Chapter 6

K++ Shell: Influence maximization in multilayer networks using community detection

SIM and CBIM take more processing time for detecting communities and seed node selection. In this chapter[1], we propose a K++-Shell decomposition algorithm. It reduces the time complexity and improves the performance in influence spreading. The K++-Shell decomposition algorithm aims to find the most influential nodes in multilayer networks. In this chapter, we tried present this concept in a different way. i.e., Viral marketing. Studying IM in multilayer networks in the context of viral marketing will be interesting. Motivated by this, this chapter investigates the K++-Shell decomposition algorithm to find the k set of influential nodes (seed nodes) in a multilayer network. We prune the nodes based on degree and assign reward points to their neighbors. We conducted a comparative study of various IM algorithms and reported the results. K++-Shell performs better in finding influential nodes on various real-time datasets under various settings and environments.

The significant contribution of this chapter has given below:

- We adapt the K-shell decomposition algorithm for influence maximization in a multilayer network.
- We adopt the Label Propagation Algorithm to find the communities for multilayer net-

[1]Under minor revision at computer networks journal, Elsevier

works.

- We proposed the K++-Shell algorithm and applied it to communities to select the seed nodes in multilayer networks.
- We compare the performance of the proposed algorithm with the State-of-the-art algorithms under the IC model.

Table 6.1: Notations used in this chapter

Notation	Description
V	Set of nodes in the multilayer network
K	Set of seed nodes
p	Propagation probability in IC model
G^i	i^th layer in multilayer network
p	Propagation probability in IC model
L	Total number of layers in multilayer networks
d_j^l	Degree of node j in l^{th} layer
$nbrs(u^l)$	Neighbors of node u in l^{th} layer

6.1 Model of multilayer networks

A multilayer network can be represented as $M = \{G^1, G^2, ..G^{|L|}\}$ [4] is a multilayer network i.e $M = \{G^i; i = 1,2,..|L|\}$. In multilayer networks, a node will have a clone in every layer, i.e., $|V^1| = |V^2| = ...|V^{|L|}| = |V|/|L|$. The same set of nodes is present in all the layers, i.e., a user will have a different social network account in each layer. In multilayer networks, edges are between a node and its counterpart in other layers. The single layer is represented as $G^i(V^i, E^i)$, where V^i set of nodes and E^i is the set of edges in layer i. The notations used in this chapter are given in Table 6.1.

6.2 K++ Shell: Influence maximization in multilayer networks using community detection

This section is divided into two phases. In the first phase, we discuss the design and development of the K++-Shell decomposition algorithm. This algorithm is used to find the influential

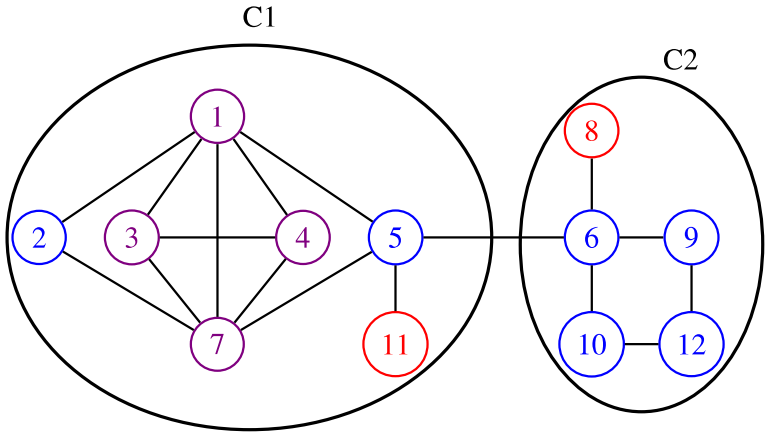


Figure 6.1: Network is divided into two communities C1 and C2.

nodes. In this phase, the K++-Shell algorithm prunes the nodes based on their degrees. K++-Shell assigns a node to its bucket. Reward points of all the neighbors of a pruned node will be incremented by one. In the second phase, K++-Shell selects the seed nodes for spreading the information. Before applying phase 1, we find the communities using the label propagation algorithm (LPA) [78], which is an efficient community-based algorithm.

