NEW GRAVITATIONAL ALGORITHMS FOR THE DETECTION OF OVERLAPPING AND DISJOINT COMMUNITIES IN WEIGHTED COMPLEX NETWORKS

Nermin Kartli, Pelin Cetin, Selin Ayhan

1. INTRODUCTION

A complex network is a graph. Recently, researchers have started to call graphs of huge size as complex networks. One reason for this is that the research topic in these large graphs is different from classical graph theory problems. In complex networks, the aim is often to reveal hidden relationships, along with clustering according to a certain criterion. Complex networks can be encountered in many areas such as social media, drug discovery, natural language processing, etc. For example, a social media manager needs to be able to divide users into certain groups, expand the social network, suggest new friends or new jobs to the user, and send the same ads to similar people. These groups that are desired to be found in complex networks are called communities. There have been serious attempts to mathematically define the concept of community, but unfortunately, none of these definitions reflect real-life problems. Moreover, almost all of these definitions do not properly represent the concept of community in the verbal sense, even for some simple graphs of small size. Various definitions of community are discussed in [1].

2. PRELIMINARIES

Recently, a similar neighborhood-based algorithm was developed by Cetin and Emrah Amrahov to find overlapping communities in unweighted complex networks [2]. This algorithm is designed in 3 stages. In the first stage, each vertex is considered as its neighbor, and the cosine similarity of neighboring vertices is calculated and assigned as weight to the edge connecting these vertices. In the second stage, each vertex is initially assigned to a community consisting only of itself, and the edges are sorted in decreasing order of weight. Then, the edges are considered one by one in this order, and one end of the edge is added to one of the communities of the other according to a certain criterion called common neighbors. The criterion $CN(u, v)$, called common neighbors, is defined by the following formula:

$$CN(u, v) = \max_{C_v \in C} |C_v \cap n_u| \tag{1}$$

DOI:

Here, C is the set of communities formed up to the step under consideration, C_v is a community of the vertex v, and n_u is the set of all neighbors of the vertex u, including itself.

In this stage, if at least one of the vertices u and v for the considered edge uv, does not have a community that does not include the other vertex, the next edge is considered on without performing any action. Otherwise, the first $CN(u, v)$ is calculated to find C_v*, for which the maximum gained in formula (1). Then $CN(v, u)$ is calculated to find C_u*, for which the maximum gained in formula (1). If $CN(u, v*) > CN(v, u*)$ then vertex u is added to the community C_v*. If there is more than one community C_v* that gives the maximum to formula (1), vertex u is added to all of these communities. If $CN(u, v*) < CN(v, u*)$ then vertex v is added to the community C_u*. If there is more than one community C_u* that gives the maximum to formula (1), vertex v is added to all of these communities.If $CN(u, v*) = CN(v, u*)$ then the degrees of the vertices u and v are checked. When the degrees of the vertices are different, the vertex with the lower degree is added to the community of the vertex with the higher degree. If the degrees are equal, the vertex v is added to the community C_u*.

The third stage of the algorithm is the merging of the communities found. If more than half of the elements of one of the two communities considered are also elements of the other, or if one of the communities has only 2 elements and one of these elements belongs to the community of the other, these communities are merged into one community. The pseudo-code of the algorithm is given in Algorithm 1.

3. PROPOSED ALGORITHMS

3.1. Proposed overlapping community detection algorithm

In this subsection, we propose a new gravitational law-based algorithm for detecting overlapping communities in weighted complex networks. Our proposed algorithm uses some of the same steps of the recently proposed algorithm by Cetin and Emrah Amrahov for finding overlapping communities in unweighted complex networks, but it is a completely new algorithm since the gravitational law is not used in [2]. The first idea that comes to mind to adapt Algorithm 1 to weighted complex networks is straightforward. Let us delete lines 1-3 of the Algorithm 1 and use the given weights instead. With a simple example, let us show that this is not a good idea.

Example 3.1. The input graph is given in Fig 1. Let us assume that this graph corresponds to social media. Weights indicate the closeness of two vertices to each other. The lower the weight between two vertices, the closer these two vertices are to each other.

Since small weights mean that the vertices are closer to each other, let us consider the edges in increasing order in Algorithm 1. The result of applying the first two stages of Algorithm 1 is shown in Table 1.

