scientific reports

Check for updates

OPEN

Large network community detection by fast label propagation

Vincent A. Traag[1]✉ & Lovro Šubelj[2]

Many networks exhibit some community structure. There exists a wide variety of approaches to detect communities in networks, each offering different interpretations and associated algorithms. For large networks, there is the additional requirement of speed. In this context, the so-called label propagation algorithm (LPA) was proposed, which runs in near-linear time. In partitions uncovered by LPA, each node is ensured to have most links to its assigned community. We here propose a fast variant of LPA (FLPA) that is based on processing a queue of nodes whose neighbourhood recently changed. We test FLPA exhaustively on benchmark networks and empirical networks, finding that it can run up to 700 times faster than LPA. In partitions found by FLPA, we prove that each node is again guaranteed to have most links to its assigned community. Our results show that FLPA is generally preferable to LPA.

Networks are relevant in various scientific fields, ranging from social networks in sociology to metabolical networks in biology. There are various techniques to try to improve our understanding of networks. One such technique is to cluster the nodes of a network, such that nodes within a cluster are relatively densely connected while they are relatively sparsely connected between clusters. There is a wide variety of clustering approaches to networks, such as modularity[1] and stochastic block models[2] and approaches based on dynamical processes on networks[3], such as random walks[4]. Sometimes, a similar approach can be solved in various ways. For example, when modularity was first proposed, it used a cutting approach based on betweenness[1]. New algorithms were continuously proposed, either improving the speed of the algorithm or the quality of the partition. This includes a fast greedy approach[5], a slower simulated annealing approach[6], a faster algorithm based on extremal optimisation[7], a fast hierarchical multi-level method, known as the Louvain algorithm[8] which was most recently improved upon in the Leiden algorithm[9]. Most of these algorithm can also optimise different quality functions, such as the Constant Potts Model[10].

A technique that takes a heuristic approach is the label propagation algorithm (LPA)[11]. Its foremost focus is on speed, trying to find clusters in as little time as possible. Simply put, LPA works by iteratively updating the label of each node to a label that is most common among its neighbours. We here propose a fast variant of LPA (FLPA), which can potentially run up to hundreds of times faster than LPA. This allows to cluster even larger networks in even less time. We consider this to be useful as a first initial look at a network, although other methods are arguably more robust and preferable[12]. The results of LPA are only local minima of a global quality function for which the optimum is simply placing all nodes into a single community[13]. Other quality functions may be more informative of any structure in the network.

We first briefly review LPA and introduce the fast variant FLPA in the next section. We then briefly analyse the performance of LPA and FLPA theoretically, followed by experimental analyses on both synthetic and empirical networks.

Label propagation algorithm

We now introduce the label propagation algorithm (LPA)[11] more formally. For a more detailed review of label propagation algorithms, we refer the reader to literature reviews[12,14].

Let $G = (V, E)$ be an undirected multigraph with nodes V and edges E, where there are $n = |V|$ nodes and $m = |E|$ edges. Let A be the adjacency matrix of graph G, such that A_{ij} is the number of edges between i and j, with $A_{ij} = 0$ if and only if nodes i and j are not connected (i.e. $(i, j) \notin E$). The implementation of LPA is quite straightforward. Let c_i be the label of node $i \in V$. Typically, each node is initially labelled differently, i.e. $c_i = i$ for all $i \in V$. At each step, we take a random $i \in V$ and change its label to the majority in its neighbourhood. In more detail, we do the following. For a specific node i, we count how many neighbours have label c as $n_c = \sum_{j \in V} A_{ij} \delta(c_j, c)$, where $\delta(c_j, c)$ is the Kronecker delta function such that $\delta(c_j, c) = 1$ if $c_j = c$ and 0 otherwise. We then consider the set of most frequent labels $\{c\} = \text{argmax}_c\, n_c$. We randomly sample uniformly from the set of most frequent labels $\{c\}$ a label c^* and update the label $c_i = c^*$. We repeat these steps over all nodes in V.

[1]Centre for Science and Technology Studies, Leiden University, Leiden, The Netherlands. [2]Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia. ✉email: v.a.traag@cwts.leidenuniv.nl

After having looped over all nodes, we check whether all labels are among most the frequent, whether the label is *maximal*. If any label is not maximal, we perform another iteration over all nodes, until all labels are maximal. This algorithm is summarised in Algorithm 1.

This version of the label propagation is also referred to as the asynchronous implementation of label propagation[11]. The synchronous implementation of label propagation showed potential problems with (near) bipartite networks and some other networks, resulting in oscillations of labels. We therefore do not consider the synchronous implementation, and limit the discussion to the asynchronous implementation.

After LPA terminates, it is guaranteed that the label c_i of each node $i \in V$ is maximal. That is, $n_{c_i} = \max_c \sum_{j \in V} A_{ij}\delta(c_j, c)$. This is trivial to prove, since LPA continues to process nodes until all labels are maximal. In the original introduction of LPA[11], they observed that this is close to the definition of "communities in the strong sense" as introduced by Ref. [15].

The overall time complexity of a single iteration over all nodes in LPA is $O(m)$. The number of iterations necessary before convergence is not known theoretically. It was observed that generally only a few iterations suffice to have most labels consistent with their final labelling[11]. However, there are no clear results for the overall runtime complexity of LPA.

Retention strategy. LPA stops iterating whenever the labels of all nodes are maximal. The original implementation of LPA[11] simply considered always updating the label with a randomly selected maximal label. Later, a so-called retention strategy was suggested: update the label only if it is not maximal[16]. This has one great benefit: we can simply keep track of whether a label was updated during the iteration, and if so, we continue iterating over all nodes. This means it is not necessary to check for maximal labels after an iteration over all nodes, making the implementation more efficient. The retention strategy is summarised in Algorithm 2.

In addition, this retention strategy introduces more stability, because it does not continuously sample from competing maximal labels for a single node like the original LPA does. LPA sometimes shows the appearance of a "giant" cluster[11,17] which might be related to the quality function which it implicitly optimises[13]. Indeed, merging neighbouring clusters does not alter the label maximality, so that any arbitrary combination of clusters in principle will still meet the original stopping criteria. The retention strategy might prevent the method from finding such "giant" clusters[16].