6.2.1 Phase 1: K++-Shell decomposition algorithm

The K++-Shell decomposition algorithm is inspired by the K-shell decomposition algorithm[9]. The K-shell decomposition algorithm prunes the nodes based on the nodes' degree. All the nodes with degree one are deleted from the network. After deleting the nodes with degree one, the degrees of the remaining nodes are updated. Due to this, the network may still have nodes with degree one. The k-shell algorithm deletes all such nodes with degree one, which continues until saturation. Similarly, this process applies for nodes with degrees 2, 3,...so on till all the nodes are deleted.

In the K-shell algorithm, the selection of seed nodes process is not efficient. If we apply the K-shell algorithm to community C1 of Figure 6.1, node 11 assigns to bucket B1. The degree of node five updates from 3 to 2. In the next iteration, as we do not have nodes with degree one, nodes 2 and 5 will be deleted and assigned to bucket B2. Degrees of nodes 1 and 7 updated from 5 to 3. The k-shell algorithm will check for nodes with degrees one and two, but as the updated graph does not contain nodes with degrees one and two, it will check for nodes with degrees three. Nodes with degrees three-1,3,4 and 7 will be deleted from the graph and assigned to bucket B3.

All the nodes in the highest bucket may not be a good choice for seed node selection. A few nodes in lower buckets may be good candidates for seed nodes. We address this drawback

in the K++-Shell algorithm by introducing reward points to the nodes. In K++-Shell decomposition, neighbors of every removed node will be rewarded by point one. While selecting influential nodes, K++-Shell considers both the node's bucket number and reward points. K++-Shell decomposition is given in Algorithm 8.

Algorithm 8 K++-Shell decomposition algorithm

1: Initialize the Node set (NS) and buckets list (BL), temp.

$$NS \leftarrow V, BL \leftarrow \Phi, \text{temp=1, n=highest degree in the network.}$$

2: Find the communities using Label Propagation Algorithm.
3: Find the degrees of the nodes.
4: **for each** $temp \leq n$ **do**
5: **for each** $i^l \leq NS$ **do**
6: **If** $degree[i^l] == temp$
7: Remove node i from the node set

$$NS \leftarrow NS - i^l,$$

8: Add one reward point to the neighbours of node i^l.
9: Add the node i^l to the degree's bucket i.e., Bucket[temp].

$$Bucket[temp] \leftarrow Bucket[temp] + i^l$$

10: **end for**
11: Still if any node left with degree $temp$, repeat the process from 5 to 8.
12: **end for**
13:

$$temp \leftarrow temp + 1$$

14: Update the buckets list BL.

Let us apply the K++ Shell decomposition of C1 and C2 in Figure 6.1. The network divides into two communities, i.e., C1 and C2. First, K++ Shell decomposition will be done on community C1. Remove Nodes with degree one, i.e., node 11 from the community C1, assign it to bucket B1 of community C1 and reward one point to its neighbor node 5. Then, as there are no nodes with degree one, nodes with 2-degree, i.e., nodes 2 and 5 will be deleted from the community C1 and assigned to bucket B2. Neighbors of nodes 2 and 4 will be rewarded by one point, so nodes 1 and 7 will have reward points two each. Now the graph has four 3-degree nodes, i.e., nodes 1, 3, 4, and 7. All four nodes will be assigned to bucket B3 of community C1. Similarly, apply K++ Shell decomposition to community C2. Remove one-degree node 8 and put it in bucket B1 of community C2. Award one reward point to its neighbor node 6. After that, remove nodes 9, 10, and 12, and put them in bucket B2 of community C2. Award two reward points to

node 6 as nodes 9, 10 are neighbors to node 6. Finally, remove node 6 and put it in bucket B1. The reward point triplet of a community is defined as (x,y,z), where x denotes Bucket number, y denotes node number, and z denotes reward points. Rewards points of communities C1 and C2 as follows $C1 = \{(B1,11,0),(B2,5,1)(B2,2,0)(B3,1,2),(B3,7,2),(B3,3,0),(B3,4,0)\}, C2 = \{(B1,8,0),(B2,6,2),(B2,9,0),(B2,10,0),(B2,12,0)\}\}$.

