Label propagation algorithm: a semi-synchronous approach

Gennaro Cordasco*

Dipartimento di Psicologia,
Second University of Naples,
Caserta 81100, Italy
E-mail: gennaro.cordasco@unina2.it
*Corresponding author

Luisa Gargano

Dipartimento di Informatica,
University of Salerno,
Fisciano 84084, Italy
E-mail: lg@dia.unisa.it

Abstract: A recently introduced novel community detection strategy is based on a label propagation (LP) algorithm which uses the diffusion of information in the network to identify communities. Studies of LP algorithms showed that the strategy is effective in finding a good community structure. Label propagation step can be performed in parallel on all nodes (synchronous model) or sequentially (asynchronous model); both models present some drawback, e.g., algorithm termination is not granted in the first case, performances can be worst in the second case. In this paper, we present a semi-synchronous version of LP algorithms which aims to combine the advantages of both synchronous and asynchronous models. We prove that our models always converge to a stable labelling. Moreover, we experimentally investigate the effectiveness of the proposed strategy comparing its performance with the asynchronous model both in terms of quality, efficiency and stability. Tests show that the proposed protocol does not harm the quality of the partitioning. Moreover, it is quite efficient; each propagation step is extremely parallelisable and it is more stable than the asynchronous model, thanks to the fact that only a small amount of randomisation is used by our proposal.

Keywords: community discovery; label propagation algorithms; distributed algorithms; graph colouring.

Reference to this paper should be made as follows: Cordasco, G. and Gargano, L. (2012) 'Label propagation algorithm: a semi-synchronous approach', *Int. J. Social Network Mining*, Vol. 1, No. 1, pp.3–26.

Biographical notes: Gennaro Cordasco received his Laurea degree (cum laude) in Computer Science from the University of Salerno (Italy) in 2002. From 2002 to 2006, he attended the PhD programme in Computer Science at the University of Salerno. From March 2005 to November 2005, he has been a research-scholar at the University of Massachusettes at Amherst (USA), doing research activity under the supervision of Professor Arnold L. Rosenberg. He received his PhD in April 2006. From May 2006 to March 2011, he has been a Post-Doctoral Fellow at the University of Salerno. His current position is an Assistant Professor at the Seconda Università degli studi di Napoli (Italy).

Luisa Gargano received her Laurea degree (cum laude) in Computer Science from the University of Salerno in 1983. Since 1994, she is a Professor of Computer Science at the University of Salerno. She has done research in several areas like coding theory, information theory and its applications to combinatorics and theoretical computer science. She has been a visiting researcher at several institution, including Columbia University, University of Bielefeld, CNRS and INRIA. She has been involved in several research projects. She served as program committee member of several conferences and as a Referee for many of the major journals and conferences in computer science.

This paper is a revised and expanded version of a paper entitled 'Community detection via semi-synchronous label propagation algorithms' presented at International Workshop on Business Applications of Social Network Analysis (BASNA 2010), Bangalore, India, 15 December 2010.

1 Introduction

Collaboration networks, the internet, the World Wide Web, biological networks, communication and transport networks, social networks are just some examples of wide complex networks. An interesting property to investigate, typical to many such networks, is the community structure, i.e., the division of networks into groups of nodes that are similar among them but dissimilar from the rest of the network. The capability of detecting the partitioning of a network into communities can give important insights into the organisation and behaviour of the system that the network models.

The discovery and analysis of communities is useful for data mining in different contexts. It allows to characterise the strength of ties (weak or strong) in a social network. Moreover, it allows to analyse the diffusion of innovation (a.k.a. cascading behaviour); indeed it can be used to bound the cascade capacity of a given network (Easley and Kleinberg, 2010).

The generally adopted notion of community structure in complex networks (Girvan and Newman, 2002) refers to the fact that nodes in many real networks appear to group into sub-graphs which are densely inter-connected and sparsely connected to other parts of the network. The quite challenging problem of community detection has attracted ample attention in recent years and community detection methods have been applied in a wide range of scientific problems (Albert and Barabási, 2002).

Several community detection algorithms have been proposed in the literature; they are typically classified in: *divisive* algorithms (Girvan and Newman, 2002), *agglomerative* algorithms (Newman and Girvan, 2004; Comellas and Miralles, 2010) (depending on whether they focus on the addition or removal of edges to or from the network), and *optimisation* algorithms (Brandes et al., 2007) which continuously update the network partition in order to maximise the quality of the partition according to a given metric. Optimisation algorithms include several approaches: greedy routing (Newman, 2004), simulated annealing (Guimera et al., 2004), spectral optimisation (Newman, 2006a), game-theoretic (Narayanam and Narahari, 2010; Chen et al., 2010), compression-based (Rosvall and Bergstrom, 2007) and flow-based (Wu and Huberman, 2004). Reviews of the various methods present in the literature can be found in Danon et al. (2005) and Orman and Labatut (2009).

Recently, Raghavan et al. (2007) proposed a label propagation (LP) algorithm for detecting network communities. This algorithm uses only the network structure as a guide, and can be summarised as follows: each node in the network is initially given a unique label; at each iteration, each node is updated by choosing the label which is the most frequent among its neighbours – if multiple choices are possible (as for example in the beginning), one among the candidate labels is picked randomly. Experiments have shown that the LP technique is very effective in discovering accurate community structure.

It was noticed in Raghavan et al. (2007) that if node labels are updated at the same time at each iteration, then the process may result in a cyclic oscillation of the labels of some vertices thus precluding the convergence (termination) of the algorithm. For this reason, in Raghavan et al. (2007) the authors suggest to use an asynchronous LP approach, which means to update the label of only one node at time. However, the asynchronous version of the algorithm is much more time consuming; indeed, it takes $O(m)$ time for each iteration vs. $O(d)$ time required by a synchronous implementation, where m and d denote respectively the number of links in the network and the maximum degree over all the nodes in the network.

Since many complex networks are large in size, the efficiency of the community detection algorithm is an important aspect that deserves consideration. A way to provide efficient algorithms is by exploiting the parallelism which characterises novel computer platforms. Nowadays, platform based on multi-core chips are becoming dominant systems. Similarly, several architectures for distributed computing (where data processing and data storing are cooperatively performed on several nodes, interconnected by a network) are evolving. The key to leveraging this hardware trend will be, of course, in the algorithms and on the *degree of parallelism* they provide.

Moreover, using the asynchronous approach, randomisation is used both in the initial label assignment and, at each iteration, for determining the order in which nodes labels are updated; this can have some negative effects on the detected community structure; hence, a number of algorithm runs are needed before one can extract the desired result.

Liu and Murata (2009) introduced a variation of the LP algorithm that allows updating simultaneously all the nodes belonging to one of the partitions of a bipartite network. The authors experience that by running this version of the LP algorithm, the oscillation problem disappears. However, no formal proof (or an informal hint) that shows why their proposal is able to defeat the oscillation issue has been provided.

In this paper we extend the approach in Liu and Murata (2009) to any graph. Our strategy allows us to retain the benefit provided by the synchronous approach (i.e., high degree of parallelism and less randomisation) while eliminating its drawbacks (i.e., labels oscillations). In particular, we are able to *formally* show that our technique always converges to a stable labelling. We apply our technique to several well-known complex networks in order to validate its effectiveness, efficiency and stability.

A preliminary version of some results described in this paper are reported in Cordasco and Gargano (2010) where a semi-synchronous LP algorithm has been proposed and a preliminary empirical evaluation of the algorithm have been provided. In this paper we provide a deeper evaluation of the semi-synchronous LP algorithm (a wider dataset of networks and several evaluation metrics have been introduced) and characterise the period of convergence of the algorithm according to several tie resolution strategies not analysed in Cordasco and Gargano (2010).

The rest of the paper is organised as follows: Section 2 describes the notation used in the paper and introduces the measures that will be used to evaluate the qualities of the discovered community structures. In Section 3, we briefly discuss on community detection strategies and introduce the LP algorithm. Section 4 discusses the tie resolution strategies and the stop criteria that are used in our LP algorithm proposal. In Section 5, we present and analyse our algorithm. In Section 6, we report on and discuss the experimental data. In Section 7, we conclude and discuss some possible extensions of this work.

