

# Controller Assignment

Shadman Protik  
Talha Ibn Aziz

10/4/2018

## 1 Problem Statement

Let a network be represented by a graph  $G = (S, L)$ , where  $S$  is the set of switches or nodes and  $L$  is the list of connecting edges. This network is to be divided into  $k$  disjoint sets of switches  $S_{i:i=1,2,3,\dots}$  and a controller is to be assigned to each set  $S_i$  such that the following four conditions are met:

1. The latency from each controller to the nodes of  $S_i$  is minimized.
2. The latency from each controller to other controllers is minimized.
3. The reliability of the network is maximized.
4. The number of controllers,  $k$  needs to be minimized as much as possible (cost efficiency).

This is called the Controller Placement Problem. Therefore, there are more than one parameters which need to be optimized. Most of the works[2] defined this problem as a multi-objective combinatorial optimization problem.

Controller Assignment can be divided into two phases:

- Clustering
- Controller Selection

Clustering is the process of determining the number of controllers  $k$  and  $k$  sets of disjoint switches or 'clusters' such that the target objectives 1 and 4 mentioned in this section are reached. We work with two well known algorithms DBCP[2] and SPICi[1] that cluster a network very efficiently to reach different goals. And we propose three new algorithms G-SPICi, I-SPICi and M-SPICi which are variations of SPICi. These algorithms have been explained in details in sections 2, 3, 5, 4 and ?? respectively.

Controller selection is the method to select a controller from each cluster that has been formed by the clustering algorithms. DBCP uses a method of controller selection which is described in section 2.2.

## 2 Algorithm 1: DBCP[2]

### 2.1 Clustering

DBCP (Density Based Controller Placement) clusters a network using the local density of a node and the minimum distance to a higher density node.

$$\rho_i = \sum_j \chi(d_{ij} - d_c) \quad (1)$$

The local density (equation 1) of a node  $s$  is the count of all the nodes which are at most  $d_c$  distance away from  $s$  and is denoted by  $\rho_s$ . The threshold  $d_c$  is a distance used to set a limit to the cluster diameter and consequently to find an approximate to the optimal value of  $k$ . Here  $d_{ij}$  gives the minimum distance between nodes  $i$  and  $j$ . The value of  $\chi(x)$  is 1 only for  $d_{ij} < d_c$  that is when  $x < 0$  and is 0 otherwise. Thus  $\rho_i$  is the number of nodes that can be reached from node  $i$  by traversing at most distance  $d_c$ .

The minimum distance to a higher density node is represented by the term  $\delta_s$  for a node  $s$ . For the node with highest density there is no other node with higher density and so only for that node,  $\delta_s$  holds the distance of the farthest node from  $s$ .

For a set of nodes  $S$  with a set of edges  $L$  among them in a graph  $G = (S, L)$  they fixed a value of  $k$ , that is the number of clusters to be formed each with one controller. They formed the clusters and selected a node as controller based on the values of  $\rho_s$  and  $\delta_s$ . The more the values  $\rho_s$  and  $\delta_s$  for a node  $s$ , the more it's chances are for being selected as the controller. The method for selecting the cluster head or controller of each sub-network is explained further in detail in the section 2.2.

---

**Algorithm 1** : Density Based Controller Placement

---

```
1: procedure DBCP(S,L)
2:  $k \leftarrow 0$ 
3: for  $s$  in  $S$ :
4:    $\rho_s = \sum_{j \in S} \chi(d_{sj} - d_c)$ 
5: end for
6: for  $s$  in  $S$ :
7:    $\delta_s = \min_{i: i \in S, \rho_i > \rho_s} (d_{is})$ 
8: end for
9:  $\delta \leftarrow \frac{1}{|S|} \sum_{s \in S} \delta_s$ 
10: for  $s$  in  $S$ :
11:   if  $\delta_s > \delta$  then
12:      $k = k + 1$ 
13:      $s \leftarrow \text{newcluster}$ 
14:   else
15:      $s \leftarrow \text{cluster of nearest higher density}$ 
16: end for
```

---

In the [2] they used hop count as the metric for latency. That is all distances were calculated considering node to node distance a constant value of 1 (1 hop). We also considered the case if it is not constant. Therefore we implemented the same algorithm for the case where node to node distance can be any positive real number and used that as the distance metric. This numeric value can represent the bandwidth of that link

or channel. It can also represent the transmission rate of the link. We named this new implementation Weighted Density Based Controller Placement or W-DBCP and included it in the results analysis and algorithm comparison of section 8.3.