Thus, after the first 2 stages of Algorithm 1, the following 4 communities are obtained:

$$C_1 = \{A, E, D, C, G, B\}$$

$$C_2 = \{E, H, C, G\}$$

Algorithm 1: detectCommunities(G)

Input : $G = (V, E)$ is an unweighted simple graph
Output : C is the list of communities

/* Stage 1: Initialization */
for *each* $(u, v) \in E$ **do**
  // calculate the similarity for each pair of adjacent vertices
  $w(u, v) \leftarrow |N_u \cap N_v| / \sqrt{|N_u||N_v|}$
end
$C \leftarrow \phi$
for *each* $v \in V$ **do**
  // add single vertex's community to the communities list
  $C \leftarrow C \cup \{v\}$
end
sort edges $(u, v) \in E$ according to $w(u, v)$ in descending order
/* Stage 2: Generating communities */
for *each edge* $(u, v) \in E$ **do**
  // if the vertices u and v have communities consisting only of themselves
  if $C_u = \{\{u\}\}$ *and* $C_v = \{\{v\}\}$ **then**
    // add $\{u, v\}$ to the community set and remove $\{u\}$ and $\{v\}$
    $C \leftarrow C \cup \{u, v\} - \{u\} - \{v\}$
  end
  else if *there are communities* $C_u \in C$ *and* $C_v \in C$ *such that* $v \notin C_u$ *and* $u \notin C_v$ **then**
    $CN(u, v) \leftarrow \max_{C_v \in C} |C_v \cap n_u|$
    $C_v^* \leftarrow arg\,max\ CN(u, v)$
    $CN(v, u) \leftarrow \max_{C_u \in C} |C_u \cap n_v|$
    $C_u^* \leftarrow arg\,max\ CN(v, u)$
    if $CN(u, v) > CN(v, u)$ *or* $(CN(u, v) \leftarrow CN(v, u)$ *and* $deg(u) < deg(v))$ **then**
      $C \leftarrow C - C_v^*$
      $C_v^* \leftarrow C_v^* \cup \{u\}$
      $C \leftarrow C \cup C_v^*$
      if $\{u\} \in C$ **then**
        $C \leftarrow C - \{u\}$
      end
    end
    else
      $C \leftarrow C - C_u^*$
      $C_u^* \leftarrow C_u^* \cup \{v\}$
      $C \leftarrow C \cup C_u^*$
      if $\{v\} \in C$ **then**
        $C \leftarrow C - \{v\}$
      end
    end
  end
end
/* Stage 3: Merging communities */
// $C = \{C_1, C_2, ...C_k\}$
$k \leftarrow length(C)$
sort the set C in according to the numbers of elements in the communities C_i in descending order
$l \leftarrow 1$
for $i = 2$ *to* k **do**
  for $j = l$ *downto* 1 **do**
    if $|C_i \cap C_j| > |C_i|/2$ *or* $(|C_i| = 2$ *and* $|C_i \cap C_j| = 1)$ **then**
      $C \leftarrow C - C_j - C_i$
      $C_i \leftarrow C_i \cup C_j$
      $C \leftarrow C \cup C_i$
    end
  end
  renumber communities in list C with indices less than or equal to i and assign the number of the renumbered communities to l
end
return C

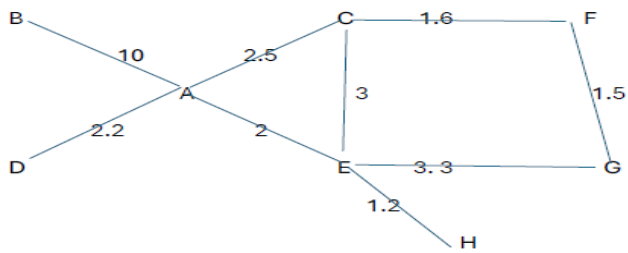

Fig. 1: Input Weighted Graph

Tab. 1: Applying first and second stages of Algorithm 1 to Example 3.1 with given weights