Algorithm 9 Seed node selection process

1: Initialize the K, n.

$$K = number\ of\ seed\ nodes, n = highest\ bucket\ number$$

2: Sort all the nodes in all the buckets in descending order based on the reward points of the node.
3: **for each** $n \geq 1$ **do**
4: **If** $K \leq |bucket - n|$
5: Select the top K nodes as seed node set S from the $bucket - n$.
6: **break**
7: **else if** $K > |bucket - n|$
8: Select all the nodes from the $bucket - n$ as seed nodes.
9: Update the number of required seed nodes.

$$K = K - |bucket - n|$$

10: decrement the n.

$$n \leftarrow n - 1$$

11: **end for**
12: return S

6.2.2 Phase 2: Finding seed nodes

After finding the communities and decomposing the nodes, K++-Shell decomposition computes the seed node quota for each community. The seed node quota is computed as follows:

$$SQ(C_i^l) = |K| * \frac{n_i^l}{|V|} \tag{6.1}$$

Where $SQ(C_i^l)$ is the seed node quota of community C_i in layer l. K represents the total

number of seed nodes to be selected from all the communities. n_i^l is the number of nodes in community C_i^l. $|V|$ is the total number of nodes in the network.

After finding the seed node quota for each community, the K++-Shell algorithm selects seed nodes from each community. The procedure is to select seed nodes given in algorithm 9. K++-Shell algorithm sorts the nodes in descending order within each bucket-based reward point. Seed nodes are selected based on the following:

- **Case 1:** If the quota seed nodes are less than the number of nodes in the bucket with the highest number, then we select all the quota nodes are selected from it.
- **Case 2:** If the required quota seed nodes are more than the nodes in the bucket with the highest number, then the K++-Shell selects all the nodes from the bucket with the highest number. If the remaining quota nodes are less than the nodes in the bucket with the next highest number, case 1 is applied. Otherwise, case 2 is repeated till all the quota nodes are selected.

In Figure 6.1, If we have to select five seed nodes from community C1 using the K-shell algorithm, it will select four nodes (nodes 1,3,4 and 7) from bucket B3 and one of the two nodes (node 2 or node 5) from bucket 2. In K++-Shell decomposition, if we have to select five seed nodes from Community C1, it will select four nodes (nodes 1,3,4, and 7) from bucket B3 and node 5 from bucket B2 as it has higher reward points than node 2.

6.3 Evaluation

This section discusses the real-time datasets used for running our proposed K++-Shell decomposition algorithm. We ran our experiments with the following configuration: Intel Xeon(R) workstation with a 2.9 GHz CPU, 64GB RAM, and Ubuntu 16.04. K++-Shell is implemented using the python programming language.

6.3.1 Time complexity

Let us evaluate the time complexity of performing K++ Shell decomposition. For finding communities in a multilayer network with V number of nodes using the LPA algorithm is $O(V)$. The complexity of finding the degree of the nodes is $O(V^2)$. The pruning of nodes based on degree and rewarding points to neighbor nodes is $O(n*V+E)$. In the seed node selection process, the complexity of sorting the nodes in descending order is $O(VlogV)$. The complexity of selecting

K seed nodes will be $O(K)$ (assuming that the degrees of nodes are available and sorted within the community).

6.3.2 Baseline models

We compare two models that are KSN [42] introduced in Chapter 3, and CIM [36] in Chapter 4. We also compare with two more models as given below.

- **K-shell [9]** : K- Shell decomposition algorithm uses the percolation theory and fractal geometry to find the network's topology. [9] decompose the network into k-shells by removing the nodes based on the degree. The highest shell index is K_{max}. Seed node selection is made from various shells.
- **CKS [78]**: Community-based K-shell decomposition (CKS) uses the Girvan-Newman algorithm to find the communities, and then it applies K-shell decomposition on the communities to find the seed nodes.