Some empirical observations in the literature noted that we can expect roughly $1.03m^{0.23}$ iterations for the retention strategy on empirical networks[18]. This leads to an overall time complexity of about $O(m^{1.23})$ for the retention strategy.

Fast label propagation algorithm. We now introduce the fast label propagation algorithm (FLPA). It is based on the same principle that is used for the fast local move in the Leiden algorithm[9]. Similar to LPA, each node $i \in V$ has an associated label c_i, and we use a similar majority update rule. However, instead of checking after an iteration whether all labels are maximal, or by considering whether labels are updated, as done in the retention strategy, we maintain an explicit queue of nodes Q that should be considered. If c_i of node i is changed, we append some of its neighbours $N_i = \{j \mid (i,j) \in E\}$ to the queue. In particular, we add each neighbour $j \in N_i$ to the queue that has a label different from the *new* label of i, $c_j \neq c_i$ and does not yet belong to the queue $j \notin Q$. At each step, we pop the node from the beginning of the queue, and we continue to process all nodes until the queue is empty. Hence, instead of iterating over *all* nodes if a label is changed, we only consider nodes in whose neighbourhood a label changed. This greatly reduces the number of nodes that we consider, making the algorithm even faster. The algorithm is summarised in Algorithm 3.

We now prove that FLPA provides the same guarantee as LPA, namely that after FLPA terminates, it is guaranteed that the label c_i of each node $i \in V$ is maximal. We first observe that a node i that is not in the queue will have its label c_i as its maximal label. That is, $n_{c_i} = \max_c \sum_{j \in V} A_{ij}\delta(c_j, c)$. This clearly holds when node i was processed. If node i is currently not part of the queue, we can discern two cases. In the first case, no labels in its neighbourhood have changed at all. In this case, the label c_i continues to be the maximal label. In the second case, the label of a neighbour $j \in N_i$ changed. It must then hold that $c_j = c_i$, since otherwise i should have been added to the queue. In that case, the number of labels in the neighbourhood of i that equal c_i increased. If n_{c_i} was maximal prior to node j changing its label, it continues to be maximal. Hence, c_i continues to be the maximal label as long as node i is not part of the queue. If the queue is empty, the label c_i of each node i is guaranteed to be the maximal label, similar to LPA.

Earlier literature also suggested a speedup of LPA in a similar fashion[19,20]. However, they seem to have taken a slightly more complicated approach, either introducing additional heuristics or requiring the algorithm to check whether a neighbour might be updated or not. Our approach is easier to implement, and seems to result in even greater speedups.

Results

To understand better the differences between LPA, its retention alternative, and FLPA, we analysed three theoretical graphs: a complete graph, a star graph and a cycle graph. In addition, we also analysed the differences in results in practice. We ran benchmarks on five different types of synthetic networks and on twelve different empirical networks. In addition to comparing LPA, its retention alternative, and FLPA, we also compare with the Leiden algorithm, which is one of the fastest available algorithms[9]. We use the Leiden algorithm to optimise for modularity. We compare both the speed and the resulting partitions.

We first discuss our theoretical results. We then present the results for the synthetic networks[21]. Following that, we discuss the results for the empirical networks.

Theoretical analysis. LPA performs a number of iterations over n nodes. Each potential update of a label for a node i has a complexity of $\Theta(k_i)$, where k_i is the degree of node i, and therefore the total complexity of a single iteration is linear in the number of edges $\Theta(m)$. If the number of iterations does not scale with n, we simply have linear complexity $\Theta(m)$. Presumably, however, the number of iterations will increase with n, but it is not clear exactly how. The complexity also relates to the resulting partitions, since finding a partition of $n/2$ equally weighted clusters will most likely take less time than finding a partition consisting of a single cluster. The same reasoning applies to the retention variant and FLPA.

For some specific graphs we can analyse the complexity in more detail. We analyse the algorithms for three theoretical graphs: a complete graph, a star graph and a cycle graph. We summarise the runtime complexities in Table 1.

Complete graph. Let us start by analysing a complete graph with n nodes and $m = \binom{n}{2}$ edges. All three algorithms, LPA, its retention variant, and FLPA, will find a partition consisting of a single cluster within a single iteration of all nodes. Suppose on the contrary that a partition consists of $k > 1$ clusters, while the label of each node is maximal. We prove by contradiction that this is not possible. With k clusters, there are k labels, and each label c occurs n_c times. Now suppose that $n_c \geq n_d$ for a pair $c \neq d$. Then each node with label d has only $n_d - 1$ neighbours with label d, and since $n_c > n_d - 1$ its own label d is not maximal. Therefore $n_c < n_d$ for all c and d, which is impossible, and there can be only one cluster.

Let us now consider the complexity. Initially, each algorithm starts with a singleton partition such that $c_i = i$. Let us analyse the first node that is considered in each algorithm. Without loss of generality, we can label this node 1. Each node has $n - 1$ neighbours, and initially each node is in its own cluster, meaning there are $n - 1$ unique labels for node 1, each of which occurs only once. Since the current label $c_1 = 1$ of node 1 is not yet maximal, a random label will be chosen by each algorithm, say label i, and we set $c_1 = i$. Then, when we consider node 2 (assuming $i \neq 2$), there are $n - 2$ unique labels, of which $n - 3$ occur once and one label (namely label i) occurs twice. Thus, there is only a single maximal label i, and node 2 will switch to label i. Subsequently, all remaining nodes will also switch to label i, and hence all nodes will be assigned label i in the end.

All three algorithms consider n nodes in this case. However, both LPA and its retention variant will consider again n nodes for updating, or checking for maximality. FLPA does not need to do this, and the queue will be empty after considering all n nodes. This means that the total runtime in this case is $\Theta(2m)$ for LPA and retention, and $\Theta(m)$ for FLPA.