Step	Edge uv	w(u,v)	CN (u,v)	CN (v,u)	Degree	
0						{A},{B},{C},{D},{E},{F},{G},{H}
1	EH	1.2				{A},{B},{C},{D},{F},{G},{E,H}
2	FG	1.5				{A },{B},{C },{D},{F,G},{E,H}
3	CF	1.6	1	1	eq.	{A},{B},{C,F},{D},{F,G},{E,H}
4	AE	2	1	1	eq.	{A,E},{B},{C,F},{D},{F,G},{E,H}
5	AD	2.2	1	1	A	{A,E,D},{B},{C,F},{F,G},{E,H}
6	AC	2.5	1	2		{A,E,D,C},{B},{C,F},{F,G},{E,H}
7	CE	3	1	1	E	{A,E,D,C},{B},{C,F},{F,G},{E,H,C}
8	EG	3.5	1	1	E	{A,E,D,C,G},{B},{C,F},{F,G},{E,H,C,G}
9	AB	10	1	1	A	{A,E,D,C,G,B},{C,F},{F,G},{E,H,C,G}

$$C_3 = \{C, F\}$$
$$C_4 = \{F, G\}$$

After the third stage of Algorithm 1, a single community is formed for this graph that includes all the vertices of the graph. In other words, if we use the given weights directly, the entire graph forms a community.

Now let us assume that there is a gravitational force between the vertices of the graph, similar to Newton's universal law of gravitation. Let us assume that the mass of a vertex $m(u)$ is its degree, and the distance r between the vertices is the weight of the edge connecting these vertices. Thus, the gravitation force between adjacent vertices u and v is calculated by the following formula:

$$F(u, v) = \frac{kdeg(u)deg(v))}{w(uv)^2} \tag{2}$$

Here, k is the parameter of our proposed algorithm, which is a constant number for all vertices of the complex network when the algorithm is applied. This parameter can be thought of as the equivalent of the gravitational constant. The choice of this parameter affects the result of our proposed algorithm, so hyperparameter optimization should be done by experiments when the algorithm is applied.

We calculate the gravitational force between vertices u and v for all edges uv in the given complex network using the formula 2 and replace the weight of this edge with the found value. Now we can delete lines 1-3 of Algorithm 1 and use the calculated weights instead. After making this change, we apply Algorithm 1 to the Example 3.1. With our proposed modification, Algorithm 1 does not produce different results depending on the choice of parameter k. Therefore, we choose $k = 1$ in this example to simplify the computations. Applying Algorithm 1 we show in Table 2 step by step.

Tab. 2: Applying first and second stages of Algorithm 1 to Example 3.1 with gravitational weights

Step	Edge uv	F(u,v)	CN(u.v)	CN(v,u)	Degree	
0						{A},{B},{C},{D},{E},{F},{G},{H}
1	AE	4				{A,E},{B},{C},{D},{F},{G},{H}
2	CF	2.34				{A,E},{B},{C,F},{D},{G},{H}
3	EH	2.04	1	1	E	{A,E,H},{B},{C,F},{D},{G}
4	AC	1.92	1	2		{A,E,H,C},{B},{C,F},{D},{G}
5	FG	1.77	1	1	equal	{A,E,H,C},{B},{C,F,G},{D}
6	CE	1.33				{A,E,H,C},{B},{C,F,G},{D}
7	AD	0.826	1	1	A	{A,E,H,C,D},{B},{C,F,G}
8	EG	0.73	2	1		{A,E,H,C,D},{B},{C,F,G,E}
9	AB	0.04	1	1	A	{A,E,H,C,D,B},{C,F,G,E}

After applying the first and second stages of Algorithm 1 we obtain 2 overlapping

communities below:

$$C_1 = \{A, E, H, C, D, B\}$$

$$C_2 = \{C, F, G, E\}$$

The number of elements C_2 in common with C_1 is 2, which is not more than half of the number of elements in C_2, so there is no merge. As a result, the algorithm finds the communities $C_1 = \{A, E, H, C, D, B\}$ and $C_2 = \{C, F, G, E\}$. We show these communities in Fig 2.

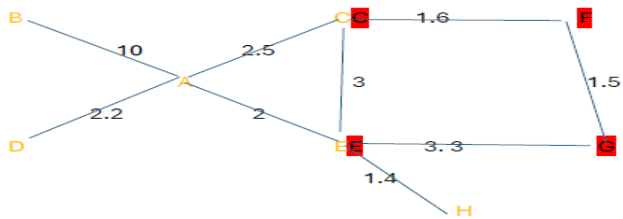

Fig. 2: Overlapping Communities after Applying Algorithm 1 to Example 3.1 with gravitational weights

Let us now discuss the formula 1. This criterion was proposed for unweighted complex networks in [2]. Since all edges are equivalent in weight for unweighted complex networks, it makes sense to use the formula 1, but the situation is different for weighted complex networks. We propose to use the function $CNW(u, v)$, which we will define below, instead of the function $CN(u, v)$ in Algorithm 1. The name of the function $CNW(u, v)$ is an abbreviation of "Common neighbors weighted".