2 Notation

Let $G = (V, E)$ be an undirected connected network having $n = |V|$ vertices and $m = |E|$ edges. For each $v \in V$, we denote by $N(v) = \{u\colon u \in V, \{u, v\} \in E\}$ the neighbourhood of v and by $\deg(v) = |N(v)|$ the degree of v. Moreover, we denote by $\deg(G) = \max_{v \in V} \deg(v)$ the maximum degree over all the vertices in G.

2.1 Colouring

Given a graph G, the network colouring problem consists in colouring the vertices of G so that no two adjacent vertices are assigned the same colour. The smallest number of colours needed to colour a graph G is called its chromatic number, $\mathcal{X}(G)$. We recall that graph colouring is computationally hard. Especially, it is NP-hard to compute the chromatic number of a graph. However, there are several simple algorithms, even parallelisable (Barenboim and Elkin, 2009), which allow to colour a graph with a number of colours upper bounded by $\deg(G) + 1$.

2.2 Graph partitioning

Denote by $\mathcal{C} = \{C_1, C_2,\ldots,C_k\}$ a partition of the vertices in V. Each C_i is called community. Each community C_i is associated to an induced sub-graph $G(C_i) = (C_i, E(C_i))$ of G, where $E(C_i) = \{\{u, v\}\colon u, v \in C_i, \{u, v\} \in E\}$.

A good community detection strategy should strive to achieve two conflicting goals:

- provide an accurate network partition
- keep low the computational complexity of the algorithm so as the algorithm can be applied to very large networks.

While the latter goal is easily measured by means of a standard worldwide recognised approach (e.g., asymptotic notation), how to evaluate the quality of a given network partition is a recently raised question. In the last few years, much work has been devoted to the problem of defining a metric apt to evaluate the accuracy of a partition algorithm and to compare it with other methods. In the following, we will briefly review the measures we use to validate our algorithm.

2.2.1 Modularity

Graph modularity was proposed by Newman and Girvan (2004). This measure is obtained by summing up, over all communities of a given partition, the difference between the observed fraction of links inside the community and the expected value for a null model, that is, a randomised network having the same size and same degree sequence. Formally, the modularity of a given partition $\mathcal{C}$ of a graph $G = (V, E)$ is defined as:

$$q(\mathcal{C}) = \sum_{C \in \mathcal{C}} \left[\frac{|E(C)|}{m} - \left(\frac{\sum_{v \in C} \deg(v)}{2m} \right)^2 \right], \tag{1}$$

where $E(C)$ is the set of edges connecting two vertices of the community $C \in \mathcal{C}$.

The value of the modularity ranges from –1/2, which corresponds to a bipartite network with canonical clustering, and tends to 1; in particular k cliques, with no extra connections, provide a modularity value equal to $1 - 1 / k$. Positive values indicate the possible presence of community structure. Thus, a good community detection strategy strives for network partitions that provide positive and possibly large values of the modularity.

The effectiveness of modularity, in terms of evaluating the accuracy of a given network partitioning has been questioned: Fortunato and Barthélemy (2007) have shown that modularity often fails to detect clusters smaller than some scale, depending on the size of the network. The origin of the resolution scale lies in the fact that modularity is a sum of terms, where each term relates to a community. Hence, the modularity represents a trade-off between quality and size of communities. The problem is that this trade-off does not always seem to reflect the community structure of the network, especially when the network is large and communities may be very heterogeneous in size.

2.2.2 Accuracy

When the network optimal partitioning is already known, community detection algorithms are often evaluated comparing the similarity of the partitioning, generated by these algorithms, with those known.

The *accuracy* or *rand index* (RI) computes the percentage of nodes membership classified coherently between two partitions (Rand, 1971; Raghavan et al., 2007). Let X and Y be two partitions: For each partition and for all possible pairs of nodes, check whether they belong to the same community. An error occurs when two nodes belonging to the same community according to X (resp., Y) are assigned to different communities by Y (resp., X). Formally,

$$RI(X, Y) = \left(1 - \frac{e}{n(n-1)/2} \right) \times 100, \tag{2}$$

where $n(n - 1) / 2$ is the number of pairs and e is the number of errors found.

2.2.3 Normalised mutual information

Several measures to evaluate community detection algorithms have been borrowed from information theory. In particular, by considering a partition as a probability distribution, the normalised mutual information (NMI) is often used to compare two partitions in order to learn how much they are related. Several variants of the NMI have been defined [see Vinh et al. (2010) for a detailed discussion]. In this paper, we use the 'max' variant which is defined as:

$$NMI_{\max}(X, Y) = \frac{I(X, Y)}{\max\{H(X), H(Y)\}}, \tag{3}$$

where $I(X, Y)$ denotes the *mutual information* and $H(X)$ denotes the Shannon entropy (the information contained in the distribution) of X. $NMI_{\max}$ enjoys several interesting properties: namely it is a *metric* and lies within a fixed range [0, 1]. Specifically, it equals 1 if the partitions are identical, whereas it has an expected value of 0 if the partitions are independent. Several works have used the NMI to test/compare the quality of community detection algorithms (Danon et al., 2005; Lancichinetti and Fortunato, 2009; Lancichinetti et al., 2009).

Both accuracy and NMI are commonly used to evaluate the stability of community detection algorithms. The community detection algorithms presented in this paper are non-deterministic and can therefore give different partitions for the same network. On the other hand, it is important to obtain stable results so as to identify very pertinent communities. Given a set of partitions, obtained using the same algorithm, it is useful to evaluate how much such partitions are related to one another. Indeed the more an algorithm is stable, the more related are its outputs.

3 LP algorithms

The community detection strategy based on a LP algorithm, first studied in Raghavan et al. (2007), identifies network partitions by an 'epidemic' approach, i.e., it uses the diffusion of information in the network to identify communities.

3.1 Main idea

Initially, each vertex in the network is assigned a unique label, which will be used to determine the community it belongs to. Subsequently, an iterative process is performed so that connected groups of vertices are able to reach a consensus on some label giving rise to a community.

At each step of the process, each vertex updates its label to a new one which corresponds to the most frequent label among its neighbours. Formally, for each vertex $v \in V$, v updates it label according to

$$l_v = \arg\max_{l} \sum_{u \in N(v)} [l_u == l] \tag{4}$$

where l_v denotes the label of v and

$$[l_u == l] = \begin{cases} 1 & \text{if } l_u = l \\ 0 & \text{otherwise.} \end{cases}$$

When more than one choice is possible, ties are broken randomly (we will refer to this tie resolution strategy as LPA-R. Different ties management schemes will be considered in the rest of the paper, c.f., Section 4). This process is performed until some stop condition is met, e.g., no vertex changes its label during one step. A network community is then identified as a connected group of vertices having the same final label.

The process is formally described as Algorithm 1: LP algorithm (synchronous). In the following, we denote by $l_v(i)$ the label of vertex v at step i, for $i = 0, 1, \ldots$ and $\forall v \in V$. Initially, all labels are distinct, that is $l_v(0) \neq l_w(0)$ $\forall v, w \in V$.