There is an exception, namely if $i = 2$, which occurs with probability $\frac{1}{n-1}$. Let us consider what happens to node 2. If $i = 2$, its current label $c_2 = 2$ is maximal, since it has $n - 1$ unique labels, each of which occurs only once, but now this includes its own label 2. The retention strategy will not draw a random label, since its current label is already among the maximal labels of its neighbours. Therefore, for retention, all remaining nodes will simply switch to label 2 and we end up with a complexity of $\Theta(2m)$. However, in LPA and FLPA, a random label is drawn from all maximal labels. With probability $\frac{1}{n-1}$, label 2 is drawn. In this case, all other remaining labels also change to label 2 and all labels are maximal after a single iteration. With probability $1 - \frac{1}{n-1}$, label $j \neq 2$ is drawn. In this case, we need to perform another round in LPA since $c_1 = 2$ is not maximal, and in FLPA node 1 is added to the queue again.

The probability of needing a second iteration over all nodes for LPA is then

$$\frac{1}{n-1}\left(1 - \frac{1}{n-1}\right),$$

which goes to 0 for $n \to \infty$. This covers only the first iteration, and there will be similar probabilities involved in each subsequent iteration, but clearly those probabilities will equally go to 0. Therefore, the expected runtime for LPA is close to $\Omega(2m)$, although slightly higher.

If nodes continuously select the label of the node that will be considered next, as in $c_1 = c_2$, there will continue to be a single label that occurs twice, while all other labels occur once. In FLPA, if a node chooses the label of the node that will be considered next, i.e. $c_i = c_{i+1}$ for $i > 1$, we add the node $i - 1$ that was previously considered to the queue. For each node, this happens with probability $\frac{1}{n-1}$, which leads to an expected number of additional nodes of

Graph	LPA	retention	FLPA
	$\Omega(2m)$	$\Theta(2m)$	$\Theta\left(m + \frac{1}{n-2}\right)$
	$\Theta\left(2m + \frac{2n(n-2)}{(n-1)^2}\right)$	$\Theta(2m)$	$\Theta\left(m + \frac{n-2}{(n-1)^2}\right)$

Table 1. Expected runtime complexity of LPA, its retention variant and FLPA for two theoretical graphs.

$$\frac{1}{n-1}\sum_{k=1}^{\infty}\left(\frac{1}{n-1}\right)^{k}=\frac{1}{(n-1)(n-2)}.$$

Since the degree of each node is equal to $n-1$, the total expected runtime for FLPA is $\Theta\left(m+\frac{1}{n-2}\right)$.

Star graph. We now analyse a star graph of n nodes, with a single node in the center and $n-1$ leaf nodes connected to the central node, with in total $m=n-1$ edges. Similar to the complete graph, all three algorithms will find a partition consisting of a single cluster. Clearly, the leaf nodes cannot have a different label as the central node, since that is their only neighbour. So, by definition, all leaf nodes must have the same label as the central node, and hence all nodes have the same label.

Regardless of the updating order, all leaf nodes will always adopt the label from the central node in each algorithm. The only question is what happens for the central node. If the central node is selected first, it will simply choose a random label from the leaves, and all leaves will adopt that label. If the central node is selected third or later, it will not change its label because its own label is then by definition the only maximal label. Thus, in both these cases, only n nodes are considered, and both LPA and retention perform another pass over all nodes to check for maximality (in the case of LPA) or because there was a change (in retention), resulting in a total runtime of $\Theta(2m)$. In FLPA, no nodes are ever added to the initial queue of n nodes, resulting in a runtime of $\Theta(m)$.

Let us examine what happens when the central node is selected as the second node to update. This only happens with probability $\frac{1}{n-1}$ and is therefore unlikely to have much effect on the overall expected runtime. For the retention strategy, if the central node is considered second, this means that the first node has already adopted the label of the central node and the retention strategy will not update its label. Thus, retention is not affected by this and maintains a runtime of $\Theta(2m)$.

In LPA, if the central node is considered second, it will choose the same label as its current label with probability $\frac{1}{n-1}$. In this case, after all the remaining nodes have also updated their label, all nodes have identical labels and LPA terminates. If a different label is selected, all other $n-2$ leaf nodes also adopt that label. In a subsequent iteration, no node except the first leaf node initially considered will update its label. Therefore, LPA will converge in at most two iterations, with the second iteration occurring with probability $\frac{1}{n-1}\frac{n-2}{n-1}$, resulting in a total expected runtime of

$$\Theta\left(2m\left[1+\frac{n-2}{(n-1)^2}\right]\right).$$

Finally, in FLPA, if the central node is considered second and it chooses a label different from its current label, the first leaf node initially considered will be added to the queue, and nothing more. Hence, the total expected runtime is

$$\Theta\left(m+\frac{n-2}{(n-1)^2}\right).$$

Cycle graph. Let us now consider a cycle graph of n nodes. In contrast to the complete graph and the star graph, each algorithm now results in different partitions. In all three algorithms, each node whose neighbours have not yet been updated, simply chooses the label of a random neighbour. The distribution of cluster sizes is difficult to analyse exactly. Regardless, all labels are maximal only if all clusters are larger than a single node. This is clear for the nodes in the interior of a cluster, since their label is identical to that of their two neighbours. For the nodes at the border of a cluster, their current label is among the maximal labels. At the borders of clusters is where differences between the individual algorithms emerge.

For the retention strategy, no updates will be considered anymore. Therefore, it will quickly settle on any partition that does not include a cluster of size one.

For LPA the picture is a bit different. At each border of two clusters, the node updates its label by randomly choosing between the label on its left and the label on its right. Thus, as long as there is still a cluster of size one in the partition, all borders will continue to change in LPA, unlike for the retention strategy. Although clusters of size one may disappear, they may also newly appear when clusters of larger size shrink.

For FLPA the picture is again slightly different. Whereas LPA simply continues to move around all nodes, and thus moves around all borders as long as there is a cluster of size one, FLPA has more local dynamics. Suppose that FLPA updates the label of a node at the border of a cluster. If this happens, its neighbour in the old cluster is added to the queue (unlike its neighbour in the new cluster). So with probability $\frac{1}{2}$ a neighbouring node will be updated, and this happens again with probability $\frac{1}{2}$ et cetera. Continuing this reasoning, the expected number of moves resulting from a single border is

$$\sum_{k=0}^{\infty}\left(\frac{1}{2}\right)^{k}=2.$$

Finally, a node in a cluster of size one is never maximal, so FLPA will continue to run until there are no more such clusters. The difference, however, is that only nodes in such clusters can be moved, rather than always considering all nodes at the borders like LPA does.