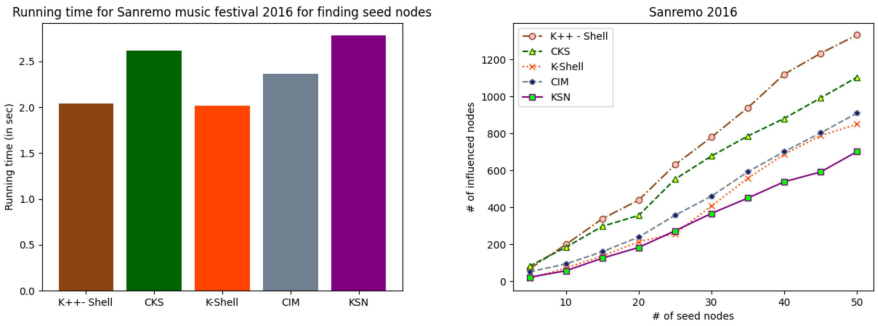
6.4 Results

In this section, the K++-Shell algorithm with other state-of-the-art models in Section 6.3.2 using the datasets listed in Section 3.2. After finding the seed nodes, we analyze and compare the influence maximization of different algorithms. For our experiments, we fix the size of the seed node set as 50. From Table 6.2, the running times of 25%, 50%, 75%, and 100% of the Sanremo 2016 final dataset are 0.435, 1.0023, 1.456, and 2.0438, respectively, for the K++ Shell algorithm. It can be observed that the running time is linear to the size of the problem. Therefore, we can conclude that the K++ Shell algorithm is scalable. From the table 6.2, the influence spread of 25%, 50%, 75%, and 100% data is 332, 688, 831, and 1332, respectively, for the K++ Shell algorithm. It can be observed that the influence spread is linear to the size of the problem. Therefore, accuracy is maintained even with the increase in the problem size.

We present the execution time for finding influential nodes and influence spread using different algorithms. We use the IC model to compute the influence spread of all the algorithms. In all the scenarios, K++-Shell is efficient in finding seed nodes and spreading the information. Figure 6.2a shows the execution time for finding seed nodes on the Sanremo music festival 2016 dataset. K++-Shell is more efficient than CKS, KSN, and CIM. K++-Shell takes marginally more time than K-shell but outperforms all the state-of-the-art models, including K-shell, in spreading the information as shown in Figure 6.2b.

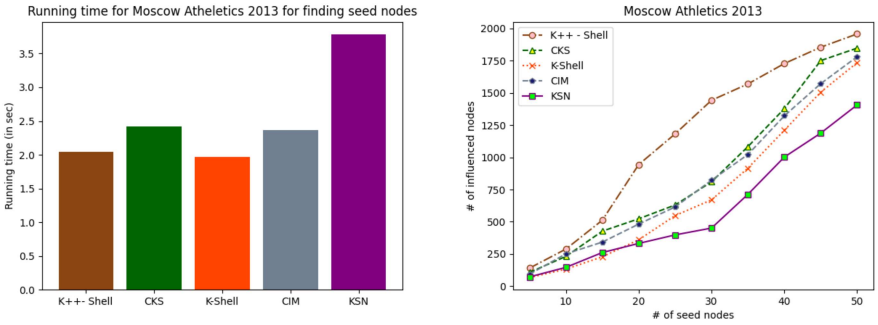
Table 6.2: Influence spread (IS) and Running time (RT in sec) on various datasets.