Algorithm 1 LP algorithm (synchronous)

```
1   **Initialize labels:** for each $v \in V$, $l_v(0) = v$;
2   i = 0;
3   **while** *the stop criterion is not met* **do**
4       i++;
5       **Propagation:**
6       **foreach** $v \in V$ **do**
7           $l_v(i) = \arg\max_l \sum_{u \in N(v)} [l_u(i-1) == l]$,
8       **end**
9   **end**
10  **return** Final labeling: $l_v(t)$ for each $v \in V$, where $t$ is the last executed step.
```

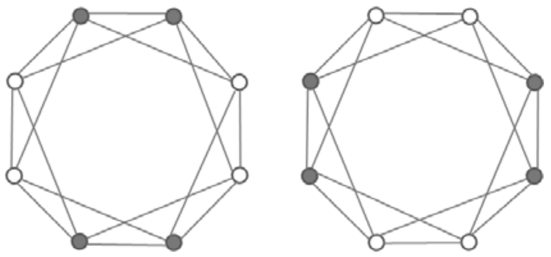
3.2 *Synchronous vs. asynchronous LP algorithm*

The LP algorithm can be either synchronous, as presented above, or asynchronous. In the synchronous model (c.f. Algorithm 1) each vertex computes its label at step i based on the label of its neighbours at step $i - 1$. The synchronous algorithm is easy to implement and embarrassingly parallelisable. Since there are no dependencies between labels belonging to the same step, each *propagation* step can be executed in parallel over the vertices. However, it has been shown that synchronous updating may result in a cyclic oscillation of labels. This problem occurs mainly when the considered network contains a bipartite or a star-like component. We observe that label oscillations are not only due to bipartite or star sub-graphs, several other kinds of graphs, see for instance Figure 1, suffer this oscillation problem. In order to avoid possible cycles and ensure termination, Raghavan et al. (2007) suggest opting for an asynchronous approach (c.f. Algorithm 2). Although, the asynchronous model deeply reduces the oscillation phenomenon, it exhibits some side effects:

- Since each vertex label is updated according to the current label of its neighbours, several dependencies need to be considered if one has to parallelise the algorithm: each vertex cannot compute his own label before each of its neighbour, which precedes it in the chosen permutation, has completed its computation. With more details, while the *foreach* in line 6 of Algorithm 1 is parallelisable, this is not true for the Algorithm 2 (the computation on line 8 depends on the current labels of the considered vertices). Parallelism is very important when the network size is large. Even if the LP algorithm requires a few propagation steps, using the asynchronous approach, each step need to be sequenced among all vertices. Accordingly, the amount of time needed by a parallelised version of the synchronous LP algorithm scales logarithmically (Leung et al., 2009), whereas the time complexity of a parallel asynchronous LP algorithm grows more than linearly, with respect to the network size.

- Moreover, because during each iteration, the updating sequence is randomly chosen, the whole algorithm becomes unstable: different run of the algorithm may provide different final labelling.
- A major limitation of the asynchronous algorithm has been observed in Leung et al. (2009): It often wrongly produces a 'monster' community and several small communities. This problem is related to the fact that, due to the asynchronous nature of the algorithm, during the initial steps, the random permutation of the vertices tends to benefit the spread of some labels with respect to the others. For this reason, certain communities do not form links which are strong enough to prevent a foreign label flooding. Several experiments confirm that the synchronous version of the algorithm slows down the formation of such *monster* communities, even though it does not completely prevent them (Leung et al., 2009).

Figure 1 The oscillation phenomenon on a non-bipartite network



Notes: Labels are represented by colours. Once one of the two configurations is entered, then the label values indefinitely oscillate between them.

Algorithm 2 LP algorithm (asynchronous)

```
1  **Initialize labels:** for each $v \in V$, $l_v(0) = v$;
2  i = 0;
3  **while** *the stop criterion is not met* **do**
4      i++;
5      let $\pi = (v_{\pi(1)}, v_{\pi(2)},\ldots,v_{\pi(n)})$ be a random permutation of the vertices.
6      **Propagation:**
7      **for** $j = 1$ **to** $n$ **do**
8          $l_{v_{\pi(j)}}(i) = \arg\max_{l} \sum_{u \in N(v_{\pi(j)})} [l_u == l]$,
9              where $l_u = \begin{cases} l_u(i) & \text{if } u = v_{\pi(k)}, k < j \\ l_u(i-1) & \text{if } u = v_{\pi(k)}, k > j \end{cases}$
10     **end**
11 **end**
12 **return** Final labeling: $l_v(t)$ for each $v \in V$, where $t$ is the last executed step.
```

3.3 LP algorithm for bipartite networks

A variation of the LP algorithm for bipartite networks was proposed in Liu and Murata (2009). The proposed algorithm has experimentally shown to be as effective as the standard LP algorithm, easily parallelisable, and stable (no label oscillations were observed). In the following we will briefly describe the LP algorithm for bipartite networks. Let B (*blue*) and R (*red*) be the two set of vertices which correspond to the canonical partition of a bipartite network. The algorithm divides each propagation step into two synchronised stages. The first stage computes the new labels for blue vertices according to the current labelling of red vertices. When the labels of all the blue vertices have been updated, the second stage updates the label of red vertices according to the current labelling of blue vertices (c.f. Algorithm 3).

This algorithm corresponds to the asynchronous model where, for each propagation step, the permutation of vertices is such that blue vertices precede red ones. Notice that, being the network bipartite, the label updates for different red (resp. blue) vertices are independent of each other and the two stages are parallelisable.

Algorithm 3 LP algorithm (bipartite networks)

```
1   Initialize labels: for each $v \in V$, $l_v(0) = v$;
2   i = 0;
3   let $B \subset V$ and $R \subset V$ be respectively the set of blue and red vertices.
4   while the stop criterion is not met do
5       i++;
6       Propagation:
7       foreach $v \in B$ do
8           $l_v(i) = \arg\max_l \sum_{u \in N(v)} [l_u(i-1) == l]$,
9       end
10      foreach $v \in R$ do
11          $l_v(i) = \arg\max_l \sum_{u \in N(v)} [l_u(i) == l]$,
12      end
13  end
14  return Final labeling: $l_v$ for each $v \in V$;
```

Source: Liu and Murata (2009)

4 LP algorithms: stopping criteria and tie resolution strategies

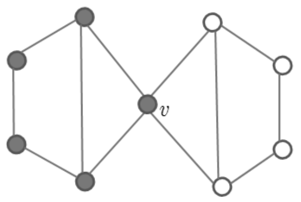
Before proceeding with the description of our proposal, we briefly describe the stop criteria and tie resolution strategies for LP algorithms that will be considered in the following. In the traditional LP algorithm, when a ties occurs [i.e., there are two or more labels that maximise the sum in equation (4)] one of the labels that maximise the sum is chosen randomly. We will call this version of the algorithm *LP algorithm random* (LPA-R).

The simplest stop criterion is the following:

if $l_v(i) = l_v(i-1)$ for each $v \in V$, then the algorithm is stopped. (c0)

That is, comparing the current labelling with the previous one, if no label change occurs, the algorithm is stopped (successive steps would not change any label). Unfortunately, the LP algorithm, even in the asynchronous model, does not prevent cycles (i.e., a sequence of steps such that the final labelling of each node corresponds to the initial one). In such cases the stop criterion (c0) fails. For instance, in the example depicted in Figure 2, the node *v* has two black and two white neighbours (labels are represented by colours). Since both the black and the white label satisfy equation (4) the node v could continuously change his labels and, for this reason, the stopping criteria (c0) may never be satisfied.

Figure 2 An example of tie



Notes: Labels are represented by colours. Vertex *v* has exactly two black and two white neighbours.

For this reason, the following more elaborated stop criterion (c1) was proposed in Raghavan et al. (2007):

if in the current labelling for each vertex $v \in V$ the current label l_v satisfies equation (4), that is, l_v is a solution of $\arg\max_l \sum_{u \in N(v)} [l_u == l]$, then the algorithm is stopped. (c1)

Said in other words, if in the next step all the label changes, if any, would come from ties, then the procedure ends and the current labelling determines a network partition. For instance the labelling presented in Figure 2 satisfy the stop criteria (c1) (for each node, its label satisfies equation (4)]. Alternatively, in order to use a simple version of the stop criterion a change in the management of ties is required as proposed in Barber and Clark (2009). If, in case of ties, the current label of the vertex has priority over the others, the probability of generating cycles is sensibly reduced (Barber and Clark, 2009). We will call this version of the algorithm *LP algorithm with precedence* (LPA-Prec). Formally, this variation is defined as follows: during the propagation step, if the current label satisfies equation (4), then the vertex keeps its current label, otherwise the algorithm follows the standard rules. For instance, using LPA-Prec, the labelling depicted in Figure 2 cannot change (in particular, since the current label of *v* satisfies the equation (4), *v* keeps the black label). Hence, using LPA-Prec the labelling in Figure 2 satisfies both the stop criteria (c0) and (c1).

LPA-Prec is typically more stable than the LP algorithm (a smaller amount of randomisation is injected into the algorithm). Moreover, in some cases the LPA-Prec stops whereas LPA-R keeps running in search of better solutions. For this reason, in addition to be more stable, the LPA-Prec usually exhibits a faster convergence than LPA-R, however the quality of the network partition generated can be a bit worse.