In summary, the retention strategy will most likely detect the most fine-grained clusters, LPA the least fine-grained clusters, and FLPA somewhere in between. Experimental simulations suggest that retention settles on

clusters with 2.72 nodes and FLPA on clusters with 4.11 nodes, whereas the cluster size increases with the number of nodes n for LPA. In terms of the runtime, retention is then expected to converge the fastest, LPA the slowest, and FLPA somewhere in between. Nonetheless, retention will still consider updating all n nodes, even when moving only a single node in a cluster of size one. So, in total, the runtime of FLPA might still be lower than the runtime of the retention strategy.

Experimental results. We now present some experimental results based on implementations of LPA, its retention variant, FLPA, and the Leiden algorithm for optimising modularity. For all test results, we report averages and standard deviations of 2 000 runs unless explicitly stated otherwise. All results were run on Dell PowerEdge M620 computing nodes with Intel E5-2697 CPUs.

Synthetic networks. As is clear, for all networks FLPA is always faster than LPA (Fig. 1). For the largest networks with 100 000 nodes, FLPA is somewhere between 3–10 times faster than LPA. On Erdős-Rényi (ER) graphs, LPA is only about twice as slow as FLPA, which is comparable to the runtime of the Leiden algorithm. On two-dimensional geometric graphs, LPA is even 3–4 times slower than the Leiden algorithm, which in turn is still about 3–4 times as slow as FLPA. We tested the algorithms on a stochastic block model (SBM) of 100 groups and a mixing parameter of $\mu = 0.6$ with average degree of $\langle k \rangle = 10$, such that each node has about 4 links within its own group and 6 scattered across the other groups. For this case, LPA is again about twice as slow as FLPA, which is again about 1.5 times faster than the Leiden algorithm.

The comparison between FLPA and the retention variant of LPA is more complex. For some networks, the retention strategy alternative is faster, while for other networks FLPA is faster. However, in cases when the retention strategy is faster, it typically fails to perform well. For instance, in ER graphs, both LPA and FLPA find a single large cluster (Fig. 2 left), as expected based on earlier literature[11], but the retention strategy simply terminates almost immediately, finding a partition that closely resembles the singleton partition of each node in its own cluster. In SBM graphs, both LPA and FLPA typically find partitions that are close to the planted partition, while the retention strategy finds many small clusters within each group of the planted partition (Fig. 2 right). This is understandable, since every group is essentially an ER graph internally. In short, the retention strategy is faster simply because it stops very soon after having found some very small community structure.

The overall scaling of both LPA and FLPA seems to be near linear, although there might be some super-linear factors. For all algorithms, we experimentally estimate runtime complexities of the form $am^b + c$ using the Levenberg-Marquardt algorithm, where m is the number of edges and the coefficient b is of central interest. For ER graphs, LPA scales as $O(m^{1.58})$ and FLPA as $O(m^{1.49})$, while the retention strategy scales as $O(m^{1.15})$. As we already noticed, in ER graphs, retention finds very small clusters, which explains its lower complexity. On forest fire graphs and geometric graphs, FLPA is much faster than LPA and the retention strategy. Indeed, we find a scaling of $O(m^{1.27})$ for forest fire graphs and $O(m^{1.21})$ for geometric graphs for FLPA, while LPA shows a scaling of $O(m^{1.67})$ and $O(m^{1.57})$ respectively, with retention showing a scaling of $O(m^{1.47})$ and $O(m^{1.37})$ respectively. Finally, on SBM graphs, the performance of LPA and FLPA is similar to the performance on ER graphs, leading to a complexity of $O(m^{1.53})$ for LPA and $O(m^{1.64})$ for FLPA. For the retention strategy, the scaling cannot be estimated unambiguously, since it finds completely different partitions depending on the size of the graph. That is, for larger graphs, it tends to find more fine-grained structure within each cluster, causing it to converge relatively faster than for smaller graphs.

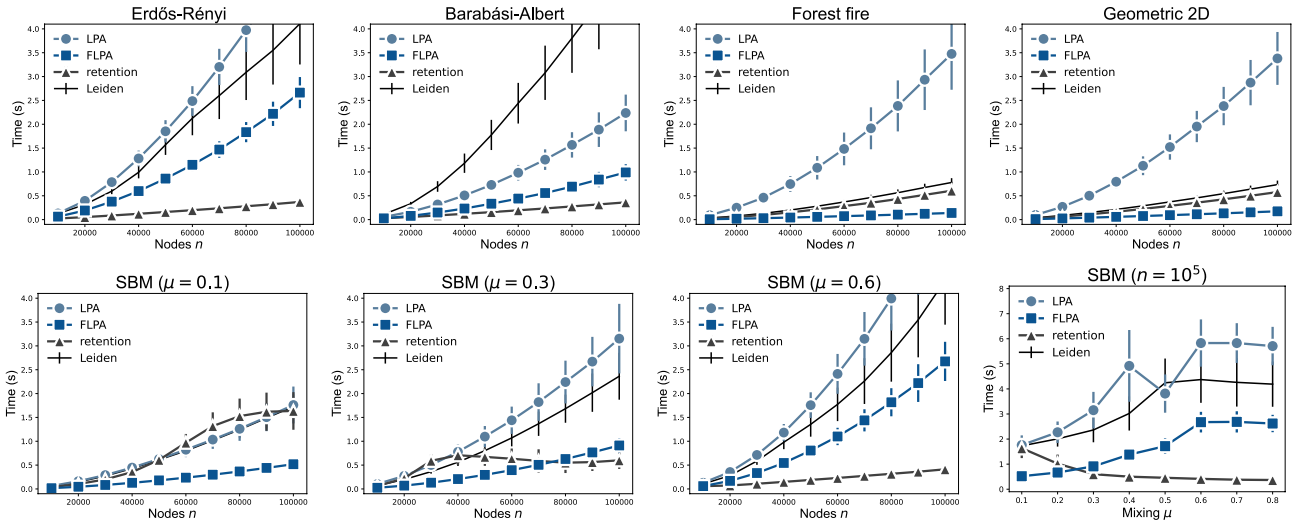

Figure 1. Algorithm runtime for synthetic networks with increasing number of nodes n and the average degree $\langle k \rangle = 10$. (top) Erdős-Rényi random graphs, Barabási-Albert scale-free graphs, forest fire graphs with burning probability 0.5 and two-dimensional geometric graphs with connection radius $\sqrt{\langle k \rangle / (\pi n)}$. (bottom) Stochastic block model graphs with 100 groups and mixing parameter μ.