Influence Spread/ Running Time				Sanremo 2016 final	Moscow Athletics 2013	Newyork climate change 2014	MLKing 2013	Cannes 2013
	25%	K++ Shell	IS	332	484	758	1694	1934
			RT	0.435	0.7623	1.0234	3.0156	3.9021
		CKS	IS	270	443	698	1583	1827
			RT	0.654	0.9243	1.5623	3.730	4.2610
		K-Shell	IS	215	442	631	1475	1495
			RT	0.478	0.5463	1.0624	2.8913	3.3017
		CIM	IS	241	458	669	1573	1745
			RT	0.601	1.3876	1.4732	3.742	3.910
		KSN	IS	191	363	579	1436	1673
			RT	0.7234	1.1652	1.742	4.176	4.6104
	50%	K++ Shell	IS	688	963	1548	3371	3842
			RT	1.0023	1.4659	2.365	5.842	7.891
		CKS	IS	576	903	1383	3201	3635
			RT	1.402	1.7327	2.9135	7.024	8.042
		K-Shell	IS	409	861	1234	2943	3021
			RT	0.9237	0.9978	2.2563	5.734	6.5901
		CIM	IS	472	896	1321	3102	3463
			RT	1.1023	2.235	2.7631	7.184	7.6027
		KSN	IS	368	711	1142	2853	3383
			RT	1.3086	2.5645	3.1367	8.0267	9.027
	75%	K++ Shell	IS	1010	1453	2309	5067	5789
			RT	1.456	2.1675	3.134	8.369	9.0176
		CKS	IS	831	1334	2081	4789	5427
			RT	1.3423	2.3678	4.134	8.9245	12.760
		K-Shell	IS	642	1292	1876	4451	4457
			RT	1.9867	1.2134	3.210	7.210	9.471
		CIM	IS	721	1362	2002	4678	5176
			RT	1.678	3.2476	4.014	9.4276	10.720
		KSN	IS	572	1093	1726	4289	5083
			RT	2.6590	3.7684	5.163	10.985	13.041
	100%	K++ Shell	IS	1332	1958	3083	6700	7701
			RT	2.0438	2.745	4.4181	11.2181	13.2181
		CKS	IS	1104	1848	2785	6377	7287
			RT	2.6196	3.124	5.6890	13.889	15.7546
		K-Shell	IS	850	1735	2512	5919	6082
			RT	2.0168	1.9893	4.3276	10.9827	12.9856
		CIM	IS	910	1781	2612	6232	6888
			RT	2.3616	4.2725	5.1423	13.5423	14.5423



(a) (b)

Figure 6.2: (a) Execution time for various algorithms to find 50 seed users on Sanremo music festival 2016 dataset. (b) Number of influenced people for 50 seed users for various algorithms using IC model on Sanremo music festival 2016 dataset.

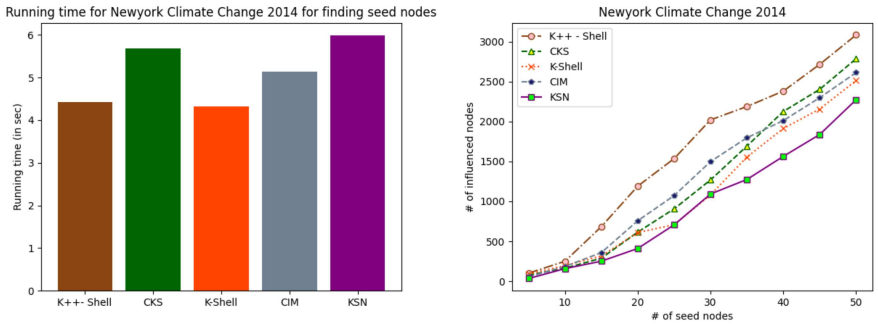


(a) (b)

Figure 6.3: (a) Execution time for various algorithms to find 50 seed users on Moscow athletics 2013 dataset. (b) Number of influenced people for 50 seed users for various algorithms using IC model on Moscow Athletics 2013 dataset.

Figure 6.3a shows the execution time for finding seed nodes on the Moscow athletics 2013 dataset. K-shell takes marginally less time than K++-Shell and less time than other algorithms. KSN takes more time than all other algorithms. K++-Shell and K-Shell take almost the same time to find influential nodes. Figure 6.3b presents the number of influenced nodes on the Moscow Athletics 2013 data set. K++-Shell has performed better than all other algorithms. CKS has performed better than KSN and K-shell. Figure 6.4a shows the execution time for finding seed nodes on the Newyork climate change 2014 dataset. CKS takes more time than all other algorithms to find seed nodes. K Shell is more efficient than all other algorithms. Figure 6.4b presents the number of people influenced by Newyork climate change in the 2014 data set. K++-Shell has influenced more people than all other algorithms. CIM has influenced marginally more than K-shell and KSN.