Another tie resolution strategy guarantees the convergence of the algorithm. Assume that the set of labels allows defining a priority between each pair of labels. For instance, we can assume that labels are integer, and that a label l has priority over a label l' if $l > l'$. In this version of the algorithm, named LPA-Max, each tie is solved deterministically by taking the label with higher priority between the set of labels that get the maximum in equation (4).

Using results in Poljak and Sura (1983), one knows that:

- *Fact 1:* The synchronous version of LPA-Max does not generate any cycle of size larger than two.

Fact 1 implies that one can adopt a simplification of the stop criterion (c1):

$$\text{if either } l_v(i) = l_v(i-1) \text{ for each } v \in V \text{ or } l_v(i) = l_v(i-2) \text{ for each } v \in V, \text{ then the algorithm is stopped.} \tag{c2}$$

We stress that, synchronous LPA-Max is completely deterministic; it will always generate the same network partition whenever it starts with the same initial vertex labels. Anyway, by randomising the initial labelling, or by using an asynchronous approach, it is possible to obtain different results.

Notice that when the number of initial labels is 3 or more, the two variants of the LP algorithm can also be applied together, we will call this tie resolution strategy LPA-Prec-Max.

5 Semi-synchronous LP

In this section, we present the main contribution of this paper. Our proposal combines the advantages of both the synchronous and asynchronous model discussed above. Namely, our system is stable and efficient (easy parallelisable) as the synchronous model while does not undergo the oscillation problem. We present a semi-synchronous LP algorithm which allows to overcome the oscillation problem in any network. We will also formally prove that our algorithm avoids oscillations, that is, it converges to a stable labelling.[1]

Our work is inspired by the LP algorithm for bipartite networks given in Liu and Murata (2009). We stress that, in general, the formal study of LP algorithms convergence is an open problem. In particular, no formal proof (or informal hint) that shows why the proposal in Liu and Murata (2009) is able to defeat the oscillation issue has been provided.

We propose an algorithm which consists of two phases:

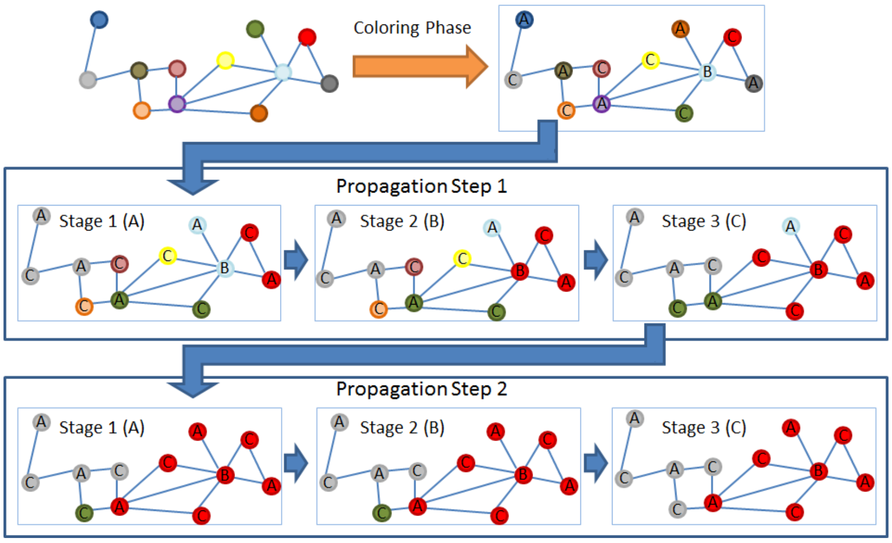
1 *Colouring phase:* Colour the network vertices so that no two adjacent vertices share the same colour (i.e., by any distributed graph colouring algorithm). The colouring phase is easily parallelisable and efficient [only $O(\deg(G)]$ synchronous parallel steps) (Barenboim and Elkin, 2009).

2 *Propagation phase:* Each LP step is divided into stages. Each stage is named upon a different colour. At stage c, labels are simultaneously propagated to the vertices that have been assigned colour c during the colouring phase.

Algorithm 4 LP algorithm (semi-synchronous)

```
1   Initialize labels: for each $v \in V$, $l_v(0) = v$;
2   Network coloring: assign a color to the vertices of the network so that no two adjacent
    vertices share the same color. Let $\ell$ be the number of used colors. Denote by $D_j$ the set of
    nodes which have got color $j$, for $j = 1,\ldots,\ell$.
3   i = 0;
4   while the stop criterion is not met do
5       i++;
6       Propagation:
7       for $j = 1$ to $\ell$ do
8           foreach $v \in D_j$ do
9               $l_v(i) = \arg\max_{l} \sum_{u \in N(v)} [l_v == l]$,
10                  where $l_u = \begin{cases} l_u(i) & \text{if } u \in D_k, k < j \\ l_u(i-1) & \text{if } u \in D_k, k > j \end{cases}$
11          end
12      end
13  end
14  return Final labeling: $l_v(t)$ for each $v \in V$, where $t$ is the last executed step.
```

Figure 3 An example of semi-synchronous LP algorithm



Notes: Initially, each vertex in the network is assigned a unique label (labels are represented by colours). Then the set of vertices is partitioned using a colouring algorithm (labels inside vertices describe this partition). Thereafter, each propagation step proceeds colour by colour. In this particular case, after two propagation steps the network is partitioned into two communities (grey and red vertices).

A formal description of the algorithm is given as Algorithm 4: LP algorithm (semi-synchronous) and depicted schematically in Figure 3. It is easy to verify that the number of stages per propagation step corresponds to the number of colour needed to colour the network. Moreover, each stage of the propagation phase is easily parallelisable: since there are no dependencies between vertices having the same colour, each stage can be executed synchronously. Hence, the amount of time needed to reach a final consensus grows like the number of propagation steps times the number of stages per propagation step; we recall that the former value has been experimentally shown to grow logarithmically with respect to n (Leung et al., 2009), while the latter is bounded by $\deg(G) + 1$ (Barenboim and Elkin, 2009).

Theorem 5.1: Consider a network $G = (V, E)$. Algorithm 4, with LPA-Prec, LPA-Max and LPA-Prec-Max tie resolution strategies, does not generate any cycle (i.e., it converges to period 1).

Proof: Consider a network $G = (V, E)$ having n nodes and m edges. Let[2]

$$f(i) = \sum_{\{u,v\}\in E} \left[l_u(i) == l_v(i) \right] \times n + \sum_{v\in N} l_v(i)$$

be a function that to each step $i \geq 1$ associates n times the number of monochromatic edges, plus the sum of all node labels. We will show that $f(i)$ grows monotonically with the step number i. Clearly, the function $f(i)$ is upper bounded by $mn + nn = n(m + n)$.

Consider any step i. If $l_v(i) = l_v(i-1)$ for each $v \in V$ then the algorithm has reached its final state and stops. Otherwise, there is at least one vertex v that has changed his label, that is,

$$l_v(i) \neq l_v(i-1). \tag{5}$$

Since the labelling of adjacent vertices occurs asynchronously, we get that whenever v is relabelled, v's neighbours do not change their labels.

We distinguish now two cases depending on whether $l_v(i-1)$ satisfies equation (4), that is $l_v(i-1)$ is a solution of $\arg\max_l \sum_{u\in N(v)} [l_u == l]$ (c.f. line 9 of Algorithm 4).

Case 1 $l_v(i-1)$ *satisfies equation (4).* We know that $l_v(i)$ and $l_v(i-1)$ are both solutions of $\arg\max_l \sum_{u\in N(v)} [l_u == l]$. Hence, $l_v(i)$ and $l_v(i-1)$ occur the same number of times among the current labels of the neighbours of v. We notice that this case can occur only when using the LPA-Max tie resolution strategy, otherwise v would keep its current label [that is, $l_v(i) = l_v(i-1)$ would hold contradicting (5)]. The LPA-Max strategy implies that v has changed his label by choosing a new one such that $l_v(i) > l_v(i-1)$.

Hence, we get that the number of monochromatic edges incident on v does not change (recall that no neighbour of v updates its label during the same step). The contribute of v to the sum of all the node labels in function f, increases by $l_v(i) - l_v(i-1) \geq 1$.

