Detecting communities in directed acyclic networks using modified LPA algorithms

Suzana Antunović
University of Split, Faculty of Civil Engineering, Architecture and Geodesy, Split, Croatia

Damir Vukičević
University of Split, Faculty of Science, Split, Croatia

Abstract

Networks (or graphs) appear as dominant structures in different domains, including sociology, biology, neuroscience and computing. In most cases, these graphs are directed which changes the semantics of the edges that are no longer symmetrical in the sense that the beginning vertex transfers some property or value to the end vertex, but not vice versa. Detecting community structure in complex networks is an interdisciplinary topic with many relevant areas of application. In order to detect communities in directed acyclic networks, apart from the direction of the edge, the requirement for topological ordering of the vertices should be taken into account. In other words, if the vertices are topologically ordered in such a way that $x_1 < x_2 < ... < x_n$ we are interested in dividing the network into communities $C_1, C_2, ..., C_k$ in such a way that:

if $x_i < x_j$, $x_i \in C_i, x_j \in C_j$ then $C_i < C_j$ or $C_i = C_j$

We present an algorithm derived from LPA algorithms which are commonly used in network detection, mostly because of their quick computational time and fairly good results. They were originally developed for undirected networks, but have been modified for this purpose.

Keywords: community detection, directed acyclic network, label propagation algorithm

MSC: 90C35, 90C27, 05C82, 05C85, 97K30

DOI: https://doi.org/10.5592/CO/CCD.2018.01
1 Introduction

In complex networks theory, community detection problem refers to grouping vertices into communities according to their similarity, which usually considers either topological features or other characteristics related to the vertices and edges of the graph. Finding communities in directed networks is a challenging task with several important applications in a wide range of domains. However, the problem has mainly been considered and studied for the case of undirected networks. The problem of community detection in directed networks is considered to be a more challenging task as compared to the undirected case [9]. In order to detect communities in directed acyclic networks, apart from the direction of the edge, the requirement for topological ordering of the vertices should be taken into account. Topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge $uv$ from vertex $u$ to vertex $v$, $u$ comes before $v$ in the ordering. A topological ordering is possible if and only if the graph has no directed cycles, that is, if it is a directed acyclic graph [1].

Let $G$ be a simple directed graph with no directed cycles and $P(G)$ the set of all bijections $p : V(G) \rightarrow \{1, ..., n\}$ such that $p(u) < p(v)$ for every directed edge $uv \in E(G)$. There is at least one function $p$ in the set $P$ [5]. One of the possible interpretations for this is that vertices represent educational units and for each unit $u$, $p(u)$ represents the order in which the lesson is learned. All vertices that point to $u$ can be thought of as prerequisites for learning lesson $u$ and for all vertices $v$ such that there is a directed edge $uv$, lesson $u$ is a prerequisite for learning and understanding lesson $v$.

For vertex $v \in V$ we define in–neighbor of $v$ as vertex $u \in V$ such that there is a directed edge $uv \in E$. Analogly, we define out–neighbor of $v$ as vertex $u \in V$ such that there is a directed edge $vu \in E$.

2 LPA algorithms

Label Propagation Algorithm uses only the network structure as input data and is relatively fast so it is one of frequently used algorithms for community detection. In practice, the algorithm proved to be very efficient. However, since it involves random processes, in different executions, the algorithm provides different network partitions [7]. The basic idea of the algorithm is that, in each step of the algorithm, vertex $x$ selects the community to which the largest number of its neighbors belong. The propagation process is repeated iteratively until label changes are no longer
possible. The most significant feature of LPA is its computer complexity (near the linear time $O(m)$) \[11\]. The problem is that the LPA is not stable: the algorithm is susceptible to the order in which the vertices update the labels in each iteration, which is why the solution may be different in the various initiations of the algorithm \[7\]. Sometimes LPA can end up with a trivial solution - all vertices are identified in the same community \[2\]. Barber and Clark expanded the LPA by modifying the update rule and proposed a new algorithm called LPAm \[2\]. Instead of choosing the label that most often appears among his neighbors, vertex $x$ selects a label that will result in maximum modularity increase. Modularity is commonly used measure for evaluating the quality of network division. It measures the actual ratio of edges within the community reduced by the expected value in the null model, where the division is the same, but the edges are placed uniformly and random \[8\]. The implementation of LPAm brings a monotonous increase in modularity and avoids the possibility of forming a trivial solution. Additionally, LPAm has the same effective speed as LPA. However, the tendency is to get stuck at a low local maximum of modularity \[8\]. Liu and Murata overlook this problem by joining communities that maximize modularity the most. In this way, the local modularity maximum is avoided. However, it is not certain that the new local maximum that we will get is good enough (though better than the previous one). For this reason it is necessary to repeat this process as long as it is no longer possible to increase modularity. The proposed algorithm is called LPAm+.