For an edge uv, we define the function $CNW(u, v)$ as follows. For a community C_v, if the intersection of this community with the set n_u is nonempty, then it means that the vertex u is connected to some vertices in the set C_v. That is, the set $E_{uv} = \{e = ux | x \in C | v\}$ is nonempty. We call the sum of the gravitational weights of the edges in the set E_{uv} the gravitational weight of this set and denote it by $F(C_v)$. Since we include each vertex in the list of its neighbors, we emphasize that at least u vertex in the set $E_{uv} \cap n_u$, i.e. this set is not empty. We define the function $CNW(u, v)$ with the following formula:

$$CNW(u, v) = \max_{C_v \in C} F(C_v) \tag{3}$$

We propose to use the function $CNW(u, v)$ as the criterion in Algorithm 1. Consequently, we propose Algorithm 2 to find overlapping communities in complex networks whose weights imply the closeness of the vertices to each other.

The following example demonstrates the difference between using $CN(u, v)$ or $CNW(u, v)$ as criteria.

Algorithm 2: detectOverlappingCommunities(G)

Input : $G = (V, E)$ is an unweighted simple graph
Output : C is the list of communities

/* Stage 1: Initialization */
for *each* $(u, v) \in E$ **do**
  // calculate the gravitational weights for each edge
  $F(u, v) \leftarrow deg(u)deg(v)/w(u, v)^2$
end
$C \leftarrow \phi$
for *each* $v \in V$ **do**
  // add single vertex's community to the communities list
  $C \leftarrow C \cup \{v\}$
end
sort edges $(u, v) \in E$ according to $F(u, v)$ in descending order
/* Stage 2: Generating communities */
for *each edge* $(u, v) \in E$ **do**
  // if the vertices u and v have communities consisting only of themselves
  if $C_u = \{\{u\}\}$ *and* $C_v = \{\{v\}\}$ **then**
    // add $\{u, v\}$ to the community set and remove $\{u\}$ and $\{v\}$
    $C \leftarrow C \cup \{u, v\} - \{u\} - \{v\}$
  end
  else if *there are communities* $C_u \in C$ *and* $C_v \in C$ *such that* $v \notin C_u$ *and* $u \notin C_v$ **then**
    $CNW(u, v) \leftarrow \max_{C_v \in C} F(C_v)$
    $C_v^* \leftarrow arg\,max\ CNW(u, v)$
    $CNW(v, u) \leftarrow \max_{C_u \in C} F(C_u)$
    $C_u^* \leftarrow arg\,max\ CNW(v, u)$
    if $CNW(u, v) > CNW(v, u)$ *or* $(CNW(u, v) = CNW(v, u)$ *and* $deg(u) < deg(v))$ **then**
      $C \leftarrow C - C_v^*$
      $C_v^* \leftarrow C_v^* \cup \{u\}$
      $C \leftarrow C \cup C_v^*$
      if $\{u\} \in C$ **then**
        $C \leftarrow C - \{u\}$
      end
    end
    else
      $C \leftarrow C - C_u^*$
      $C_u^* \leftarrow C_u^* \cup \{v\}$
      $C \leftarrow C \cup C_u^*$
      if $\{v\} \in C$ **then**
        $C \leftarrow C - \{v\}$
      end
    end
  end
end
/* Stage 3: Merging communities */
// $C = \{C_1, C_2, ...C_k\}$
$k \leftarrow length(C)$
sort the set C in according to the numbers of elements in the communities C_i in descending order
$l \leftarrow 1$
for $i = 2$ *to* k **do**
  for $j = l$ *downto* 1 **do**
    if $|C_i \cap C_j| > |C_i|/2$ *or* $(|C_i| = 2$ *and* $|C_i \cap C_j| = 1)$ **then**
      $C \leftarrow C - C_j - C_i$
      $C_i \leftarrow C_i \cup C_j$
      $C \leftarrow C \cup C_i$
    end
  end
  renumber communities in list C with indices less than or equal to i and assign the number of the renumbered communities to l
end
return C

Example 3.2. The input graph is given in Fig 3. Let us assume that this graph corresponds to social media. Weights indicate the closeness of two vertices to each other. The lower the weight between two vertices, the closer these two vertices are to each other.