Case 2 $l_v(i-1)$ *does not satisfy equation (4).* Now, $l_v(i)$ appears more times than $l_v(i-1)$, among the current labels of the neighbours of v. Therefore, the number of monochromatic edges incident on v increases at least by 1 (since no neighbour of v updates its label concurrently); this implies an increase of n to the value of the function f. Moreover, $l_v(i) - l_v(i - 1) \geq 1 - n = -(n - 1)$. Hence, we get that the contribute of v to the function f increases at least by $n - (n - 1) = 1$ going from step $i - i$ to step i.

The above reasoning applies to each other vertex in G that changes its label during the same step. Hence, $f(i) > f(i - 1)$ and the result follows. □

Theorem 5.1 implies that semi-synchronous LPA-Prec, LPA-Max and LPA-Prec-Max can be implemented using the trivial stop criteria (c0).

Corollary 5.1: Algorithm 4 with stop criterion (c0) converges with LPA-Prec, LPAMax and LPA-Prec-Max tie resolution strategies.

In order to ensure that even the classical LPA-R algorithm converges, it is necessary to use a more elaborate stop criterion.

Theorem 5.2: Consider a network $G = (V, E)$. Assume that Algorithm 4 uses the stop criterion (c1), that is, it ends at the first step t such that for each $v \in V$ one of the following condition holds

1 $l_v(t) = l_v(t - 1)$

2 $l_v(t) \neq l_v(\mathrm{i} - 1)$ but this change is due to a tie.

Then Algorithm 4 converges, independently of the tie management rule.

Proof: By using the stop criterion (c1) we have that as long as the stop criterion is not met, at least one vertex v updates his label without the occurrence of a tie. Algorithm 4 assures that when a vertex is relabelled, its neighbours do not change their labels. This implies that the number of monochromatic edges strictly increases during step i. □

6 Experimental results

LP algorithms are particularly difficult to analyse from a formal point of view (Brandes et al., 2007). In this respect, in fact, very few results have been obtained. For this reason, the only way to analyse and compare LP algorithms is by an empirical approach. Testing an LP algorithm means to execute the algorithm on a network, for which we know the community structure, and verify the accuracy of the obtained results, comparing also with the results obtained by other algorithms.

We have implemented both the asynchronous and the semi-synchronous LP algorithms with four tie resolution strategies: LPA-R, LPA-Prec, LPA-Max and LPA-Prec-Max. We compared the quality, the efficiency and the stability of such algorithms on a set of real networks.

6.1 Analysed network and their properties

The choice of networks to be used as a benchmark is a crucial problem. Several experiments have been executed on computer generated networks with a community structure known by construction (Girvan and Newman, 2002; Pang et al., 2009). However, generated networks cannot model real networks. Several studies have been developed in order to devise a class of benchmark graphs, having a known community structure and able to resemble real networks (Lancichinetti et al., 2008). Our tests have been performed over real networks having a known community structure (Newman, 2006b; Leskovec, 2011). Table 1 reports some known properties of such networks:

1 *Zachary's karate:* social network of friendships between 34 members of a karate club at a US university in the 1970s (Zachary, 1977)

2 *Dolphins:* social network of frequent associations between 62 dolphins in a community living off Doubtful Sound, New Zealand (Lusseau et al., 2003)

3 *Football:* network of American football games between Division IA colleges during regular season Fall 2000 (Girvan and Newman, 2002) (115 teams)

4 *NetScience:* co-authorship network of 1,589 scientists working on network theory and experiment, as compiled by Newman (2006a) in May 2006

5 *Power:* a network representing the topology of the Western States Power Grid of the USA (Watts and Strogatz, 1998) (4,941 vertices)

6 *Hep-TH:* Arxiv HEP-TH (high energy physics theory) citation graph (Gehrke et al., 2003; Leskovec, 2011) (9,877 authors)

7 *Internet:* a symmetrised snapshot of the structure of the internet at the level of autonomous systems, reconstructed from BGP tables posted by the University of Oregon Route Views (Newman, 2006b) (22,963 nodes)

8 *Cond-Mat:* co-authorships between 31,163 scientists posting preprints between January 1, 1995 and June 30, 2003 on the Condensed Matter E-Print Archive (Newman, 2001)

9 *Enron e-mail:* Enron e-mail communication network covers all the e-mail communication within a dataset of around half million e-mails (Leskovec et al., 2005; Leskovec, 2011) (36,692 e-mail addresses).

Table 1 Networks' properties

	# of vertices	*# of edges*	*Modularity (known real partitioning)*	*Maximum modularity*
Karate	34	78	0.358	0.431
Dolphins	62	159	0.378	0.527
Football	115	613	0.553	0.588
NetScience	1,589	2,742	0.955	0.917
Power	4,941	6,594	–	0.807
HepTh	9,877	25,973	–	0.699
Internet	22,963	48,436	–	0.516
Cond-Mat	31,163	120,029	0.688	0.657
Enron e-mail	36,692	367,662	–	0.583

6.2 Test settings

We have performed our tests by considering 72 different test settings. Each test setting is characterised by the choice of the LP timing (asynchronous or semi-synchronous), the network and the tie resolution strategy.

Since LP algorithms involve a certain amount of randomisation, we execute each test 100 times and use the means and the variance of their 'performances' for our comparisons.

Each test setting is composed as follows:

1 We randomly assign the initial network labelling and, whenever is needed, execute a greedy graph colouring algorithm which considers the vertices in increasing order of their labels. Notice that different initial assignment may correspond to different network colouring.

2 We execute the LP algorithm using the stop criterion (c1) proposed in Raghavan et al. (2007). During each test we collect the number of phases required to reach the final labelling (*efficiency*).

3 Then we use a simple propagation algorithm which identifies the communities as connected group of nodes having the same final label. Notice that disconnected groups of nodes having the same label are considered as different communities.

4 We measure and compare the effectiveness (both quality and stability) of the partition obtained according to different metrics (c.f. Section 2.2).

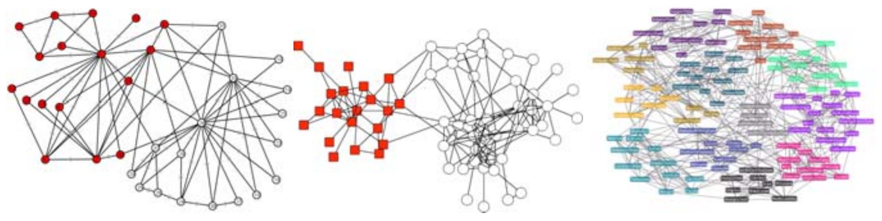
In the following each test setting is identified by a triple (P, N, T) where

- $P \in$ {*Asynchronous*, *Semi-synchronous*} indicates the LP timing
- $N \in$ {*Karate*, *Dolphins*, *Football*, *NetScience*, *Power*, *HepTH*, *Internet*, *Cond-Mat*, *E-mail*} indicates the network
- $T \in$ {*LPA-R*, *LPA-Prec*, *LPA-Max*, *LPA-Prec-Max*} indicates the tie resolution strategy.

6.3 Preliminary validation result: networks for which we know the real community partitioning

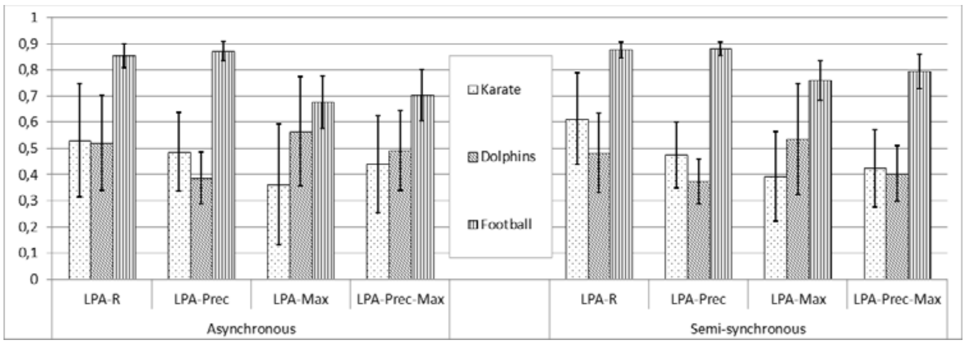
In order to validate the effectiveness of the proposed algorithms we conducted a preliminary pilot test. As we know the communities present in *Zachary*, *Dolphins* and *Football* networks (c.f. Figure 4), we applied both algorithms (asynchronous and synchronous) to these networks and we then measured the similarity of the results comparing, using both *NMI* and *Accuracy* (see Section 2.2), each solution with the real partitioning. Figure 5 depicts the average *NMI* [Figure 5(a)] and *Accuracy* [Figure 5(b)] observed for each test setting. The results are encouraging and show that both semi-synchronous and asynchronous algorithms perform quite well (the average accuracy is around 0.8). A detailed analysis of the charts shows that the semi-synchronous approach performs slightly better than the asynchronous. The results also show that the Football network appears to be easier to partition. This is evident not only by the fact that the average *NMI* and *Accuracy* are higher but also because the range of these values (represented by error bars in the figure) is very small.