The runtime complexity also depends on how challenging it is to find a partition. To investigate this more closely, we consider the runtime for each algorithm while varying the average degree $\langle k \rangle$ (Fig. 3). LPA, FLPA and retention all find small clusters in ER graphs with low average degree. If the degree is sufficiently large, both LPA and FLPA will find a single large cluster (Fig. 3 left), as also suggested by our theoretical analysis of a complete graph. The larger the degree, the faster both LPA and FLPA converge towards this large cluster (Fig. 3 middle). Ultimately, when $\langle k \rangle = n - 1$, the graph is a complete graph, for which our theoretical runtimes indicate that FLPA is about twice as fast as LPA. In contrast, the retention strategy does not show a convergence to a single large cluster for $\langle k \rangle < 25$.

The overall figure is very similar for SBM graphs (Fig. 3 right). Initially, all algorithms struggle to uncover the planted partition. When the degree is sufficiently high, the planted partition becomes more easily recognisable, and the algorithms converge faster.

For SBM graphs, we also compare the detected partitions with the planted partition of 100 groups (Fig. 4) using the normalised mutual information (NMI)[22]. When we increase the mixing parameter μ (Fig. 4 left), we find that both LPA and FLPA are able to detect the correct partition up to a threshold. For $\mu = 0$ all edges are within groups, while for $\mu = 1$ all edges are between groups, and so increasing μ makes it more difficult to correctly detect the planted partition. For $\mu = \frac{n-n_c}{n-1}$ the SBM is identical to an ER graph (where n_c is the size of the communities), although it already becomes essentially indistinguishable before this threshold due to stochastic fluctuations[23]. Up to $\mu = 0.3$ both LPA and FLPA detect the planted partition perfectly, for $\mu = 0.4$ the performance starts to degrade, and for $\mu = 0.5$ the algorithms no longer find the planted partition at all. This closely resembles the results of the Leiden algorithm, which we use to optimise modularity. The retention strategy is never able to detect the planted partition correctly. This is mostly because it finds more fine-grained structure within each planted cluster. If we increase the average degree $\langle k \rangle$ (Fig. 4 middle), the retention strategy is also able to find the planted partition. However, the retention strategy requires a far larger degree $\langle k \rangle > 15$ to do so than LPA, FLPA and the Leiden algorithm.

We also compare the differences across partitions that are detected by the different algorithms (Fig. 5 top) based on the normalised variation of information (VI)[24]. The VI of an algorithm with itself denotes the average VI of two runs of the same algorithm on the exact same graph. We can interpret this self-VI as a measure of stability: larger VI values suggest that partitions differ quite a bit from run to run, indicating a lower stability, while lower VI values suggest that partitions are quite similar from run to run, indicating a higher stability. For ER graphs, there is essentially no variation within and between LPA and FLPA, because they always find a single large cluster. The retention strategy, as explained earlier, typically finds some structure within ER graphs, which can also differ quite a bit from run to run. The Leiden algorithm also shows quite different results from run to run, and also finds some structure within ER graphs, which is a known result for modularity[25]. For both the forest fire graphs and the two-dimensional geometric graphs, the three variants of LPA find relatively comparable structures, which differs from the partitions from the Leiden algorithm. The stability of the partitions in the two-dimensional geometric graph of the three LPA variants is similar to the stability of the Leiden algorithm. Finally,

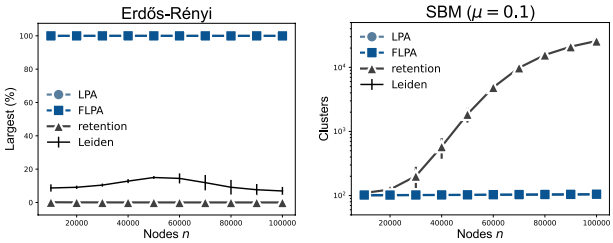

Figure 2. (left) Largest cluster size for Erdőős-Rényi random graphs and (right) the number of clusters for stochastic block model graphs with increasing number of nodes n and the average degree $\langle k \rangle = 10$. Note that the results for LPA are not well visible because they overlap with the results of FLPA.

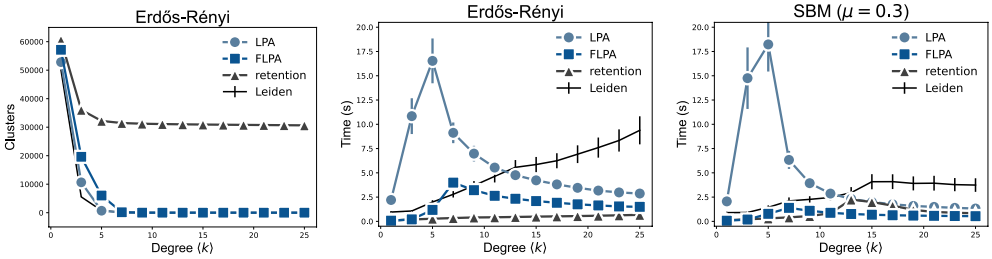

Figure 3. Algorithm runtime for (middle) Erdős-Rényi random graphs and (right) stochastic block model graphs with $n = 10^5$ nodes and varying average degree $\langle k \rangle$.

on SBM graphs, the retention strategy seems to be unable to find any meaningful partition. As explained earlier, each group in the SBM is similar to an ER graph, where the retention strategy also finds very small substructures, which hence also shows up in the SBM results. In contrast, both LPA and FLPA are able to find relatively stable partitions that are similar to each other, and similar to the results of the Leiden algorithm, as they all detect the planted partition of the SBM.