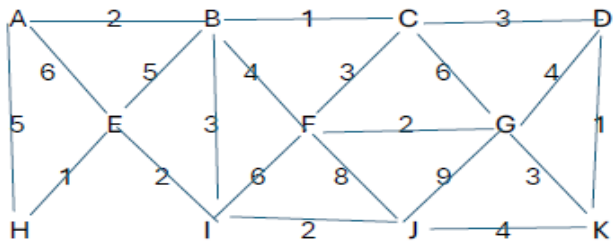

Fig. 3: Input Weighted Graph

Since the result of Algorithm 2 is not affected by the number k, we can choose $k = 1$.

Solution 1 (Detection of overlapping communities using Algorithm 1 in Example 3.2 with gravitational weights) The step-by-step application of the first and second stages of Algorithm 1 is shown in Table 3.

After applying the first 2 stages of the algorithm we obtain 4 communities:

$$C_1 = \{E, H, I, J, B, F, A\}$$

$$C_2 = \{D, K, G, J, C, F\}$$

$$C_3 = \{F, G, C, D, J\}$$

$$C_4 = \{A, B, C\}$$

After the third, merging stage of the algorithm, we have 2 communities:

$$C_1 = \{E, H, I, J, B, F, A, C\}$$

$$C_2 = \{D, K, G, J, C, F\}$$

The communities found are shown in Fig 4.

Solution 2 (Detection of overlapping communities using Algorithm 2 in Example 3.2 with gravitational weights) The step-by-step application of the first and second stages of Algorithm 2 is shown in Table 4.

After applying the first 2 stages of the Algorithm 2, we obtain 4 communities:

$$C_1 = \{E, H, I, J, B, F, A, G\}$$

$$C_2 = \{D, K, G, J, C, F\}$$

$$C_3 = \{A, B, C, F, G\}$$

$$C_4 = \{F, G\}$$

Tab. 3: Application of first and second stages of Algorithm1 to Example 3.2 with gravitational weights

Step	Edge uv	F (u,v)	CN (u,v)	CN (v,u)	Degree	Communities
0						{A},{B},{C},{D},{E},{F}, {G},{H},{I},{J},{K}
1	BC	20				{A},{B,C},{D},{E},{F},{G},{H},{I},{J},{K}
2	DK	9				{A},{B,C},{D,K},{E},{F},{G},{H},{I},{J}
3	EH	8				{A},{B,C},{D,K},{E,H},{F},{G},{I},{J}
4	FG	6.25				{A},{B,C},{D,K},{E,H},{F,G},{I},{J}
5	EI	4	1	1	eq.	{A},{B,C},{D,K},{E,H,I},{F,G},{J}
6	IJ	4	1	1	eq.	{A},{B,C},{D,K},{E,H,I,J},{F,G}
7	AB	3.75	1	1	B	{A,B,C},{D,K},{E,H,I,J},{F,G}
8	BI	2.2	2	1		{A,B,C},{D,K},{E,H,I,J,B},{F,G}
9	CF	2.2	2	2	F	{A,B,C},{D,K},{E,H,I,J,B},{F,G,C}
10	GK	1.67	2	1		{A,B,C},{D,K,G},{E,H,I,J,B},{F,G,C}
11	BF	1.56	2	3		{A,B,C},{D,K,G},{E,H,I,J,B,F},{F,G,C}
12	CD	1.33	2	2	C	{A,B,C},{D,K,G},{E,H,I,J,B,F},{F,G,C,D}
13	DG	0.94				{A,B,C},{D,K,G},{E,H,I,J,B,F},{F,G,C,D}
14	BE	0.8				{A,B,C},{D,K,G},{E,H,I,J,B,F},{F,G,C,D}
15	JK	0.75	2	1		{A,B,C},{D,K,G,J},{E,H,I,J,B,F},{F,G,C,D}
16	CG	0.56	2	1		{A,B,C},{D,K,G,J,C},{E,H,I,J,B,F},{F,G,C,D}
17	FI	0.56				A,B,C},{D,K,G,J,C},{E,H,I,J,B,F},{F,G,C,D}
18	AE	0.33	3	2		{A,B,C},{D,K,G,J,C},{E,H,I,J,B,F,A},{F,G,C,D}
19	FJ	0.31	3	2		{A,B,C},{D,K,G,J,C,F},{E,H,I,J,B,F,A},{F,G,C,D}
20	GJ	0.25	2	2	G	{A,B,C},{D,K,G,J,C,F},{E,H,I,J,B,F,A},{F,G,C,D,J}
21	AH	0.24				{A,B,C},{D,K,G,J,C,F},{E,H,I,J,B,F,A},{F,G,C,D,J}