Figure 4 Three network, with their known real partitioning, are depicted, (a) karate (b) dolphins (c) football

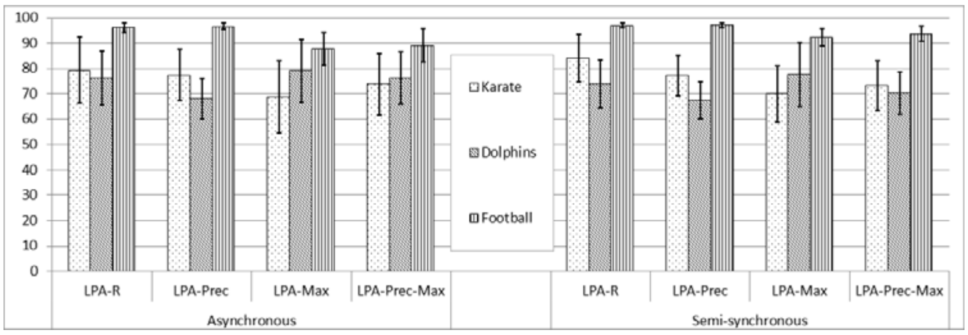


(a) (b) (c)

Figure 5 Average and range of (a) *NMI*$_{\max}$ and (b) *Accuracy* measured comparing the partitions obtained on each test setting with the real partitioning



(a)



(b)

Note: Error bars denote the range obtained.

6.4 Massive tests: large networks

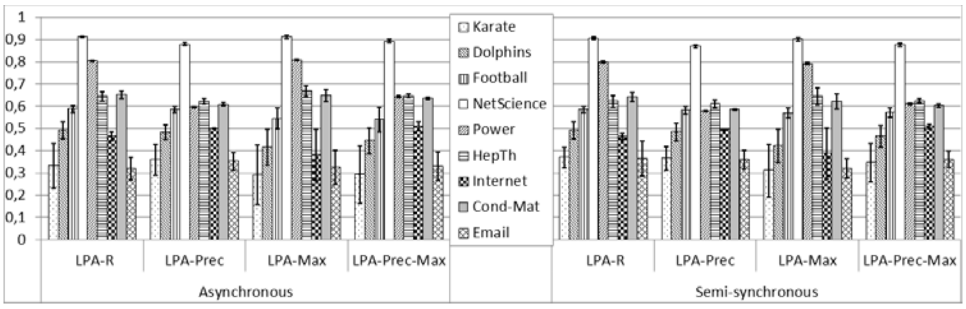
After the preliminary test we developed a complete test that includes all the networks described in Section 6.1 with the aim of assessing the quality, efficiency and stability of the proposed algorithm.

6.4.1 Quality

Figure 6 depicts the average modularity observed for each test setting. The results are quite similar, and reproduce the known values presented in Table 1. In particular, the semi-synchronous approach slightly outperforms the asynchronous one on *Karate*, *Dolphins*, *Football* and *E-mail* networks while it is slightly worse on *NetScience*, *Power*, *HepTh* and *Cond-Mat* networks. On the *Internet* network performances are almost identical.

The quality of the results seems to be independent of the strategy used for the management of ties: except for the *Power* network, where strategies LPA-R and LPA-Max provide a much better modularity (≈ 0.8) compared with LPA-Prec and LPA-Prec-Max (≈ 0.6) and the *Internet* network, where, on the other hand, LPA-Max works a little bit worse, the other results are comparable.

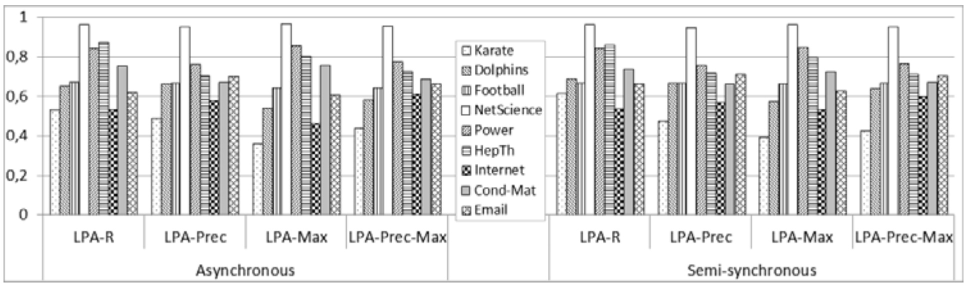
Figure 6 Average modularity and standard deviation measured on each test setting



Notes: Each test has been executed 100 times. Error bars denote the standard deviation obtained.

As for the preliminary test, we have made, for each test setting, a comparative analysis between the results obtained from different runs of the algorithm and the best result obtained. For each test setting the results arising from 100 executions are compared, using NMI, with the best solution, according to the *modularity*. Figure 7 presents the results of this comparison, for each test setting the average NMI is depicted. The results show that both the approaches provide consistent results. In particular, semi-synchronous algorithms appear more often near the optimal solution which indicates a greater stability of such algorithms.

Figure 7 Average $NMI_{\max}$ measured comparing the partitions obtained on each test setting with the overall best partition, according to the *modularity*

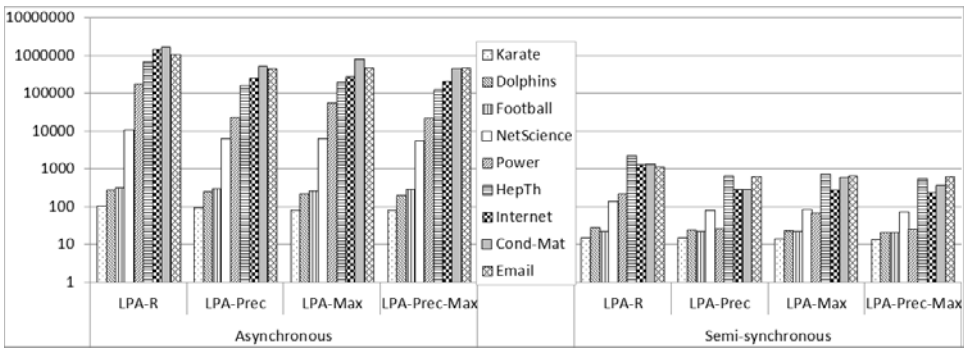


We stress that our goal here is not to improve the quality of the results obtained by the asynchronous approach; rather we want to show that the speed-up obtained with the semi-synchronous approach does not degrade any quality.

6.4.2 Efficiency

The efficiency of our proposal is shown in Figure 8. The improvement grows with the network size. In our tests it ranges from ≈ 5.7: (asynchronous, *Karate*, LPA-Max) required 80 stages, on average, while (semi-synchronous, *Karate*, LPA-Max) only 14 stages, to ≈ 1,802: (asynchronous, *Cond-Mat*, LPA-Prec) required 528,772 stages, on average, while (semi-synchronous, *Cond-Mat*, LPA-Prec) required 290 stages. This means that our algorithm would greatly benefit from parallel implementations. Indeed, assuming that we are able to use a parallel system with an unbounded number of CPUs, the wall clock time will be proportional to the number of stages required.

Figure 8 Average number of stages required by each test setting



Note: The *y*-axis, which denotes the number of stages, is on a logarithmic scale.