Empirical networks. We now discuss the results of the empirical networks. We first tested the algorithms on seven large empirical networks. They vary in size, ranging from 317 080 nodes and 1 049 866 edges for the smallest network (`com-dblp`), up to 6 297 539 nodes and 16 057 711 edges (`bitcoin`) or 4 847 571 nodes and 68 993 773 edges (`livejournal`). FLPA is between 30–700 times faster than LPA (Table 2) and between 4–15 times faster than the retention strategy. For the largest network (`bitcoin`), LPA takes over 45 min on average, while FLPA is finished in 37 s. LPA is by far the slowest on the `us-patents` network, where it takes over 7 h, while FLPA is finished in 38 s. This may be partly due to LPA finding a much coarser partition: the largest cluster covers more than 38% of the nodes, and it finds about over 6 times fewer clusters than the retention alternative, and over 3 times fewer clusters than FLPA. This conforms to the general pattern that LPA finds the least clusters, FLPA finds more clusters, while the retention strategy finds even more clusters. There is no a priori reason to prefer larger over smaller clusters, so we cannot say whether a method would be preferable over the other based on this observation.

The partitions of most empirical networks themselves are similar between the three variants of LPA (Fig. 5 bottom). Moreover, they differ similarly from the results of the Leiden algorithm. The Leiden algorithm shows greater VI than (F)LPA for the `livejournal` and `twitter-sample` networks. This might be related to the community sizes that Leiden finds, since we use it to optimise modularity, which suffers from a resolution limit[26]. The resolution limit might lead the method to aggregate several clusters together, which may be somewhat arbitrary. Indeed, the Leiden algorithm using modularity typically finds an order of magnitude fewer clusters than LPA. As already noted earlier, for the `us-patents` network there is a larger difference between LPA and the retention variant and FLPA.

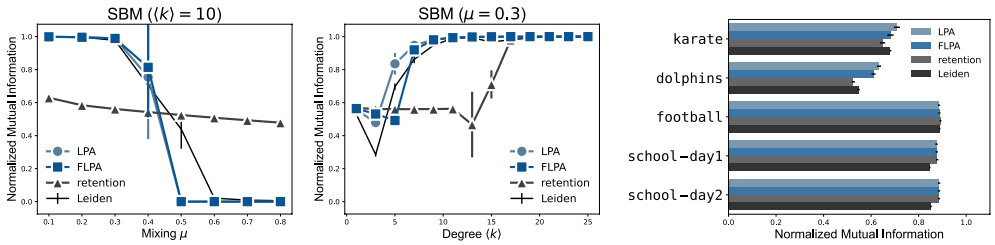

Figure 4. Accuracy of partitions for stochastic block model graphs with $n = 10^5$ nodes and (left) varying mixing parameter μ, where higher μ corresponds to partitions that are more challenging to detect, or (middle) varying average degree $\langle k \rangle$, and (right) small social networks with the metadata on node clusters.

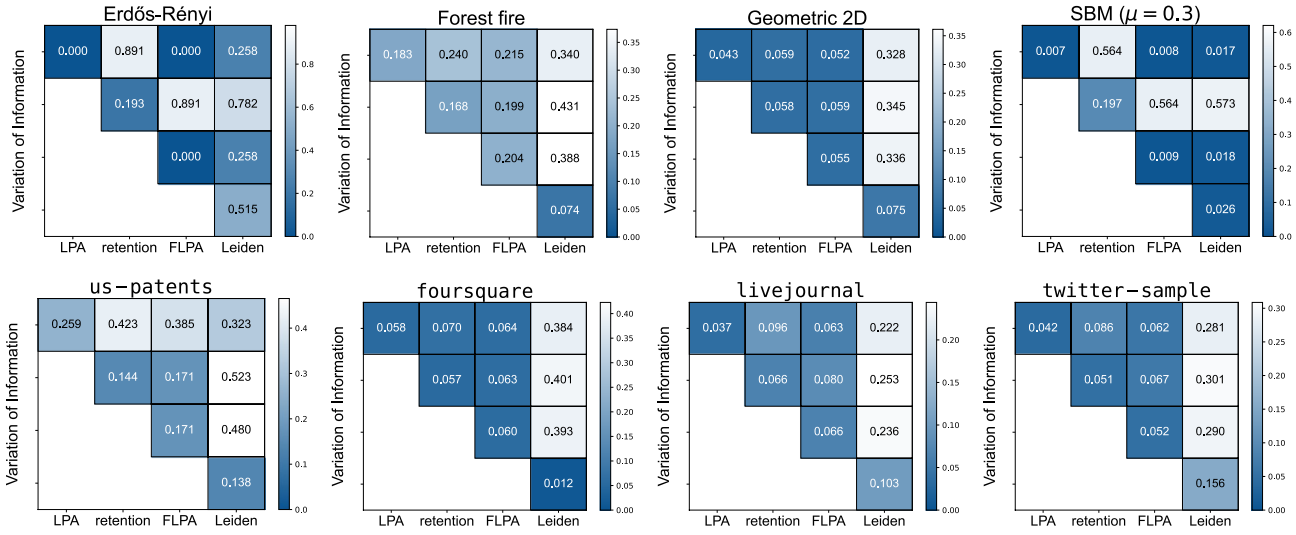

Figure 5. Distance between partitions of (top) synthetic networks with $n = 10^5$ nodes and (bottom) empirical networks.

We also tested the algorithms on five small social networks with a known sociological division of nodes into communities (Fig. 4 right). We compare the detected partitions with the node clusters described in the Methods. We do not present the timing results for these networks, since they are too small for the measurements to be informative. Overall, the results indicate that LPA finds the partitions with the highest NMI for these networks, followed by FLPA and then the retention strategy. The Leiden algorithm, which optimises modularity, finds the worst partitions as measured by the NMI. However, the results do not differ much between the algorithms.