3 OLPAm+ algorithm

The algorithm that we propose to detect communities in directed acyclic networks is a heuristic algorithm based on label propagation and the maximization of the modularity $Q_d$, defined in Eq. \[1\], which is defined for directed networks.

The basic idea is similar to the idea of LPA algorithms. Vertices take one of their neighbors’ labels, which maximally increases the modularity $Q_d$, but does not violate the requirement for partial arrangement of a set of the communities. Each step selects the optimal solution found on the basis of currently available information in the hope that the final solution will approach the global optimum. This algorithmic paradigm is known as a ”greedy algorithm” \[4\] and is often used in optimization problems.

Let us define modularity for directed networks. Let there be a directed network with $n$ vertices and $m$ directed edges represented by adjacency matrix $A$. Let $d^{in}(i)$ and $d^{out}(i)$ be in–degree and out–degree of a vertex $i \in V(G)$. Let vertex $i$ belong
to a community \( l_i \). Modularity for directed networks is defined as \([6]\)

\[
Q_d = \frac{1}{m} \sum_{1 \leq i,j \leq n} \left[ A_{ij} - \frac{d_{\text{in}}(j) d_{\text{out}}(i)}{m} \right] \delta(l_i, l_j)
\] (1)

where \( \delta(l_i, l_j) \) is Kronecker’s delta.

The algorithm can be described as follows. Each vertex \( i \in V(G) \) is assigned a unique numeric label \( l_i = p(i) \). The vertices are placed in random order and the process of label propagation begins. For each vertex in that order, the change in modularity caused by the change of label is calculated. From equation (1) it follows that modularity increase can be caused by those pairs of vertices \( i, j \in V(G) \) for which \( A_{ij} \neq 0 \) or \( A_{ji} \neq 0 \) so specific vertex can update its label to one of its neighbors’ labels (including in–neighbors and out–neighbors). When vertex \( i \) changes the label, i.e. changes the community to which it belongs, it causes modification of the modularity that can be calculated in the following way. Let \( i \) change the existing label \( l_i \) to the new label \( l_j \). The change in modularity caused by this change follows from equation (1) and is calculated as

\[
\Delta Q_d(ij) = \frac{d_i^j}{m} - \left[ \frac{d_{\text{out}}(i) S_{\text{in}}(j) + d_{\text{in}}(i) S_{\text{out}}(j)}{m^2} \right]
\] (2)

where

- \( d_i^j \) is the number of all neighbors of \( i \) with label \( l_j \)
- \( S_{\text{in}}(j) \) is the total in–degree of vertices with label \( l_j \)
- \( S_{\text{out}}(j) \) is the total out–degree of vertices with label \( l_j \)

