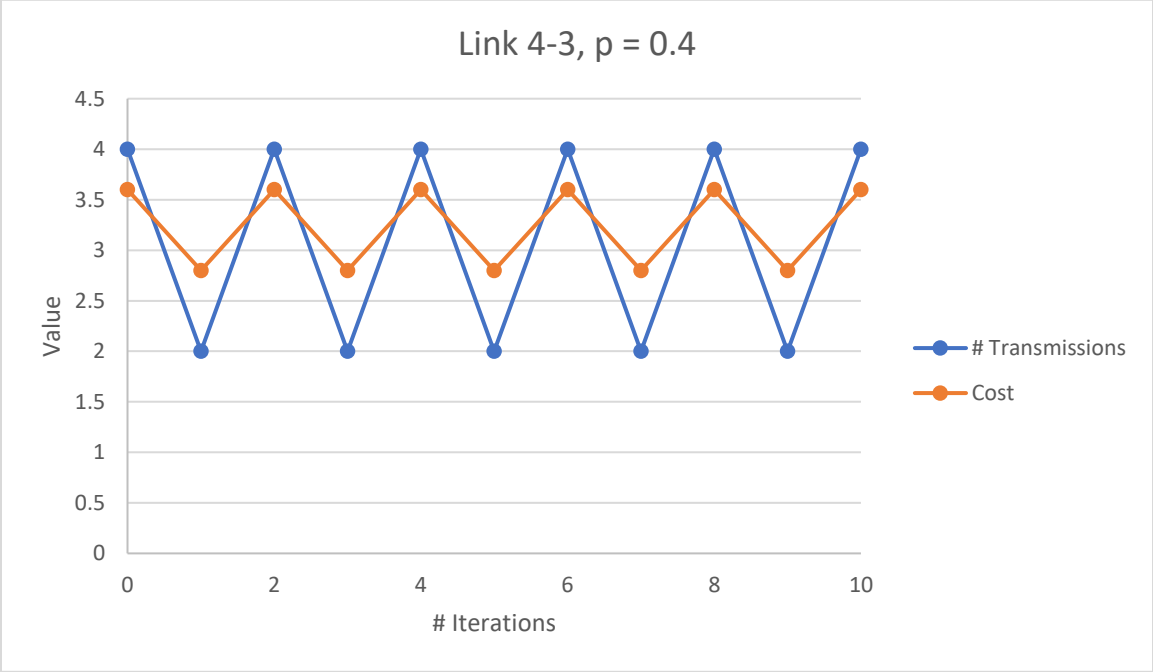

Link 2-1, p = 0.2
Value
Iterations
Transmissions
Cost
0
0.5
1
1.5
2
2.5
3
3.5
4
0
2
4
6
8
10

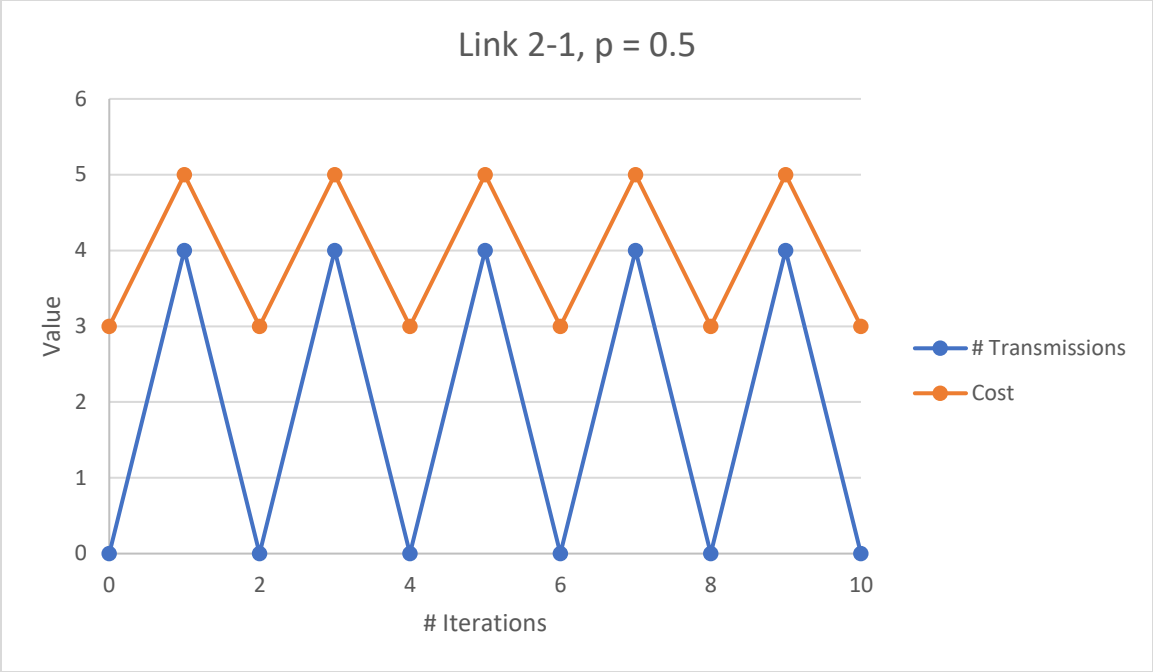

Link 4-3, p = 0.2
Value
Iterations
Transmissions
Cost
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
2
4
6
8
10