This algorithm has a few drawbacks which are also pointed out with example graphs in the section 8.3.

## 2.2 Controller Selection

DBCP uses three metrics to determine the controller for a cluster. The cluster head of a network is a node from where the cluster formation starts. A controller is the node which will act as the control plane for that specific sub-network data plane (rest of the switches). We selected the controllers for the sub-networks using the same metrics as DBCP. These three metrics are described in the following sections.

### 2.2.1 Controller-to-switch latency

For a sub-network graph  $S(\theta)$ , the average propagation latency for a placement location  $v : v \in S(\theta)$  of a controller is calculated as:

$$\pi_{avglatency}(S(\theta)) = \min_{v \in S(\theta)} \frac{1}{|S(\theta)|} \sum_{s \in S(\theta)} d(v, s) \quad (2)$$

Thus the average of the distances of the node  $v$  from all other nodes is calculated. To select a node as the controller,  $\pi_{avglatency}$  must be minimum. For the worst case scenario another metric is defined. This metric is denoted by  $\pi_{maxlatency}$ .

$$\pi_{maxlatency}(S(\theta)) = \min_{v \in S(\theta)} \max_{s \in S(\theta)} d(v, s) \quad (3)$$

This is the maximum of the distances of the node  $v$  from all the other nodes  $s : s \in S(\theta)$  in the cluster  $S(\theta)$ . This maximum must be minimized for a cluster. Therefore all nodes are considered when calculating  $\pi_{maxlatency}$  and only the switch with the minimum value is selected as controller.

### 2.2.2 Inter-controller latency

The inter controller latency must be reduced as much as possible when selecting controller. But the controller to controller distance cannot be determined when the controllers are yet to be selected. Thus a new metric is used which is denoted by  $\pi_{inter\_controller}$ .

$$\pi_{inter\_controller}(S(\theta)) = \min_{v \in S(\theta)} \frac{1}{|S|} \sum_{s \in (S-S(\theta))} d(v, s) \quad (4)$$

Thus it is the sum of the distances of the node  $v$  of the cluster  $S(\theta)$  to all the nodes of other clusters divided by the total number of nodes in the network. The node which has the minimum value of  $\pi_{inter\_controller}$  is selected as the controller  $\theta$ .

### 2.2.3 Total Latency

This is the final metric which is the sum of all three previous metrics in equations (2), (3) and (4).

$$\pi_{latency} = \pi_{avglatency} + \pi_{maxlatency} + \pi_{inter\_controller} \quad (5)$$

$$\begin{aligned} \pi_{latency}S(\theta) = & \min_{v \in S(\theta)} \left( \frac{1}{|S(\theta)|} \sum_{s \in S(\theta)} d(v, s) + \max_{s \in S(\theta)} d(v, s) \right) \\ & + \frac{1}{|S|} \sum_{s \in (S-S(\theta))} d(v, s) \end{aligned} \quad (6)$$

Thus for each node in a cluster  $S(\theta)$  we find the value of  $\pi_{latency}(S(\theta))$  and then we select the node with minimum  $\pi_{latency}$  as the controller  $\theta$  of the cluster.

## 2.3 Drawbacks

There are quite a few drawbacks of this network clustering algorithm. These are described in brief as follows:

1. For a densely connected graph almost every other node can be reached from a particular node in 1 or 2 hops. As such the only possible values of constant  $d_c$  are 1 or 2. For the value of constant  $d_c = 2$  all the nodes have the same density and therefore average minimum distance to higher density nodes is equal to minimum distance to higher density node for all the nodes and so according to the algorithm  $k = 0$  which basically means there are no clusters, which is not possible.
2. For  $d_c$  equal to 1 we are just basically calculating density based on incident degree of a node. Thus only 2 minimum distances to higher density nodes are possible which are 1 and 2. There are only two possible results in this case, one is  $k = 0$  as mentioned in previous point and the other is a certain number of controllers. Therefore there is very less variation in the number of clusters.
3. The Algorithm selects the cluster for a node following the condition of line 15 of algorithm 2. That is an un-clustered node is assigned a cluster based on the cluster of its nearest higher density. This creates a problem if a loop occurs such that there are two switches  $s_1$  and  $s_2$ , and  $s_1$  has nearest higher density  $s_2$  and  $s_2$  has nearest higher density  $s_1$ . This is possible when one switch is of highest density and other is farthest from the switch with highest density which is quite often in the case of dense graphs.