The label update process is asynchronous. If the vertex \( i \) changes the label at iteration \( t \), some of its neighbors \( j \in V(G) \) have already changed the label in the current iteration and have the label \( l_j(t) \) while some neighbors \( k \in V(G) \) still have a label from the previous iteration \( l_k(t-1) \). The vertex \( i \) makes a decision based on the state it has found at iteration \( t \), i.e. selects the label that causes the maximum increase in modularity without disturbing the community order. If such label does not exist, the vertex retains its label. To ensure that the vertices (and the resulting communities) are in a valid order, vertex \( i \) can choose the largest among the labels of its in–neighbors or the smallest among the labels of its out–neighbors. Otherwise, the order will be disrupted.
When every vertex in the order is considered exactly once, the vertices are put in random order and the process is repeated. The algorithm stops when by changing the label it is no longer possible to get a positive change in modularity. At the end of the algorithm, we identify communities as groups of vertices that have the same label. The described algorithm was called Orientation Respecting LPAm (OLPAm). Pseudocode is available in the Algorithm [1].

As described in the last chapter, the LPAm algorithm is susceptible to the vertex order that is random in each iteration and is inclined to stuck in the poor local maximum modularity. The same goes for OLPAm. When this happens, we calculate the change in modularity that is generated by joining community pairs and merging the pair of communities resulting in the greatest increase in modularity and not disturbing the community order. If we merge communities with labels $l_i$ and $l_j$, the modularity change caused by merging can be calculated as

$$\Delta Q_d(l_i l_j) = \frac{E_{ij}}{m} - \left[ \frac{S_{\text{out}}(i)S_{\text{in}}(j) + S_{\text{in}}(i)S_{\text{out}}(j)}{m^2} \right]$$ (3)

where $E_{ij}$ is the number of edges between communities $l_i$ and $l_j$. The equation (3) is derived from (2) by summing over all the vertices in the community $l_i$. From equation (3) it is clear that it is sufficient to compute $\Delta Q_d(l_i l_j)$ for pairs of communities that are connected because only connected communities can make a positive change in modularity. Although merging communities increases modularity, it is not certain that the maximum is at the same time global, so again we call OLPAm and repeat the process as far as it is possible for the communities to get increased modularity without disturbing the community order.

In order to maintain a valid order of communities, consider the following. If there is at least one vertex $x_i$ in the community $A_i$ and at least one vertex $x_j$ in the community $A_j$ such that there exists an edge $x_i x_j \in E(G)$ we say that the community $A_i$ points to the community $A_j$. In the same way as with vertices, the community labeled $A$ can be merged with community labeled $A_i$ that has the largest label $l_{\text{max}}$ among the community labels that point to $A$ or with the community $A_j$ with the smallest label $l_{\text{min}}$ among the community labels to which $A$ points. Otherwise, the condition from the beginning of the chapter will not be met. The complete algorithm (OLPAm with merging communities) was called Orientation Respecting LPAm+ (OLPAm+). Pseudocode can be found in the Algorithm [2].

As far as the OLPAm computational complexity is concerned, one step of label propagation in OLPAm has the complexity of $O(n)$ since for each vertex we calculate two values so the total complexity of the OLPAm algorithm is equal to...
where \( r \) is the maximal number of steps label propagation needs to find the maximal modularity value. Furthermore, a method of merging communities in OLPAm+ algorithm has the complexity of \( O(n) \). Namely, \( n \) is the maximal number of communities obtained so for calculating all values \( \Delta Q_d(l_{ij}) \), the time required is \( O(n) \). Determining the maximal value of a string of \( n \) elements has the complexity of \( O(n) \), which gives the total complexity of one merging of communities \( O(n + n) = O(n) \). Let \( h \) the number of times the communities merge. The total computational complexity of the algorithm is \( O(rn + h[O(n) + O(rn)]) = O(n) \). The value of the \( h \) parameter can not be accurately estimated because it depends on the quality of the solution obtained in the OLPAm algorithm. Even the value of parameter \( r \) can not be fully predicted.

Table 1: Estimated values of \( r \) and \( h \) parameters in OLPAm+. Table shows the average number of steps \( r \) needed for the OLPAm to converge and the average number of times \( h \) of merging communities in OLPAm+.