Discussion

Detecting communities in networks is a frequent task in network analysis. Label propagation is one of the fastest algorithms available. It may be useful to get an initial first look at a network. We here suggested a faster variant of label propagation. Fast label propagation can run up to 700 times faster than the original label propagation. Additionally, the same guarantees continue to hold, and results from both algorithms are largely comparable. The quality of the partitions that FLPA finds seem to be on par with LPA. When using label propagation, we believe our fast variant will bring benefits at no additional costs. Although label propagation may be useful to get a first look at a network, other methods may be likely to provide more accurate results. One possibility is to use FLPA to obtain an initial rough partition, which is then further improved by the Leiden algorithm, aiming for a specific objective function. Label propagation was also used to effectively and efficiently compress the Facebook graph[27] to calculate its four degrees of separation[28]. At the same time, it is quite similar to majority opinion simulations [29]. The suggested speedup might also be relevant in the context of such applications.

Methods

Algorithm implementation. We have implemented FLPA in C in `igraph` and made it available in its Python interface `python-igraph`. The C source code can be found in https://github.com/vtraag/python-igraph/tree/flpa, while the Python interface can be found in https://github.com/vtraag/igraph/tree/flpa. We compared FLPA to the existing implementation of LPA in (`python-`)`igraph`.

Network	Nodes	Edges	Algorithm	Clusters	Largest	Time (s)	Speedup
com-dblp	317 080	1 049 866	LPA	22 048	20.1%	185.0 ± 105.0	188.5×
			retention	46 380	2.1%	10.6 ± 6.0	10.8×
			FLPA	32 329	2.7%	1.0 ± 0.4	
roadnet-ca	1 965 206	5 533 214	LPA	219 392	0.0%	940.4 ± 341.8	161.8×
			retention	610 087	0.0%	23.2 ± 8.6	4.0×
			FLPA	341 773	0.0%	5.8 ± 1.9	
us-patents	3 774 768	16 522 438	LPA	53 337	38.3%	26 704.4 ± 12 100.4	705.2×
			retention	359 233	1.4%	601.8 ± 273.4	15.9×
			FLPA	183 476	1.8%	37.9 ± 12.3	
foursquare	3 935 215	22 809 624	LPA	73 805	6.2%	977.3 ± 357.3	63.8×
			retention	77 352	5.3%	117.6 ± 41.3	7.7×
			FLPA	76 028	5.8%	15.3 ± 4.7	
livejournal	4 847 571	68 993 773	LPA	59 742	69.4%	2 248.1 ± 1 259.9	30.2×
			retention	149 555	60.5%	959.6 ± 477.0	12.9×
			FLPA	93 501	65.5%	74.4 ± 26.3	
twitter-sample	5 384 162	16 011 444	LPA	16 011	55.1%	1 343.5 ± 544.6	93.0×
			retention	20 651	47.5%	92.8 ± 38.9	6.4×
			FLPA	18 560	51.8%	14.5 ± 4.9	
bitcoin	6 297 539	16 057 711	LPA	42 388	50.9%	2 937.7 ± 1 077.3	80.3×
			retention	1 059 801	10.7%	597.1 ± 522.6	16.3×
			FLPA	247 404	24.3%	36.6 ± 13.0	

Table 2. Speedup of fast label propagation for empirical networks. Results are averages over 2 000 runs of the algorithms (1 986 runs for the `bitcoin` network).

Require: undirected multigraph $G(V,E)$
Ensure: node clustering $C = \{c_i\}_{i=1}^{n}$
$C \leftarrow \{1,2,3\ldots n\}$
while C **not maximal do**
  for $i \in$ **shuffle** V **do**
    $\{c\} \leftarrow \text{argsmax}_c \sum_{j \in V} A_{ij}\delta(c_j,c)$
    $c_i \leftarrow \text{random}_c\{c\}$
  end for
end while
return C

Algorithm 1. Label propagation algorithm (LPA).

Require: undirected multigraph $G(V,E)$
Ensure: node clustering $C = \{c_i\}_{i=1}^{n}$
$C \leftarrow \{1,2,3\ldots n\}$
while C **changed do**
  for $i \in$ **shuffle** V **do**
    $\{c\} \leftarrow \text{argsmax}_c \sum_{j \in V} A_{ij}\delta(c_j,c)$
    if $c_i \notin \{c\}$ **then**
      $c_i \leftarrow \text{random}_c\{c\}$
    end if
  end for
end while
return C

Algorithm 2. Label propagation algorithm with retention.

Require: undirected multigraph $G(V,E)$
Ensure: node clustering $C = \{c_i\}_{i=1}^{n}$
$C \leftarrow \{1,2,3\ldots n\}$
$Q \leftarrow$ **shuffle** V
while Q **not empty do**
  $i \leftarrow$ **pop** Q
  $\{c\} \leftarrow \text{argsmax}_c \sum_{j \in V} A_{ij}\delta(c_j,c)$
  $c \leftarrow \text{random}_c\{c\}$
  if $c_i \neq c$ **then**
    $c_i \leftarrow c$
    for $j \in N_i : c_j \neq c \wedge j \notin Q$ **do**
      $j \rightarrow$ **push** Q
    end for
  end if
end while
return C

Algorithm 3. Fast label propagation algorithm (FLPA).

Empirical networks. The large empirical networks from Table 2 are part of the Netzschleuder repository and can be downloaded from https://networks.skewed.de. All networks have been reduced to their largest connected component. The `com-dblp` is a co-authorship network extracted from the DBLP database in 2012[30], the `roadnet-ca` is the road network of California[31], the `us-patents` is the U.S. patent citation network from 1975 to 1999[32], the `foursquare` network represents check-in events on Foursquare from April 2012 to September 2013[33], the `livejournal` is an online social network of the LiveJournal members in 2006[34], the

`twitter-sample` is a sample of the Twitter follower network extracted in 2012[35], and the `bitcoin` is a network of Bitcoin transactions from January 2009 to April 2013[36].