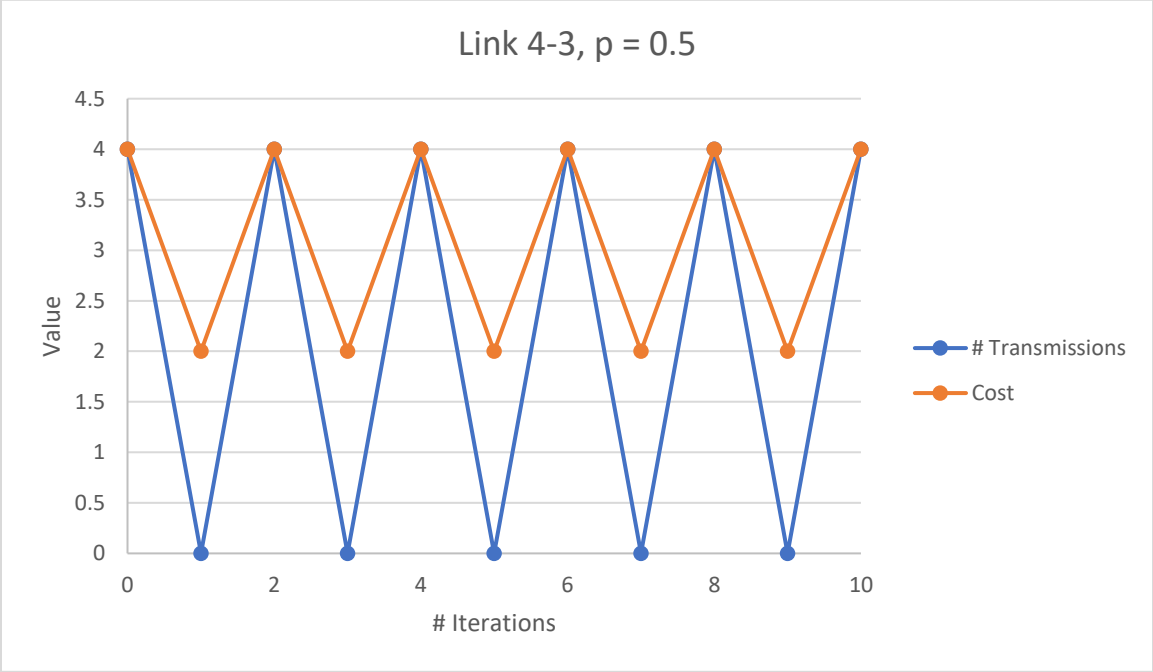

Link 2-1, p = 0.3
Value
Iterations
Transmissions
Cost
0
0.5
1
1.5
2
2.5
3
3.5
4
0
2
4
6
8
10


Link 4-3, p = 0.3
Value
Iterations
Transmissions
Cost
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
2
4
6
8
10


Link 2-1, p = 0.4
Value
Iterations
Transmissions
Cost
0
0.5
1
1.5
2
2.5
3
3.5
4
0
2
4
6
8
10


Link 4-3, p = 0.4
Value
Iterations
Transmissions
Cost
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
0
2
4
6
8
10





Looking through all of the plots. It is easy to tell that number of tranmissions and link cost oscilates over time for each iteration. When the link cost is low, the number of transmisisons is high and when the link cost is high, the number of tranmissions is low. With the low cost – high tranmisison, traffic will start to build up and congestion will occur. When

congestion increases, the link cost increases and causes the cost-adapting routing mechanism to execute to bring transmissions to lower-cost links. The low-cost links then have less congestion and become more favorable for high number of transmissions. However, when p = 0.0 and 0.1, links 2-1's and 4-3's cost and number of transmissions remain constant because they never use for data tranmission; the link costs are too high and become less favorable for transmissions. It appears that link 4-3 performs stably and well when p = 0.2, 0.3, and 0.4.

Conclusion

This project helped gain better understanding of the Dijkstra's algorithm and its application to analyze and evaluate any network in terms of link costs, number of transmissions, and their relationship. Using the modified algorithm for part II, it was viable to see the impacts of cost adaption to the data transmission. Links in network will oscillate due to costant switiching between high and low congestion; and this is a result of the cost-adapting mechanism constantly switching between low-cost and high-cost link. The projects also showed that with different values of p, the link cost and transmission relationship would vary and that link 4-3 performed better than link 2-1. Overall, to purpose of the project was to see how data was routed throughout a network.