<table>
<thead>
<tr>
<th>Network</th>
<th>( n )</th>
<th>( m )</th>
<th>( r )</th>
<th>( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number set ( \mathbb{Q} )</td>
<td>47</td>
<td>254</td>
<td>7.31</td>
<td>3.25</td>
</tr>
<tr>
<td>Elementary functions</td>
<td>84</td>
<td>502</td>
<td>6.09</td>
<td>2.64</td>
</tr>
<tr>
<td>Integral</td>
<td>223</td>
<td>656</td>
<td>11.02</td>
<td>5.51</td>
</tr>
<tr>
<td>Data processing</td>
<td>54</td>
<td>197</td>
<td>6.09</td>
<td>2.64</td>
</tr>
<tr>
<td>Primary production</td>
<td>28</td>
<td>93</td>
<td>7.11</td>
<td>1.23</td>
</tr>
<tr>
<td>Physics</td>
<td>31</td>
<td>49</td>
<td>4.77</td>
<td>1.99</td>
</tr>
</tbody>
</table>

When joining communities in the LPAm+ algorithm, we merged only those two communities whose merger results in the largest increase in modularity without distorting the order. Following the example of [5], we consider modifying the OLPAm+ algorithm. When OLPAm stops at the local maximum (further label propagation does not increase modularity), we calculate the modularity changes caused by merging communities and joining those pairs of communities that maximize modularity the most. The pseudo-code of this modified OLPAm+ algorithm is given in the Algorithm [3].
3. **OLPAM+ ALGORITHM**

Algorithm 1 Orientation Respecting LPAm (OLPAm)

**Require:** Edge list

**Ensure:** Community division, modularity

1: to each vertex $i$ assign a unique numerical label $l_i(0) = p(i)$
2: set $t = 1$
3: **repeat**
4: put vertices in random order $X$
5: **for** each vertex $i \in X$ do
6: among in–neighbors $x_{i1}, x_{i2}, \ldots, x_{ik}$ of vertex $i$ with labels $l_{i1}, l_{i2}, \ldots, l_{ik}$ find the largest label $l_{ik}$
7: among out–neighbors $x_{ik+1}, x_{ik+2}, \ldots, x_{in}$ of vertex $i$ with labels $l_{ik+1}, l_{ik+2}, \ldots, l_{in}$ find the smallest label $l_{in}$
8: calculate $\Delta Q_d(i, max)$ and $\Delta Q_d(i, min)$
9: **if** $\Delta Q_d(i, max) > \Delta Q_d(i, min)$ and $\Delta Q_d(i, max) > 0$ then
10: set $l_i(t) = l_{max}$
11: **else if** $\Delta Q_d(i, min) > \Delta Q_d(i, max)$ and $\Delta Q_d(i, min) > 0$ then
12: set $l_i(t) = l_{min}$
13: **else if** $\Delta Q_d(i, min) = \Delta Q_d(i, max) > 0$ then
14: uniformly at random pick $l_{max}$ or $l_{min}$ and set it for $l_i$
15: **end if**
16: set $t = t + 1$
17: **end for**
18: **if** neither of vertices $i \in X$ changes its label **then**
19: end algorithm
20: **else**
21: set $t = t + 1$
22: **end if**
23: **until** neither vertex in the iteration changes its label
Algorithm 2 Orientation Respecting LPAm+ (OLPAm+)
1: assign to each vertex a unique numeric label
2: using OLPAm algorithm maximize modularity $Q_d$
3: \textbf{while} there are communities $A_i$ and $A_j$ such that $\Delta Q_d(l_i,l_j) > 0$ \textbf{do}
4: \hspace{1em} \textbf{for} each community $A_i$ \textbf{do}
5: \hspace{2em} calculate $\Delta Q_d(l_{i \text{max}})$ and $\Delta Q_d(l_{i \text{min}})$
6: \hspace{2em} \textbf{end for}
7: \hspace{1em} find the maximal value of all $\Delta Q_d(l_i,l_j) > 0$
8: \hspace{1em} merge communities $A_i$ and $A_j$ such that $\Delta Q_d(l_i,l_j) > 0$ is maximal
9: \hspace{1em} maximize modularity $Q_d$ using OLPAm algorithm
10: \textbf{end while}