Figure 6.5a shows the execution time for finding seed nodes on Martin Luther King's



(a) (b)

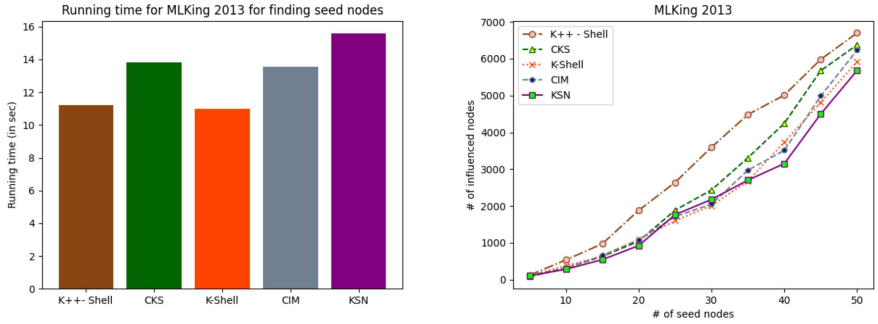
Figure 6.4: (a) Execution time for various algorithms to find 50 seed nodes on Newyork climate change 2014 dataset. (b) Number of influenced people for 50 seed users for various algorithms using IC model on Newyork climate change 2014 dataset.

50th-anniversary speech dataset. CKS takes more time than all other algorithms, and CIM is more efficient than KSN. K-shell is more efficient than all other algorithms. K++-Shell is more efficient than CIM and KSN. Figure 6.5b presents the influence maximization on Martin Luther King's 50th-anniversary speech in 2013. K++-Shell has influenced more nodes than all the other algorithms. CIM has influenced marginally more than KSN and K-shell. K-shell influenced a marginally less number of people than CKS and CIM.

Figure 6.6a shows the execution time for finding seed nodes at the Cannes film festival 2013. CKS takes more time than all other algorithms, and K-shell takes less time than all other algorithms. KSN takes less time than CKS, and KSN takes more time than K++-Shell, K-Shell, and CIM. Figure 6.6b presents the influence maximization on the Cannes film festival 2013 dataset. K++-Shell has influenced more nodes than all other algorithms. K-shell has influenced less number of nodes than all other algorithms. CIM has influenced more people than K-shell and KSN. CIM has influenced less number of nodes than K++-Shell and CKS. From our experiments, we observed that K++-Shell is an effective algorithm for spreading information in multilayer networks. The advantage is due to the exploitation of clone structure in multilayer networks by K++-Shell algorithms.

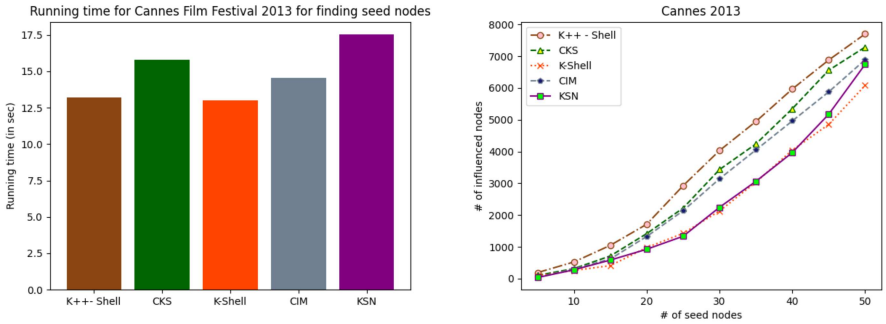
6.4.1 Comparision with CIM, SIM and CBIM

This section compares and analyzes the results of K++-Shell, CBIM, SIM, and CIM. Results are in two scenarios, i.e., the influence spread and execution time for finding seed nodes on real-time datasets, which are listed in Section 3.2. Figure 6.7a shows the influence spread on the Sanremo music festival 2016 dataset. In influence spread, K++-Shell has performed better than CBIM, SIM, and CIM. Figure 6.7b presents the running time to find seed nodes on the



(a) (b)

Figure 6.5: (a) Execution time for various algorithms to find 50 seed users on Martin Luther King's 50th anniversary 2013 dataset. (b) Number of influenced people for 50 seed users for various algorithms using IC model on Martin Luther King's 50th anniversary 2013 dataset.