We showed the drawbacks using different scenarios again in section 8.3.

## 3 Algorithm 2: SPICi

SPICi ('spicy', Speed and Performance In Clustering) clusters a connected undirected graph  $G = (V, E)$  with edges that have confidence values of the continuous range  $(0, 1)$  so that each cluster has a seed as center from which clustering starts. These edges can be represented by  $w_{u,v}$  such that  $u, v \in V$ ,  $w_{u,v} \in E$  and  $0 \leq w_{u,v} \leq 1$ . It works with three variables and two thresholds. The three variables can be defined as the weighted degree of a node  $d_w$ , the density of a set of nodes  $S$  and the support of a node  $u \in V$  with respect to a set of nodes  $S$ :

$$d_w(u)_{u \in V} = \sum_{v: v \in V, (u,v) \in E} w_{u,v} \quad (7)$$

Therefore  $d_w(u)_{u \in V}$  is the sum of all the weights of the edges that connect  $u$  with any other adjacent node  $v$  of the graph  $G = (V, E)$ .

$$density(S) = \frac{\sum_{(u,v) \in E} w_{u,v}}{|S| * (|S| - 1)/2} \quad (8)$$

In other words,  $density(S)$  is the sum of the edges that connect every node  $u$  with every other node  $v$  of  $S$  divided by the number of total possible nodes that is  $|S| * (|S| - 1)/2$  where  $|S|$  is the number of nodes present in the set of nodes  $S$ .

$$support(u, S) = \sum_{v \in S} w_{u,v} \quad (9)$$

And  $support(u, S)$  is the sum of the edges that connect a node  $u$  with the nodes adjacent to it present in the set of nodes  $S$ .

The thresholds are:  $T_s$  which determines whether a node is to be included in the cluster based on the cluster size and the connectivity of the node to the cluster and:  $T_d$  which includes a node to the cluster based on the density increased when the node is added. The nodes are added to bins 1,2,3,4 and 5

---

#### Algorithm 2 : SPICi

---

```

1: procedure SEARCH(V,E)
2: Initialize DegreeQ = V
3: While DegreeQ  $\neq$  empty
4:   Extract u from DegreeQ with largest  $d_w(u)$ 
5:   if there is  $v: v, u \in E \in DegreeQ$  then
6:      $v \leftarrow secondseed(DegreeQ, E, u)$ 
7:     if  $v \neq null$  then  $S \leftarrow Expand(u, v)$ 
8:   else
9:      $S \leftarrow \{u\}$ 
10:   $V \leftarrow V - S$ 
11:   $DegreeQ \leftarrow DegreeQ - S$ 
12:   $d_w(t)_{t: t \in DegreeQ, (t,s) \in S \in E} = d_w(t) - support(t, S)$ 
13:
14: procedure SECONDSEED(V,E,u)
15:  $bin[i]_{i: i=(1,5)} \leftarrow s_{s: s \in V, (s,u) \in E}$ 
16: for i = 1 to 5
17:   if  $bin[i] \neq empty$  then
18:     return v if  $d_w(v) = \max_{s: s \in bin[i]} d_w(s)$ 
19: return null
20:
21: procedure EXPAND(V,E,u,v)
22: Initialize the cluster  $S \leftarrow \{u, v\}$ 
23: Initialize  $CandidateQ = S_{s: s \in S, (s,u), (s,v) \in E}$ 
24: While  $CandidateQ \neq empty$ 
25:   Extract t from Candidate with highest  $support(t, S)$ 
26:   if  $support(t, S) \geq T_s * |S| * density(S)$  and  $density(S + t) > T_d$  then
27:      $S \leftarrow S + \{t\}$ 
28:      $CandidateQ \leftarrow CandidateQ + \{s_{s: (s,t) \in E}\}$ 
29:      $CandidateQ \leftarrow CandidateQ - \{s_{s: s \notin CandidateQ}\}$ 
30:   else
31:     break from loop
32: return S

```

---

respectively if their connected edge value  $w_{s,u} \leq 0.2, 0.3, 0.4$

and 0.5 such that  $s, u \in V$  and  $(s, u) \in E$ .

## 4 Proposed 1: I-SPICi

The algorithm I-SPICi or Inverse-SPICi is a derivative of SPICi and one of our proposed algorithms. It has two steps *Clustering* and *Controller Selection* which are given as follows.