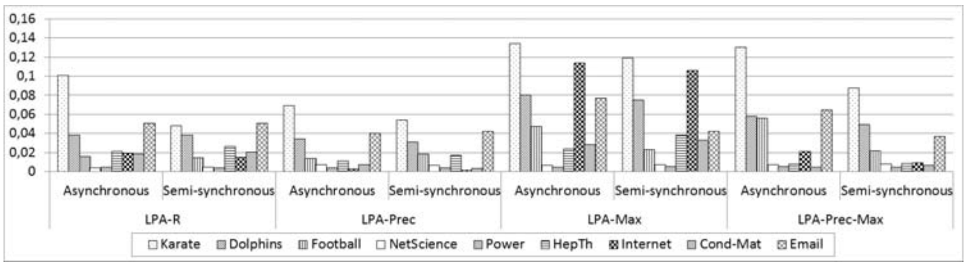
Timing results also confirm that the number of steps needed to reach a final labelling grows logarithmically with respect to the size of the network and is independent from the model used (asynchronous or semi-synchronous).

A careful comparison of these results also shows that all the variants of the standard LPA-R (that is, LPA-Max, LPA-Prec and LPA-Prec-Max) converge faster to the final labelling. As an example, the number of propagation steps required by (*, *Power*, LPAR) is 33.15, on average, while (*, *Power*, LPA-Max) is only 10.54. Notice that, in Figure 8 results are shown with logarithmic scale in order to represent both the results of the asynchronous and semi-synchronous approaches.

6.4.3 Stability

We report, in Figures 6 and 9, the standard deviation (σ) of the modularity obtained for each test setting. The results indicate that the semi-synchronous approach is more stable than the asynchronous one.

Figure 9 Standard deviation of the modularity measured on each test setting

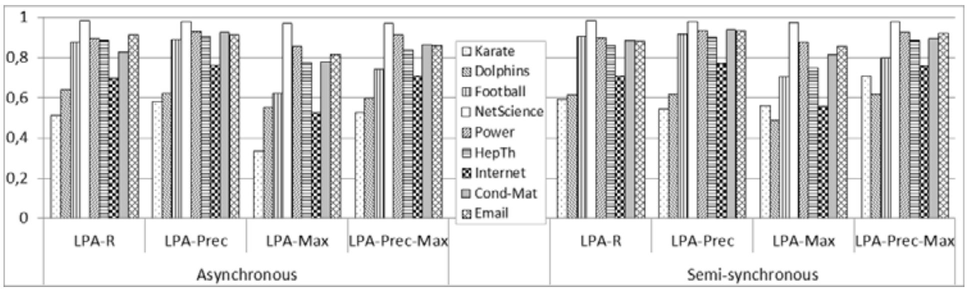


Note: Each test has been executed 100 times.

Furthermore, by an accurate analysis of the results, it is possible to infer that whatever the model used (semi-synchronous or asynchronous) the stability of results is strongly influenced by the strategy used for the management of ties. For this reason, we show these results in four different charts, one for each LP algorithm variant (c.f. Figure 9). In particular, the LPA-Prec is more stable than the others, while the LPAMax is very unstable, especially in *Karate*, *Dolphins*, *Internet* and *E-mail* networks. The instability of LPA-Max is mainly due to the fact that before each test the initial labelling is randomised and consequently the way in which ties are solved changes too. On the other hand, LPA-Prec solves each tie in favour of the node own label irrespective of the value of the involved labels.

Another test was implemented in order to understand the stability of the algorithms. For each test setting, the similarity between different solutions, obtained by running ten times the same test, has been computed. Each solution has been compared, using NMI, with all the other (overall 45 comparisons for each test setting). The average NMI measured is given in Figure 10. The results confirm that the semi-synchronous approach is more stable than the asynchronous one.

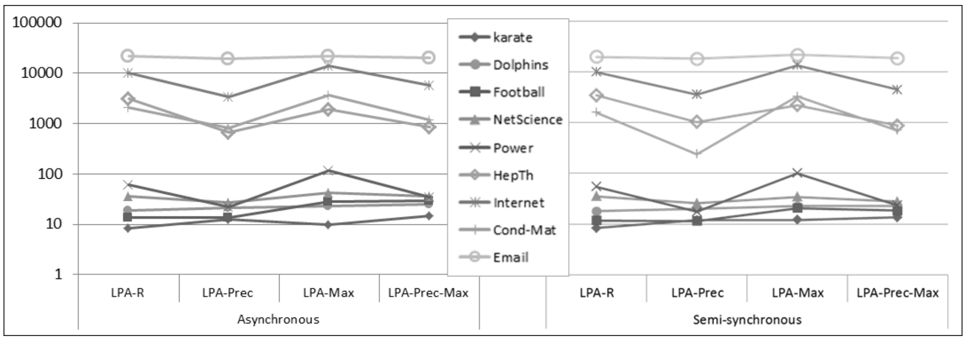
Figure 10 Similarity between the partitions



Notes: For each test setting the stability of solutions is measured by comparing, using the NMI, all pairs of ten solutions (overall 45 comparisons). The figure depicts the average $NMI_{\max}$ obtained for each test setting.

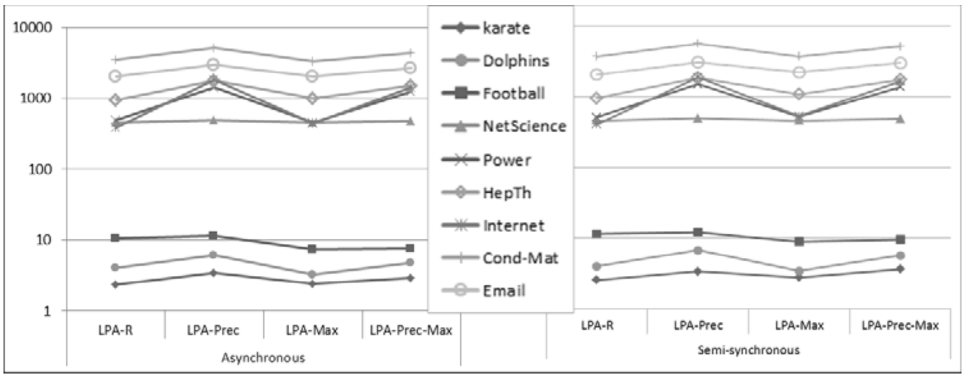
We have also collected information about the size and the number of the communities generated during each test (c.f. Figures 11 and 12). The rationale behind this analysis is to understand whether the semi-synchronous algorithm avoids the generation of a 'monster' community (c.f. Section 3.2). The results confirm that the semi-synchronous LP algorithm slightly reduces this phenomenon: the average size of the largest community is always smaller while the average number of community is larger. Results also show that several networks (e.g., *Internet*) suffer this problem much more than others (e.g., *NetScience*).

Figure 11 Average size of the largest community



Note: The y-axis, which denotes the size of the largest community, is on a logarithmic scale.

Figure 12 Average number of communities



Note: The y-axis, which denotes the number of communities, is on a logarithmic scale.

To conclude we also note that even in this experiment, the stability of the algorithm is influenced by the strategy used to manage ties. All tests have shown that LPA-Prec is more stable than LPA-R and LPA-Prec-Max is more stable than LPA-Max.

7 Conclusions

We have presented a novel semi-synchronised LP algorithm for detecting network communities. Our proposal has shown to be

- *effective*: the accuracy of our approach is comparable with standard LP algorithms
- *efficient*: it is easy to parallelise and consequently would greatly benefit, in terms of convergence timing, of a parallel implementation

- *stable*: our proposal has the advantage of 'limiting' randomisation to such an extent that results are quite uniform.

The set of developed tests also gives us the chance to perform a detailed comparison of four different LP algorithm tie resolution strategies. Results show that each strategy has some peculiarity: LPA-Max is faster than the other approaches, LPA-Prec is more stable and LPA-Prec-Max seems to be a good trade-off. Results show also that the peculiarity of LP algorithm variants are not influenced by the approach (asynchronous or semi-synchronous) used during the LP step.

We showed that using a quite simple stop criterion, our algorithm is able to defeat the oscillation phenomena that preclude the use of synchronous strategies. In particular, we have showed that semi-synchronous LPA-Prec, LPA-Max and LPA-Prec-Max converge to a period 1.