Algorithm 3 Modified OLPAm+ with multiple merging of communities
1: assign to each vertex a unique numeric label
2: using OLPAm algorithm maximize modularity $Q_d$
3: \textbf{while} $\exists$ pair of communities $(A_i, A_j)$ such that $\Delta Q(l_i,l_j) > 0$ \textbf{do}
4: \hspace{1em} \textbf{for} each pair of connected communities $(A_i, A_j)$ where $\Delta Q(l_i,l_j) > 0$ \textbf{do}
5: \hspace{2em} \textbf{if} there is no community $A_l$ labeled $l$ such that $\Delta Q(l,l_i) > \Delta Q(l_i,l_j)$ and $\Delta Q(l,l_j) > \Delta Q(l_i,l_j)$ \textbf{then}
6: \hspace{3em} merge communities $A_i$ and $A_j$
7: \hspace{2em} \textbf{end if}
8: \hspace{1em} \textbf{end for}
9: \hspace{1em} maximize modularity $Q_d$ using OLPAm algorithm
10: \textbf{end while}

3.1 Experiments and results
The proposed algorithm is implemented in Microsoft Visual Studio 2015 program tool. Since random processes are used to update the label, the results may vary in each execution of the algorithm. For this reason, based on the example of [8], we ran the algorithm 100 times for each of the networks described below.

3.1.1 Data sets
The OLPAm+ algorithm was originally developed for curriculum networks, directed acyclic networks where vertices represent educational units and directed edge from
vertex \( u \) to \( v \) means that unit \( u \) is necessary for learning and understanding unit \( v \). In order to better analyze the structure and community division, experts from different fields in science and education (mathematics, physics, computer science and biology) were asked to create a network for this purpose. The networks are named by the key concept whose understanding is set as a learning objective for that area. Some basic statistics of curriculum networks can be found in Tables 2 and 3.

"Number set \( \mathbb{Q} \)" is a curriculum network with 47 vertices and 254 directed edges. Vertex labeled 1 denotes the term natural number, vertex labeled 47 represents the term set of rational numbers.

"Elementary functions" is a network of terms needed for understanding the term elementary functions. Network has 84 vertices and 502 directed edges. Vertex labeled 1 is set, vertex labeled 82 is elementary function.

"Integral " is a curriculum network of terms needed for passing the introductory course in mathematics. Network has 223 vertices and 655 directed edges. Vertex labeled 1 is set, vertex labeled 223 is improper integral.

"Physics" is curriculum network of topics covered in 7th grade. Network has 31 vertices and 49 directed edges. Vertex labeled 1 represents the term length, vertex labeled 31 is the strength of the lens.

"Data processing" is curriculum network of topics covered in the introductory college course. Network has 54 vertices and 197 directed edges. Vertex labeled 1 represents the term data, vertex labeled 31 is data processing.

"Primary production" is a curriculum network of terms needed for understanding the process of primary production in oceans. Network has 28 vertices and 93 directed edges. Vertex labeled 1 is photosynthesis, vertex labeled 28 is primary production.
Table 2: Basic statistics for curriculum networks. Notation: number of vertices $n$, number of directed edges $m$, largest in–degree $d_{in}$, largest out–degree $d_{out}$, average degree $d_{avg}$, average shortest path length $l$ for pairs of connected vertices, clustering coefficient $C$. Measures used are defined in [10].