### 4.1 Clustering

This algorithm is taken from SPICi without any post processing or modification except that the cost is taken in an inverse manner. If the graph is represented by  $G = (V, E)$  where  $V$  is the set of nodes and  $E$  is the set of edges containing edge costs  $w_{u,v}$  such that  $u, v \in V$  and  $(u, v) \in E$  the costs can be expressed by the following equations.

$$max\_cost = \max_{u,v \in E} (w_{u,v} + 1) \quad (10)$$

$$cost_{u,v} = max\_cost - w_{u,v} \quad (11)$$

Here  $max\_cost$  is the maximum edge distance or cost of all the edges present in the graph. To each edge cost, 1 is added before calculating the maximum cost so that the selected maximum cost ( $max\_cost$ ) is greater than the actual maximum cost. Therefore after subtracting each of the edge costs from this maximum cost will not result in zero even for the actual maximum cost. When all the costs have been re-assigned, this results in the costs being reversed for any pair of adjacent nodes  $u, v : (u, v) \in E$ .

Thus the highest edge cost becomes the lowest and the lowest edge cost becomes the highest. This cost acts as the new edge value when calculating the values of weighted degree, support and density of the required set of nodes.

### 4.2 Controller Selection

We selected the controller for a cluster by calculating the  $support(u, S)$  (as expressed in equation (9)) for a set of nodes  $S$  which in this case is the cluster in consideration and a node  $u : u \in S$ . Each node gives a value of support and we selected the node with the maximum support. This can be improved further if we again take the original edge costs instead of the reversed ones or if we use the same controller selection method as used in DBCP.

## 5 Proposed 2: G-SPICi

### 5.1 Clustering

This algorithm is also taken from SPICi without any post processing. The only modification is that all the nodes of a cluster except the **first seed protein** (as per stated in [1]) or **cluster head** (in terms of clustering terminology), are selected using the same method. The node with maximum support with respect to the current cluster that is being built is selected each time. Therefore there would be no second seed selection function. The pseudo-code would also be changed. The changed pseudo-code is given below in algorithm 3.

---

**Algorithm 3 :Greedy-SPICi**

---

```

1: procedure SEARCH(V,E)
2: Initialize DegreeQ = V
3: While DegreeQ  $\neq$  empty
4:   Extract u from DegreeQ with largest  $d_w(u)$ 
5:    $S \leftarrow \text{Expand}(u)$ 
6:    $V \leftarrow V - S$ 
7:   DegreeQ  $\leftarrow$  DegreeQ - S
8:    $d_w(t)_{t:t \in \text{DegreeQ}, (t,s) \in S \in E} = d_w(t) - \text{support}(t, S)$ 
9:
10: procedure EXPAND(V,E,u)
11: Initialize the cluster  $S \leftarrow \{u\}$ 
12: Initialize CandidateQ =  $S_{S:s \in S, (s,u) \in E}$ 
13: While CandidateQ  $\neq$  empty
14:   Extract t from Candidate with highest  $\text{support}(t, S)$ 
15:   if  $\text{support}(t, S) \geq T_s * |S| * \text{density}(S)$  and  $\text{density}(S + t) > T_d$  then
16:      $S \leftarrow S + \{t\}$ 
17:     CandidateQ  $\leftarrow$  CandidateQ +  $\{s_{s:(s,t) \in E}\}$ 
18:     CandidateQ  $\leftarrow$  CandidateQ -  $\{s_{s:s \notin \text{CandidateQ}}\}$ 
19:   else
20:     break from loop
21: return S

```

---

This algorithm performs better than SPICi because SPICi selects the second seed in such a way that it has the maximum distance from first seed protein. This does not give good result when there are link with greater latencies. SPICi tends to select both nodes connected by the larger edge for the same cluster. Greedy SPICi or G-SPICi selects the one with maximum support thus this problem is somewhat reduced.

## 5.2 Controller Selection

We selected the controller for a cluster by calculating the  $\text{support}(u, S)$  (as expressed in equation (9)) for a set of nodes  $S$  which in this case is the cluster in consideration and a node  $u : u \in S$ . Each node gives a value of support and we selected the node with the maximum support. This can be improved further if we again take the original edge costs instead of the reversed ones or if we use the same controller selection method as used in DBCP.

## 6 Proposed 3: M-SPICi (20/4/2018)

### 6.1 Clustering

This algorithm is taken from SPICi with a few modifications. These modifications are given in the algorithm pseudo-code 4.