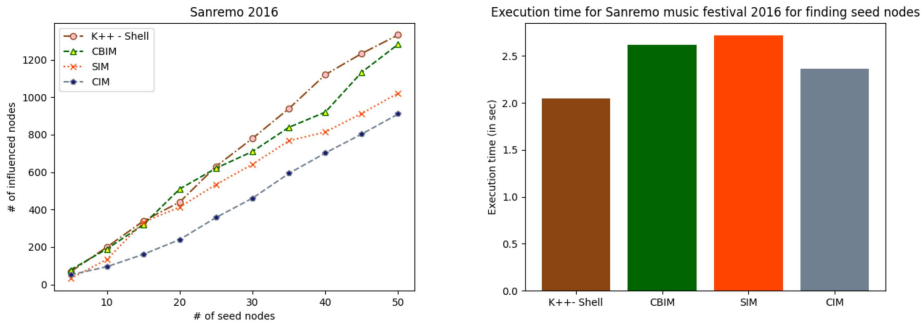
(a) (b)

Figure 6.6: (a) Execution time for various algorithms to find 50 seed users on Cannes film festival 2013 dataset. (b) Number of influenced people for 50 seed users for various algorithms using IC model on Cannes film festival 2013 dataset.

Sanremo music festival 2016 dataset. SIM takes more time than K++-Shell, CBIM, and CIM.

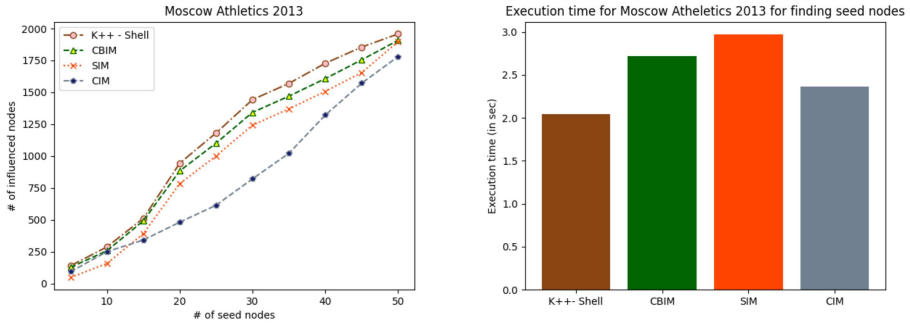
Figure 6.8a shows the influence spread on Moscow Athletics 2013 dataset. In influence spread, K++-Shell has performed better than CBIM, SIM, and CIM. Figure 6.8b presents the running time to find seed nodes on the Moscow athletics 2013 dataset. SIM takes more time than K++-Shell, CBIM, and CIM. Figure 6.9a shows the influence spread on the Newyork climate change 2014 dataset. In influence spread, K++-Shell has performed better than CBIM, SIM, and CIM. Figure 6.9b presents the running time to find seed nodes on the Newyork climate change 2014 dataset. SIM takes more time than K++-Shell, CBIM, and CIM.

Figure 6.10a shows the influence spread on Martin Luther King's 50th anniversary 2013 dataset. In influence spread, K++-Shell has performed better than CBIM, SIM, and CIM. Figure 6.10b presents the running time to find seed nodes on Martin Luther King's 50th anniversary 2013 dataset. SIM takes more time than K++-Shell, CBIM, and CIM. Figure 6.11a shows the



(a) (b)

Figure 6.7: (a) Number of influenced people for 50 seed users for various algorithms using IC model on Sanremo music festival 2016 dataset. (b) Execution time for various algorithms to find 50 seed users on Sanremo music festival 2016 dataset.