Several problems still remain open. In particular, we notice that, while the experiments reported in (Leung et al., 2009) empirically show that the expected number of iterations grows logarithmically with respect to the size of the network, the problem of characterising the convergence of synchronous LP algorithms and the problem of determining its running time are still open (in both the synchronous and asynchronous models). For instance, it has been shown that synchronous LPA-Max converges to a period 2 (Poljak and Sura, 1983). What about the LPA-Prec and the LPA-Prec-Max variants? Which is analytically the convergence running time of LP algorithms in their different variants?

References

Albert, R. and Barabási, A-L. (2002) 'Statistical mechanics of complex networks', *Rev. Mod. Phys.*, January, Vol. 74, No. 1, pp.47–97.

Barber, M.J. and Clark, J.W. (2009) 'Detecting network communities by propagating labels under constraints', *Phys. Rev. E*, Vol. 80, No. 2, pp.1–16.

Barenboim, L. and Elkin, M. (2009) 'Distributed ($\delta + 1$)-coloring in linear (in δ) time', *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*, pp.111–120.

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z. and Wagner, D. (2007) 'On finding graph clusterings with maximum modularity', *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science*, in A. Brandstädt, D. Kratsch, and H. Müller (Eds.): *Lecture Notes in Computer Science*, Vol. 4769, pp.121–132, Springer, Berlin.

Chen, W., Liu, Z., Sun, X. and Wang, Y. (2010) 'A game-theoretic framework to identify overlapping communities in social networks', *Data Min. Knowl. Discov.*, September, Vol. 21, No. 1, pp.224–240.

Comellas, F. and Miralles, A. (2010) 'A fast and efficient algorithm to identify clusters in networks', *Applied Mathematics and Computation*, Vol. 217, No. 5, pp.2007–2014.

Cordasco, G. and Gargano, L. (2010) 'Community detection via semi-synchronous label propagation algorithms', *Proceedings of the IEEE International Workshop on Business Applications of Social Network Analysis (BASNA 2010)*.

Danon, L., Duaz-Guilera, A., Duch, J. and Arenas, A. (2005) 'Comparing community structure identification', *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 2005, No. 9, pp.1–10.

Easley, D. and Kleinberg, J. (2010) *Networks, Crowds, and Markets Reasoning About a Highly Connected World*, 31 July 2010, Cambridge University Press, Cambridge UK, Hardcover, 800pp.

Fortunato, S. and Barthélemy, M. (2007) 'Resolution limit in community detection', *Proceedings of the National Academy of Sciences*, January, Vol. 104, No. 1, pp.36–41.

Gehrke, J., Ginsparg, P. and Kleinberg, J.M. (2003) 'Overview of the 2003 KDD cup', *SIGKDD Explorations*, Vol. 5, No. 2, pp.149–151.

Girvan, M. and Newman, M.E.J. (2002) 'Community structure in social and biological networks', *Proc. Natl. Acad. Sci.*, USA, Vol. 99, No. 12, pp.7821–7826.

Guimera, R., Sales-Pardo, M. and Amaral, L.A.N. (2004) 'Modularity from fluctuations in random graphs and complex networks', *Phys. Rev. E*, Vol. 70, No. 2, pp.1–4.

Lancichinetti, A. and Fortunato, S. (2009) 'Community detection algorithms: a comparative analysis', *Phys. Rev. E*, November, Vol. 80, No. 5, pp.1–11.

Lancichinetti, A., Fortunato, S. and Kertész, J. (2009) 'Detecting the overlapping and hierarchical community structure in complex networks', *New Journal of Physics*, Vol. 11, No. 3, pp.1–18.

Lancichinetti, A., Fortunato, S. and Radicchi, F. (2008) 'Benchmark graphs for testing community detection algorithms', *Physical Review E*, Vol. 78, 046110, pp.1–5.

Leskovec, J. (2011) *Stanford Large Network Dataset Collection*, available at http://snap.stanford.edu/data/ (accessed on July 2011).

Leskovec, J., Kleinberg, J. and Faloutsos, C. (2005) 'Graphs over time: densification laws, shrinking diameters and possible explanations', *International Conference on Knowledge Discovery and Data Mining (KDD '05)*.

Leung, I.X.Y., Hui, P., Liò, P. and Crowcroft, J. (2009) 'Towards real-time community detection in large networks', *Phys. Rev. E*, June, Vol. 79, No. 6, pp.1–10.

Liu, X. and Murata, T. (2009) 'How does label propagation algorithm work in bipartite networks?', *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '09)*, pp.5–8.

Lusseau, D., Schneider, K., Boisseau, O.J., Haase, P., Slooten, E. and Dawson, S.M. (2003) 'The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations: can geographic isolation explain this unique trait?', *Behavioral Ecology and Sociobiology*, Vol. 54, No. 4, pp.396–405.

Narayanam, R. and Narahari, Y. (2010) *A Game Theoretic Approach to Graph Partitioning with Application to Community Detection in Social Networks*, under review.

Newman, M.E.J. (2001) 'The structure of scientific collaboration networks', *Proc. Natl. Acad. Sci.*, USA, Vol. 98, pp.404–409.

Newman, M.E.J. (2004) 'Fast algorithm for detecting community structure in networks', *Phys. Rev. E*, Vol. 69, No. 6, 066133, pp.1–5.

Newman, M.E.J. (2006a) 'Finding community structure in networks using the eigenvectors of matrices', *Phys. Rev. E*, September, Vol. 74, 036104, No. 3, pp.1–22.

Newman, M.E.J. (2006b) *Network Data*, available at http://www.personal.umich.edu/~mejn/netdata/ (accessed on July 2011).

Newman, M.E.J. and Girvan, M. (2004) 'Finding and evaluating community structure in networks', *Phys. Rev. E*, February, Vol. 69, 026113, No. 2, pp.1–16.

Orman, G.K. and Labatut, V. (2009) 'A comparison of community detection algorithms on artificial networks', *Proceedings of the 12th International Conference on Discovery Science (DS'09)*, Porto, Portugal, *LNCS*, Vol. 5808, pp.242–256, Springer, Berlin/Heidelberg.

Pang, C., Shao, F., Sun, R. and Li, S. (2009) 'Detecting community structure in networks by propagating labels of nodes', *Proceedings of the 6th International Symposium on Neural Networks: Advances in Neural Networks – Part III (ISNN 2009)*, *LNCS*, Vol. 5553, pp.839–846, Springer, Berlin/Heidelberg.

Poljak, S. and Sura, M. (1983) 'On periodical behaviour in societies with symmetric influences', *Combinatorica*, Vol. 3, No. 1, pp.119–121.

Raghavan, U.N., Albert, R. and Kumara, S. (2007) 'Near linear time algorithm to detect community structures in large-scale networks', *Phys. Rev. E*, September, Vol. 76, 036106, No. 3, pp.1–12.

Rand, W.M. (1971) 'Objective criteria for the evaluation of clustering methods', *Journal of the American Statistical Association*, Vol. 66, No. 336, pp.846–850.

Rosvall, M. and Bergstrom, C.T. (2007) 'An information-theoretic framework for resolving community structure in complex networks', *Proceedings of the National Academy of Sciences*, USA, Vol. 104, pp.7327–7331.

Vinh, N.X., Epps, J. and Bailey, J. (2010) 'Information theoretic measures for clusterings comparison: variants, properties, normalization and correction for chance', *Journal of Machine Learning Research*, October, Vol. 11, No. 10, pp.2837–2854.

Watts, D.J. and Strogatz, S.H. (1998) 'Collective dynamics of 'small-world' networks', *Nature*, Vol. 393, No. 6684, pp.440–442.

Wu, F. and Huberman, B.A. (2004) 'Finding communities in linear time: a physics approach', *The European Physical Journal B – Condensed Matter and Complex Systems*, Vol. 38, No. 2, pp.331–338.

Zachary, W.W. (1977) 'An information flow model for conflict and fission in small groups', *Journal of Anthropological Research*, Vol. 33, No. 4, pp.452–473.

Notes

1 Depending on the propagation rule used, some cycles can still occur due to the management of ties.

2 Notice that we are assuming that the initial node labels are 1,…,n, however the proof immediately holds for any initial labelling of the nodes by simply substituting n by $\max_v l_v(0)$.