## 7 Local Search and Thresholds(Up.)

We performed local search on all of the SPICi variations and used those results as our final results. We also applied Local Search on both DBCP and W-DBCP. Then we named these implementations L-DBCP and LW-DBCP and used their results for comparison. All the Local Search algorithm use the

---

**Algorithm 4 : SPICi**

---

```

1: procedure SEARCH(V,E)
2: for all  $w_{u,v}$  in E
3:    $w_{u,v} = 1/w_{u,v}$ 
4: end loop
5: Initialize DegreeQ = V
6: While DegreeQ  $\neq$  empty
7:   Extract u from DegreeQ with largest  $d_w(u)$ 
8:   if there is  $v_{v:v,u \in E} \in \text{DegreeQ}$  then
9:      $v \leftarrow \text{secondseed}(\text{DegreeQ}, E, u)$ 
10:    if  $v \neq \text{null}$  then  $S \leftarrow \text{Expand}(u, v)$ 
11:  else
12:     $S \leftarrow \{u\}$ 
13:   $V \leftarrow V - S$ 
14:  DegreeQ  $\leftarrow$  DegreeQ - S
15:   $d_w(t)_{t:t \in \text{DegreeQ}, (t,s) \in S \in E} = d_w(t) - \text{support}(t, S)$ 
16:  for all s in V
17:    if s is isolated then
18:      Include s in adjacent cluster with nearest first seed
19:    end loop
20:
21: procedure SECONDSEED(V,E,u)
22:  $\text{bin}[i]_{i:(1,5)} \leftarrow s_{s:s \in V, (s,u) \in E}$ 
23: for i = 1 to 5
24:   if  $\text{bin}[i] \neq \text{empty}$  then
25:     return v if  $d_w(v) = \max_{s:s \in \text{bin}[i]} d_w(s)$ 
26: return null
27:
28: procedure EXPAND(V,E,u,v)
29: Initialize the cluster  $S \leftarrow \{u, v\}$ 
30: Initialize CandidateQ =  $S_{S:s \in S, (s,u), (s,v) \in E}$ 
31: While CandidateQ  $\neq$  empty
32:   Extract t from Candidate with highest  $\text{support}(t, S)$ 
33:   if  $\text{support}(t, S) \geq T_s * |S| * \text{density}(S)$  and  $\text{density}(S + t) > T_d$  then
34:      $S \leftarrow S + \{t\}$ 
35:     CandidateQ  $\leftarrow$  CandidateQ +  $\{s_{s:(s,t) \in E}\}$ 
36:     CandidateQ  $\leftarrow$  CandidateQ -  $\{s_{s:s \notin \text{CandidateQ}}\}$ 
37:   else
38:     d = degree of node t
39:     if  $d > \text{degreeof}(80\% \text{nodes})$  then
40:        $S \leftarrow S + \{t\}$ 
41: return S

```

---

same step by step process. Only the metric is considered different for two different cases as given in section 8.3. This step by step process algorithm is given in Algorithm 5.

---

**Algorithm 5 : Local Search**


---

```

1: procedure LOCAL-SEARCH(V,E)
2: calculate:
3: set-controller-head()
4: cluster head to head distance()
5: cluster-diameter()
6: return latency(m-latency/our-latency)
7: loop for n times:
8:   local-search(previous-latency):
9:   randomly select c
10: loop for c times:
11:   randomly select node a
12:   randomly select cluster b
13:   if combination a,b is already taken then continue;
14:   change cluster of a to b;
15:   new-latency=calculate()
16:   if new-latency less than previous-latency then
17:     call local-search(new-latency)
18:   break
19: else
20:   set cluster of a to previous cluster of a
21: end loop
22: end loop

```

---

In the given algorithm  $n$  is the number of iterations. For all versions of SPICi we used 100 iterations and for DBCP and W-DBCP we used 5 iterations as the solution was already close to optimal, more iterations takes a huge amount of time to be computed.

In SPICi the value of  $T_s$  have an effect on the cluster size. The Greater the value, less number of nodes there are in a cluster. The value was originally set to 0.5 in SPICi. But as we experimented more, we think the value depends more on the type of input graph rather than being a fixed value. For SPICi we tried to take the value near 0.5.

For Inverse-SPICi when the value is near 1 it gives better results. Slight variation in the value may change the cluster size dramatically.

**Problem:** Determining the value of  $T_s$

**Solution:** We can do a Ternary search.

**Procedure:**

1. First We will select a value of K (The number of cluster). We won't have more than K clusters.
2. Perform a Search to determine the value of  $T_s$  depending on the result that SPICi gives. We can find a suitable metric for this.