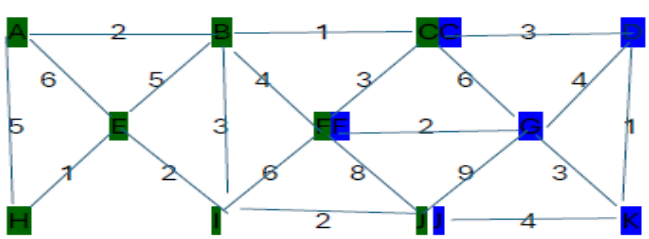

Fig. 4: Application of Algorithm 1 to Example 3.2 with gravitational weights

Tab. 4: Application of first and second stages of Algorithm 2 to Example 3.2 with gravitational weights

Step	Edge uv	F (uv)	CNW (uv)	CNW (vu)	Weight	Communities
0						{A},{B},{C},{D},{E},{F},{G},{H},{I},{J},{K}
1	BC	20				{A},{B,C},{D},{E},{F},{G},{H},{I},{J},{K}
2	DK	9				{A},{B,C},{D,K},{E},{F},{G},{H},{I},{J}
3	EH	8				{A},{B,C},{D,K},{E,H},{F},{G},{I},{J}
4	FG	6.25				{A},{B,C},{D,K},{E,H},{F,G},{I},{J}
5	EI	4	4	4	eq.	{A},{B,C},{D,K},{E,H,I},{F,G},{J}
6	IJ	4	4	4	eq.	{A},{B,C},{D,K},{E,H,I,J},{F,G}
7	AB	3.75	3.8	3.8	B	{A,B,C},{D,K},{E,H,I,J},{F,G}
8	BI	2.2	3	2.2		{A,B,C},{D,K},{E,H,I,J,B},{F,G}
9	CF	2.2	2.8	3.8		{A,B,C,F},{D,K},{E,H,I,J,B},{F,G}
10	GK	1.67	2.6	1.7		{A,B,C,F},{D,K,G},{E,H,I,J,B},{F,G}
11	BF	1.56	1.56	2.4		{A,B,C,F},{D,K,G},{E,H,I,J,B,F},{F,G}
12	CD	1.33	1.9	1.33		{A,B,C,F},{D,K,G,C},{E,H,I,J,B,F},{F,G}
13	DG	0.94				{A,B,C,F},{D,K,G,C},{E,H,I,J,B,F},{F,G}
14	BE	0.8				{A,B,C,F},{D,K,G,C},{E,H,I,J,B,F},{F,G}
15	JK	0.75	1	0.75		{A,B,C,F},{D,K,G,C,J},{E,H,I,J,B,F},{F,G}
16	CG	0.56	2.76	6.81		{A,B,C,F,G},{D,K,G,C,J},{E,H,I,J,B,F},{F,G}
17	FI	0.56				{A,B,C,F,G},{D,K,G,C,J},{E,H,I,J,B,F},{F,G}
18	AE	0.33	4.32	1.13		{A,B,C,F,G},{D,K,G,C,J},{E,H,I,J,B,F,A},{F,G}
19	FJ	0.31	8.76	0.56		{A,B,C,F,G},{D,K,G,C,J,F},{E,H,I,J,B,F,A},{F,G}
20	GJ	0.25	6.5	0.56		{A,B,C,F,G},{D,K,G,C,J,F},{E,H,I,J,B,F,A,G},{F,G}
21	AH	0.24				{A,B,C,F,G},{D,K,G,C,J,F},{E,H,I,J,B,F,A,G},{F,G}

After the third, merging stage of Algorithm 2, the following 2 communities are found:

$$C_1 = \{E, H, I, J, B, F, A, G, C\}$$

$$C_2 = \{D, K, G, J, C, F, A, B\}$$

We show these communities in Fig. 5.