<table>
<thead>
<tr>
<th>Network</th>
<th>$n$</th>
<th>$m$</th>
<th>$d_{in}$</th>
<th>$d_{out}$</th>
<th>$d_{avg}$</th>
<th>$l$</th>
<th>$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number set</td>
<td>47</td>
<td>254</td>
<td>17</td>
<td>26</td>
<td>5.404</td>
<td>2.011</td>
<td>0.254</td>
</tr>
<tr>
<td>Elementary functions</td>
<td>84</td>
<td>502</td>
<td>27</td>
<td>51</td>
<td>5.976</td>
<td>2.132</td>
<td>0.255</td>
</tr>
<tr>
<td>Integral</td>
<td>223</td>
<td>655</td>
<td>15</td>
<td>28</td>
<td>2.941</td>
<td>3.899</td>
<td>0.084</td>
</tr>
<tr>
<td>Physics</td>
<td>31</td>
<td>49</td>
<td>4</td>
<td>8</td>
<td>1.581</td>
<td>1.575</td>
<td>0.049</td>
</tr>
<tr>
<td>Primary production</td>
<td>28</td>
<td>93</td>
<td>9</td>
<td>14</td>
<td>3.321</td>
<td>2.135</td>
<td>0.183</td>
</tr>
<tr>
<td>Data processing</td>
<td>54</td>
<td>197</td>
<td>12</td>
<td>22</td>
<td>3.648</td>
<td>1.744</td>
<td>0.338</td>
</tr>
</tbody>
</table>

Table 3: Other basic statistics for curriculum networks. Notation: diameter $diam$, density $D$, maximal value of betweenness centrality $c$, maximal value of hub centrality $h$, maximal value of authority centrality $a$.

<table>
<thead>
<tr>
<th>Network</th>
<th>$diam$</th>
<th>$D$</th>
<th>$c$</th>
<th>$h$</th>
<th>$a$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number set</td>
<td>5</td>
<td>0.117</td>
<td>117.940</td>
<td>0.1219</td>
<td>0.0461</td>
</tr>
<tr>
<td>Elementary functions</td>
<td>6</td>
<td>0.072</td>
<td>309.429</td>
<td>0.1125</td>
<td>0.0313</td>
</tr>
<tr>
<td>Integral</td>
<td>10</td>
<td>0.013</td>
<td>2041.796</td>
<td>0.0881</td>
<td>0.0489</td>
</tr>
<tr>
<td>Physics</td>
<td>4</td>
<td>0.053</td>
<td>12.000</td>
<td>0.2402</td>
<td>0.1066</td>
</tr>
<tr>
<td>Primary production</td>
<td>5</td>
<td>0.123</td>
<td>66.089</td>
<td>0.1764</td>
<td>0.0885</td>
</tr>
<tr>
<td>Data processing</td>
<td>5</td>
<td>0.069</td>
<td>79.000</td>
<td>0.1463</td>
<td>0.0682</td>
</tr>
</tbody>
</table>
3.1.2 Results

In all following tables, \( n \) denotes the number of vertices and \( m \) denotes the number of directed edges.

We have compared the results obtained with the OLPAm+ algorithm with community division proposed by the authors of each of the networks. For the proposed division for each network, we calculated the modularity values of \( Q_d \). The results can be found in Table 4. For each network, the algorithm gives higher values of modularity or better division into communities.

Table 4: Comparison of results from OLPAm+ algorithm with community division proposed by the creators of the networks. Notation: \( Q_{\text{max}} \) is the maximal value of modularity obtained, \( N_c \) is the number of communities which gives the value \( Q_{\text{max}} \), \( Q_d \) is the value of modularity calculated for community division proposed by the creators.

<table>
<thead>
<tr>
<th>Notation</th>
<th>( n )</th>
<th>( m )</th>
<th>( Q_{\text{max}} )</th>
<th>( N_c )</th>
<th>( Q_d )</th>
<th>( N_c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number set ( Q )</td>
<td>47</td>
<td>254</td>
<td>0.377</td>
<td>4</td>
<td>0.311</td>
<td>5</td>
</tr>
<tr>
<td>Elementary functions</td>
<td>84</td>
<td>502</td>
<td>0.354</td>
<td>4</td>
<td>0.239</td>
<td>6</td>
</tr>
<tr>
<td>Integral</td>
<td>223</td>
<td>655</td>
<td>0.468</td>
<td>7</td>
<td>0.455</td>
<td>10</td>
</tr>
<tr>
<td>Data processing</td>
<td>54</td>
<td>197</td>
<td>0.426</td>
<td>5</td>
<td>0.389</td>
<td>6</td>
</tr>
<tr>
<td>Primary production</td>
<td>28</td>
<td>93</td>
<td>0.293</td>
<td>3</td>
<td>0.237</td>
<td>3</td>
</tr>
<tr>
<td>Physics</td>
<td>31</td>
<td>49</td>
<td>0.476</td>
<td>6</td>
<td>0.238</td>
<td>6</td>
</tr>
</tbody>
</table>