For example the values of thresholds of SPICi versions were calculated as given in algorithm 6.

We calculated the value of the thresholds for both I-SPICi and G-SPICi in this way. And for DBCP and W-DBCP we calculated the threshold  $d_c$  by looping through the values from  $0.3 \times diameter$  to  $0.5 \times diameter$  of the graph as per given in [2]. We incremented the values each time by 0.5 and tested

---

**Algorithm 6 : Ternary Search**


---

```

1: procedure TERNARY-SEARCH(V,E)
2: low=0.000001 high=1.500000 loop 60 times:
3:   mid1=1/3 of the range
4:   mid2=2/3 of the range
5:   set Ts=mid1
6:   spici()
7:   temp1=calculate()
8:   set Ts=mid2
9:   spici()
10:  temp2=calculate()
11:  if temp1 < temp2 then set high=mid2
12:  else
13:    set low=mid1
  end loop local-search()

```

---

the latency for each iteration to find the best result (minimum latency).

## 8 Performance Evaluation (20/4/2018)

Initially we defined the metrics that we used to evaluate the performance of the algorithms. Then we compared the algorithms using those metrics.

### 8.1 Performance Metrics 1

We used the performance metric  $m_{latency}$  which has been used in [2]. The metric can be expressed using the following equation.

$$m_{latency} = \frac{1}{|S|(|S| - 1)} \sum_{s_i, s_j \in S, i \neq j} \{d(s_i, v_i) + \max_{s_m \in Path_{i,j}} (d(v_i, v_m) + d(v_m, s_m))\} \quad (12)$$

Here  $|S|$  is the total number of nodes in the whole network  $S$ . Switches  $s_i$  and  $s_j$  are two switches which have a path  $Path_{i,j}$  between them.  $s_m$  represents the switches that are in the path connecting  $s_i$  and  $s_j$ .  $v_i$  is the controller of the cluster where  $s_i$  is, and  $v_m$  is the controller of the cluster of  $s_m$ .

Therefore for every node or switch  $s_m$  in the path  $Path_{i,j}$  connecting any pair of switches, the maximum distance between the node or switch  $s_m$ , and the controller  $v_m$ , to which it is assigned,  $d(v_m, s_m)$  is added to the distance from this controller  $v_m$  to the controller  $v_i$  of the starting switch  $s_i$ . This maximum distance is added to the distance from the starting switch  $s_i$  to the starting controller  $v_i$ . This is the latency from one switch to another. There are  $|S| * (|S| - 1) / 2$  possible pairs of switches in a set of switches  $S$  giving that many latency values. All these values are added and then divided by  $|S| * (|S| - 1)$ . This is the  $m_{latency}$  for a network.

### 8.2 Performance Metrics 2

We have used another metric to evaluate the results of our algorithm compared to other algorithms. We define this metric  $\pi_{latency}$  of a clustered network as the sum of two variables

Table 1: Scenario Networks Specifications

	Nodes	Edges	Cycles/Loop
Scenario 1	40	52	Yes
Scenario 2	50	68	Yes
Scenario 3	60	77	Yes
Scenario 4	70	2907	Yes
Scenario 5	80	108	Yes
Scenario 6	90	120	Yes
Scenario 7	100	131	Yes

which are called  $\pi_{avglatency}$  and  $\pi_{intercontroller}$ . They can be expressed as the following two equations.

$$\pi_{avglatency} = \frac{\sum_{S \in V} \sum_{c, i \in S} d_{ci}}{k} \quad (13)$$

Here  $S$  is the set of nodes or cluster of nodes that belong to the nodes ( $V$ ) of the graph ( $G = (V, E)$ ). For each cluster  $S$  the controller is the node  $c$  and  $i$  any other node. Therefore  $d_{ci}$  is any distance from a controller to any of the nodes of the cluster.  $k$  is the number of clusters created or in other words, number of controllers.

$$\pi_{intercontroller} = \frac{\sum_{i, j \in V} d_{ij}}{k} \quad (14)$$

Here  $i$  and  $j$  are controllers of the clustered network and  $d_{ij}$  is the distance between controller  $i$  and  $j$ . Therefore  $\pi_{intercontroller}$  is the sum of all possible controller to controller distance divided by the number of controllers or number of clusters. It is the average of the sum of distances of each controller to all other controllers.