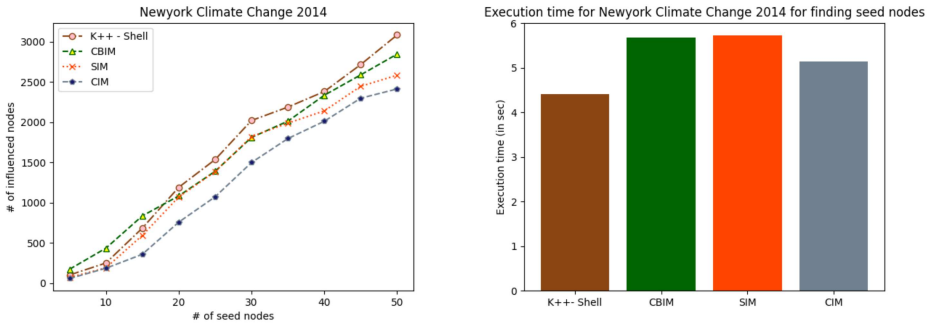
(a) (b)

Figure 6.8: (a) Number of influenced people for 50 seed users for various algorithms using IC model on Moscow Athletics 2013 dataset. (b) Execution time for various algorithms to find 50 seed users on Moscow athletics 2013 dataset.

influence spread on the Cannes film festival 2013 dataset. CBIM has performed better than SIM and CIM in influence spread. Figure 6.11b shows the running time to find seed nodes on the Cannes film festival 2013 dataset. SIM takes more time than CBIM and CIM.

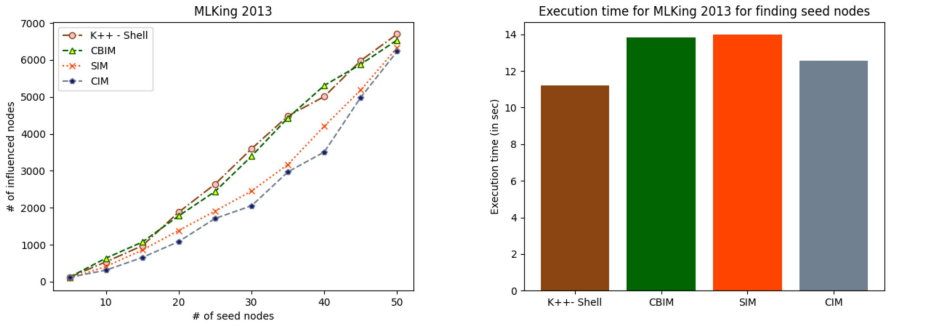
6.5 Summary

This chapter proposes the K++-Shell decomposition algorithm in multilayer networks. K++-Shell algorithm selects the seed nodes that influence more users than competing algorithms. K++-Shell exploits the clone structure of multilayer networks to select influential seed nodes. K++-Shell selects seed nodes efficiently. The proposed algorithms outperform the state-of-the-



(a) (b)

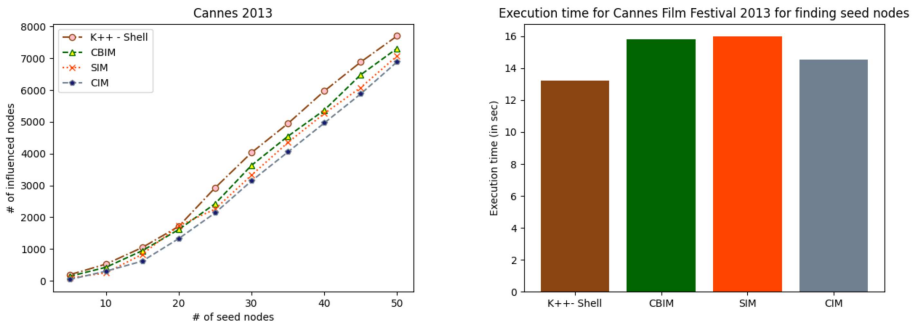
Figure 6.9: (a) Number of influenced people for 50 seed users for various algorithms using IC model on Newyork climate change 2014 dataset. (b) Execution time for various algorithms to find 50 seed nodes on Newyork climate change 2014 dataset.



(a) (b)

Figure 6.10: (a) Number of influenced people for 50 seed users for various algorithms using IC model on Martin Luther King's 50th anniversary 2013 dataset. (b) Execution time for various algorithms to find 50 seed users on Martin Luther King's 50th anniversary 2013 dataset.

art models in spreading information in multilayer networks.



(a) (b)

Figure 6.11: (a) Number of influenced people for 50 seed users for various algorithms using IC model on Cannes film festival 2013 dataset. (b) Execution time for various algorithms to find 50 seed users on Cannes film festival 2013 dataset.