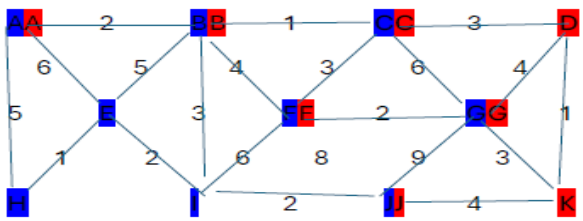

Fig. 5: Overlapping Communities after Applying Algorithm2 to Example 3.2 with gravitational weights

3.2. Proposed disjoint community detection algorithm

In this subsection, we explain our proposed disjoint community detection algorithm. The main idea of our proposed algorithm is as follows: First, we find overlapping communities with the help of Algorithm 2. Then, we consider vertices that are in more than one community one by one. For example, let vertex u be in k communities, namely $u \in C_1$, $u \in C_2$,..., $u \in C_k$. We calculate the gravity force by which u is included in each of these communities. For each i, we find the sum of the gravitational weights of the edges formed by the vertex u and the vertices in the community C_i. For whichever community this sum is the largest, we keep the vertex u in that community and delete it from all other communities. If there is more than one community where the maximum sum is obtained, we randomly choose one of these communities to hold the vertex u. If there is more than one community where the maximum sum is obtained, we randomly select one of these communities to hold the vertex u.

Let us explain the proposed algorithm on the Example 3.2. After applying Algorithm 2, we obtain two overlapping communities:

$$C_1 = \{E, H, I, J, B, F, A, G, C\}$$

$$C_2 = \{D, K, G, J, C, F, A, B\}$$

The vertices A, B, C, F, G, J are in both communities. The force holding the vertex A in the community C_2 is $F(AB)$. The force holding this vertex in the community C_1 is $F(AB) + F(AE) + F(AH)$. Since this vertex is more strongly present in the community C_1, we delete it from C_2 and keep it in C_1. The gravity force on the vertex B by the vertices in C_2 is equal to $F(BA) + F(BF) + F(BC)$. The gravity force on this vertex

by the vertices in C_1 is equal to $F(BA) + F(BE) + F(BF) + F(BI) + F(BC)$. This means that the vertex B is expelled from C_2.Proceeding similarly, we delete the vertices C and G from C_1, and the vertices F and J from C_2.As a result, we obtain the following disjoint communities:

$$C_1 = \{E, H, I, J, B, A, F\}$$

$$C_2 = \{D, K, G, C\}$$

We show these disjoint communities in Fig. 6.

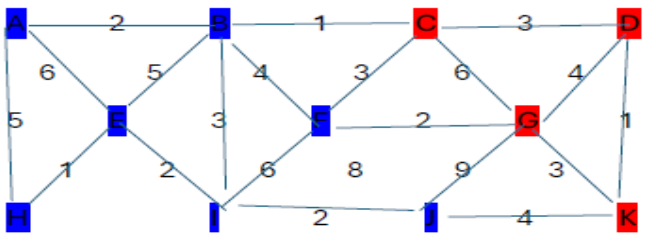

Fig. 6: Disjoint Communities after Applying Algorithm 3 to Example 3.2 with gravitational weights

To write the pseudocode of the proposed algorithm, let us denote by C_u the community to which a vertex u belongs. We define the gravitational force acting on the vertex u by the vertices in the community C_u as follows:

$$F(C_u) = \sum_{v \in C_u} F(uv) \tag{4}$$

The pseudo-code of the proposed algorithm to find disjoint communities is given in Algorithm 3.

Algorithm 3: detectDisjointCommunities(G)

Input : $G = (V, E)$ is an weighted simple graph
Output : C is the list of disjoint communities

`/* Stage 1: Finding of overlapping communities */`
Call Algorithm 2($G = (V, E)$)
`/* Stage 2: Generating disjoint communities */`
for *each vertex* $u \in V$ **do**
  `// We count the number of communities that the vertex` u `is included in and assign it to the variable` s.
  $s \leftarrow |\{C_u : C_u \in C\}|$
  if $s > 1$ **then**
    $F_u \leftarrow \max_{C_u \in C} F(C_u)$
    $C_u^* = arg\,max\ F_u$
    Delete vertex u from all C_u communities except C_u^*
  end
end
return C

REFERENCES

[1] S. E. Amrahov, B. Tugrul: A community detection algorithm on graph data. In: 2018 International Conference on Artificial Intelligence and Data Processing (IDAP) IEEE (2018) 1–4. DOI:10.1109/IDAP.2018.8620850

[2] P. Cetin, S. Emrah Amrahov: A new overlapping community detection algorithm based on similarity of neighbors in complex networks. Kybernetika *58(2)* (2022) 277–300. DOI: 10.14736/kyb-2022-2-0277