Therefore the final metric is  $\pi_{latency}$  which can be expressed as the following equation.

$$\pi_{latency} = \pi_{avglatency} + \pi_{intercontroller} \quad (15)$$

### 8.3 Comparison of Algorithms(Update)

We have created 7 scenarios for the evaluation of the algorithm and to compare it with other well-known existing algorithms. A brief description of the networks in these scenarios are given in the following table.

We have applied all the algorithms on these 7 scenarios in two ways. One is keeping the networks as it is. The second is changing the random weights that the networks have to 1. Therefore the distance from a node to node is represented by 1 hop instead of the previously assigned weights that it had.

In figure 1 we compared the well-known DBCP algorithm with our proposed algorithm. As the DBCP works with hop count, we used unweighted versions of the 7 networks we showed in table 1. And as both algorithms that is DBCP and our proposed algorithm give different values of  $k$  (number of controllers) we forced DBCP to use the same  $k$  as that of our algorithm and named it Forced-DBCP. In this figure our proposed algorithm performs better than DBCP. Forced-DBCP does better than DBCP and almost close to our algorithm but our algorithm is still better.

DBCP and our proposed algorithm gives different values of  $k$  but this is also an important criteria when clustering a

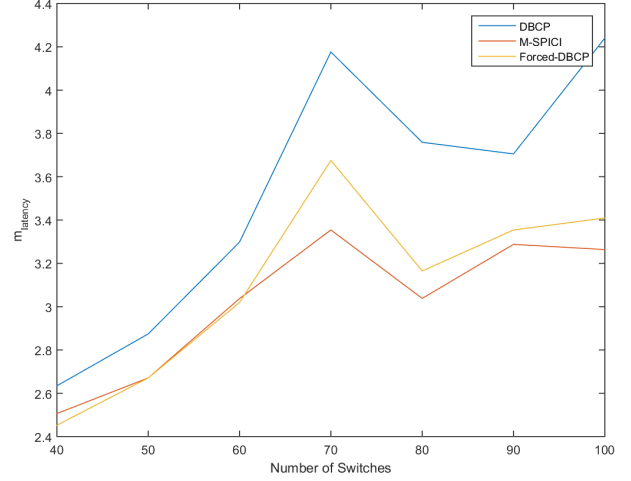


Figure 1: Comparison among DBCP, proposed algorithm and also Forced-DBCP which is forced to use the same value of  $k$  as proposed algorithm using  $m_{latency}$  (DBCP Metric) for Unweighted Networks

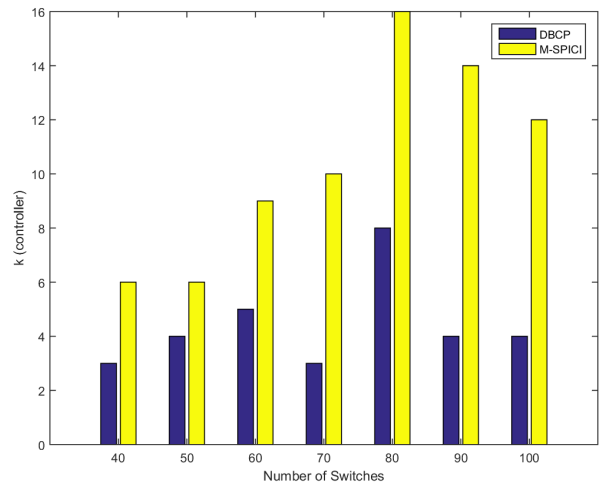


Figure 2: Values of  $k$  for different algorithms in case of Unweighted Networks

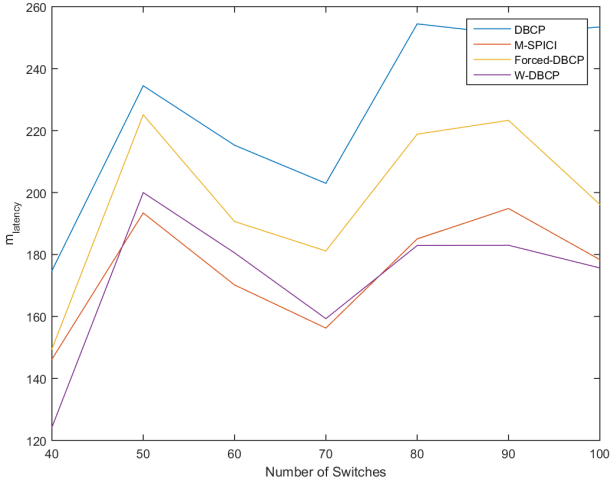


Figure 3: Comparison among different SPiCi versions using  $m_{latency}$  (DBCP Metric) for Weighted Networks

network so we showed a comparison of the  $k$  values given by both for the given scenarios in figure 2. This is also for un-weighted network graphs. In case of Forced-DBCP the value of  $k$  is the same as our proposed algorithm so we did not show it separately.