Since we were not familiar with any algorithm that requires the arrangement of a set of obtained communities, we compared the values obtained by the proposed OLPAm+ algorithm with the values obtained with the LPAm+ algorithm [8] and the Girvan–Newman algorithm [3] as follows. The LPAm+ algorithm was run 100 for each of the networks and the maximum value of the modularity \( Q \) was observed. For the resulting community division, we calculated the modularity values for the directed networks \( Q_d \) in the formula (1). The same was done for division by the Girvan–Newman algorithm. Although this is a disregard for the community order requirement, we can compare how much that demand affects the value of modularity and network division quality. The comparison of results is shown in Table 5.
Table 5: **Comparation of the results obtained from several community detection algorithms** Table shows results obtained from running LPAm+ i OLPAm+ algorithms 100 times and Girvan–Newman algorithm for each of the networks. **Notation:** $Q_{\text{max}}$ is the maximal value of modularity $Q_d$ obtained, $N_c$ is the number of communities which gives the value $Q_{\text{max}}$, $\text{OLPAm} + (m)$ refers to multiple merging of communities.

<table>
<thead>
<tr>
<th></th>
<th>LPAm+</th>
<th></th>
<th>OlPAm+</th>
<th>OlPAm+ + (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$Q_d$</td>
<td>$N_c$</td>
<td>$Q_d$</td>
<td>$N_c$</td>
</tr>
<tr>
<td>Number set Q</td>
<td>0.376</td>
<td>4</td>
<td>0.367</td>
<td>4</td>
</tr>
<tr>
<td>El. functions</td>
<td>0.361</td>
<td>4</td>
<td>0.223</td>
<td>4</td>
</tr>
<tr>
<td>Integral</td>
<td>0.567</td>
<td>7</td>
<td>0.542</td>
<td>12</td>
</tr>
<tr>
<td>Data processing</td>
<td>0.477</td>
<td>4</td>
<td>0.438</td>
<td>4</td>
</tr>
<tr>
<td>Pr. production</td>
<td>0.297</td>
<td>3</td>
<td>0.099</td>
<td>5</td>
</tr>
<tr>
<td>Physics</td>
<td>0.457</td>
<td>7</td>
<td>0.377</td>
<td>8</td>
</tr>
</tbody>
</table>

It can be seen that OLPAm+ gives quite good results compared to the other two algorithms mentioned, although, in part, is limited by the requirement for a valid set of established communities.

**Conclusion**

\(\text{OLPAm+}\) is a greedy algorithm for detecting communities in directed acyclic networks under the following condition: if the vertices are topologically ordered in such a way that \(x_1 < x_2 < \ldots < x_n\) algorithm divides the network into communities \(C_1, C_2, \ldots, C_k\) in such a way that:

\[
\text{if } x_i < x_j, x_i \in C_i, x_j \in C_j \text{ then } C_i < C_j \text{ or } C_i = C_j
\]

Algorithm has been tested on curriculum networks described in Section 3.1.1, and given results were compared with other community detection algorithms. We conclude that the \(\text{OLPAm+}\) algorithm gives very good results in terms of modularity for directed networks $Q_d$ defined in Eq. [1]. It is also very efficient. The total computational complexity of the algorithm is $O(rn) + h[O(n) + O(rn)] = O(n)$. 

12
The value of the $h$ and $r$ parameters can not be accurately estimated because it depends on the quality of the solution obtained in the OLPAm algorithm as explained in Section 3.

References