We used different kinds of SPiCi variations to compare amongst themselves. Then we chose the best algorithm out of these to compare with DBCP and W-DBCP. As we can see in figure ?? our algorithm surpasses all algorithms and only weighted-DBCP comes near to our results.

In figure 4 we compared the well-known DBCP algorithm with our proposed algorithm. In this case we used weighted versions of the 7 networks we showed in table 1. And as both algorithms that is DBCP and our proposed algorithm give different values of  $k$  (number of controllers) we forced DBCP to use the same  $k$  as that of our algorithm and named it Forced-DBCP. In this figure our proposed algorithm performs a lot better than DBCP. This is because DBCP does not consider weights of the edges. So we used a new DBCP which works on weighted graph and named it W-DBCP or weighted DBCP. Only this algorithm, that is this version of DBCP competes with our proposed algorithm. Forced-DBCP is close to our algorithm in terms of latency as both have the same value of  $k$  for clustering.

DBCP and our proposed algorithm gives different values of  $k$  but this is also an important criteria when clustering a network so we showed a comparison of the  $k$  values given by both for the given scenarios in figure 2. This is also for un-weighted network graphs. In case of Forced-DBCP the value of  $k$  is the same as our proposed algorithm so we did not show it separately.

## References

- [1] Peng Jiang and Mona Singh. Spici: a fast clustering algorithm for large biological networks. *Bioinformatics*, 26(8):1105–1111, 2010.
- [2] Jingyu Wang Qi Qi Kai Li Jianxin Liao, Haifeng Sun and

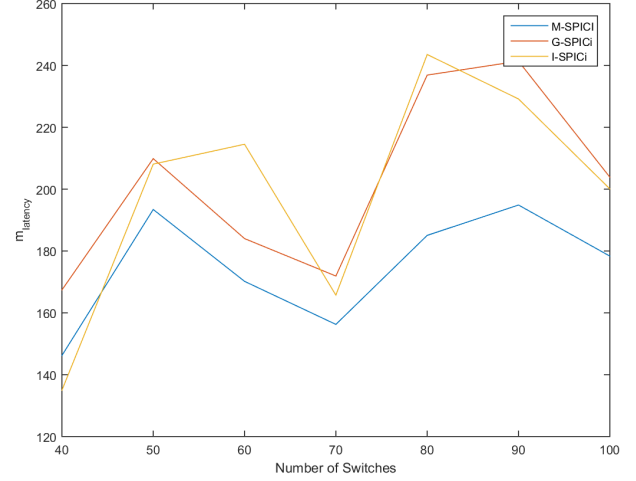


Figure 4: Comparison among DBCP, Weighted DBCP, proposed algorithm and also Forced-DBCP which is forced to use the same value of  $k$  as proposed algorithms using  $m_{latency}$  (DBCP Metric) for Weighted Networks

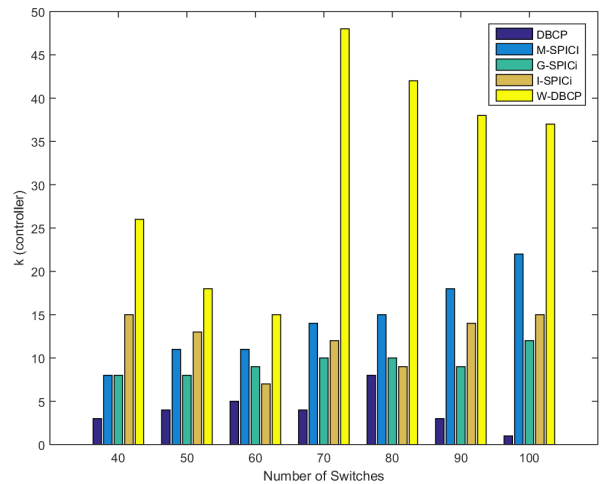


Figure 5: Values of  $k$  for different algorithms in case of Weighted Networks

Tonghong Li. Density cluster based approach for controller placement problem in large-scale software defined networkings. *Computer Networks*, (112):24–35, 2017.