The small social networks from Fig. 4 are either part of the Netzschleuder repository or available as supplementary material from Ref.[37]. The `karate` is a friendship network among members of a university karate club divided into two factions[38] (34 nodes, 78 edges). The `dolphins` is a social network of frequent associations observed among dolphins living off New Zealand[39], with a sociological division of dolphins into two groups (62 nodes, 159 edges). The `football` network represents American football games between U.S. colleges during the 2000 regular season[40], with each college assigned to one of twelve conferences (115 nodes, 616 edges). The `school-day1` and `school-day2` networks encode face-to-face interactions between children and teachers in a French elementary school on two consecutive days[37], where the metadata contain the assignment of children to 10 classes. Following the original study, we only include edges between individuals who interacted for at least 2 minutes (236 and 238 nodes, 1 956 and 2 177 edges, respectively).

Data availability

The code created for the current study is implemented in the `igraph` library, available from https://github.com/vtraag/igraph/tree/flpa. The datasets analysed during the current study are available in the Netzschleuder repository, https://networks.skewed.de, or via references in the published article.

Received: 27 September 2022; Accepted: 7 February 2023
Published online: 15 February 2023

References

1. Newman, M. E. J. & Girvan, M. Finding and evaluating community structure in networks. *Phys. Rev. E* **69**, 026113 (2004).
2. Peixoto, T. P. Bayesian stochastic blockmodeling. In Doreian, P., Batagelj, V. & Ferligoj, A. (eds.) *Advances in Network Clustering and Blockmodeling*, Computational and Quantitative Social Science, 281–324 (Wiley, New York, 2020), 1st edn.
3. Rosvall, M. & Bergstrom, C. T. An information-theoretic framework for resolving community structure in complex networks. *P. Natl. Acad. Sci. USA* **104**, 7327–7331 (2007).
4. Rosvall, M. & Bergstrom, C. T. Maps of random walks on complex networks reveal community structure. *Proc. Natl. Acad. Sci. U. S. A.* **105**, 1118–1123 (2008).
5. Clauset, A., Newman, M. E. J. & Moore, C. Finding community structure in very large networks. *Physi. Rev. E* **70**, 066111 (2004).
6. Reichardt, J. & Bornholdt, S. Statistical mechanics of community detection. *Phys. Rev. E* **74**, 016110 (2006).
7. Duch, J. & Arenas, A. Community detection in complex networks using extremal optimization. *Phys. Rev. E* **72**, 027104 (2005).
8. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *J. Stat. Mech.* **P10008** (2008).
9. Traag, V. A., Waltman, L. & Van Eck, N. J. From Louvain to Leiden: Guaranteeing well-connected communities. *Sci. Rep.* **9**, 5233 (2019).
10. Traag, V. A., Van Dooren, P. & Nesterov, Y. Narrow scope for resolution-limit-free community detection. *Phys. Rev. E* **84**, 016114 (2011).
11. Raghavan, U. N., Albert, R. & Kumara, S. Near linear time algorithm to detect community structures in large-scale networks. *Phys. Rev. E* **76**, 036106 (2007).
12. Šubelj, L. Label propagation for clustering. In Doreian, P., Batagelj, V. & Ferligoj, A. (eds.) *Advances in Network Clustering and Blockmodeling*, Computational and Quantitative Social Science, 121–150 (Wiley, New York, 2020), 1st edn.
13. Tibély, G. & Kertész, J. On the equivalence of the label propagation method of community detection and a potts model approach. *Physica A: Stat. Mech. Appl.* **387**, 4982–4984 (2008).
14. Garza, S. E. & Schaeffer, S. E. Community detection with the label propagation algorithm: A survey. *Physica A: Stat. Mech. Appl.* **534**, 122058 (2019).
15. Radicchi, F., Castellano, C., Cecconi, F., Loreto, V. & Parisi, D. Defining and identifying communities in networks. *Proc. Natl. Acad. Sci.* **101**, 2658–2663 (2004).
16. Barber, M. J. & Clark, J. W. Detecting network communities by propagating labels under constraints. *Phys. Rev. E* **80**, 026129 (2009).
17. Leung, I. X. Y., Hui, P., Liò, P. & Crowcroft, J. Towards real-time community detection in large networks. *Phys. Rev. E* **79**, 066107 (2009).
18. Šubelj, L. & Bajec, M. Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction. *Phys. Rev. E* **83**, 036103 (2011).
19. Xie, J. & Szymanski, B. K. Community detection using a neighborhood strength driven label propagation algorithm. In *Proceedings of the IEEE International Workshop on Network Science*, 188–195 (West Point, NY, USA, 2011).
20. Tasgin, M. & Bingol, H. O. Community detection using boundary nodes in complex networks. *Physica A: Stat. Mech. Appl.* **513**, 315–324 (2019).
21. Newman, M. E. J. *Networks* (Oxford University Press, Oxford, 2018), 2nd edn.
22. Danon, L., Díaz-Guilera, A., Duch, J. & Arenas, A. Comparing community structure identification. *J. Stat. Mech.* P09008 (2005).
23. Floretta, L., Liechti, J., Flammini, A. & De Los Rios, P. Stochastic fluctuations and the detectability limit of network communities. *Phys. Rev. E* **88**, 060801. https://doi.org/10.1103/PhysRevE.88.060801 (2013).
24. Meilă, M. Comparing clusterings: An information based distance. *J. Multivar. Anal.* **98**, 873–895 (2007).
25. Guimerà, R., Sales-Pardo, M. & Amaral, L. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E* **70**, 025101 (2004).
26. Fortunato, S. & Barthélemy, M. Resolution limit in community detection. *Proc. Natl. Acad. Sci.* **104**, 36 (2007).
27. Boldi, P., Rosa, M., Santini, M. & Vigna, S. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the International World Wide Web Conference*, 587–596 (Hyderabad, India, 2011).
28. Backstrom, L., Boldi, P., Rosa, M., Ugander, J. & Vigna, S. Four degrees of separation. In *Proceedings of the ACM International Conference on Web Science*, 45–54 (Evanston, IL, USA, 2012).
29. Lambiotte, R. & Ausloos, M. Coexistence of opposite opinions in a network with communities. *J. Stat. Mech.: Theor. Exp.* **2007**, P08026 (2007).
30. Yang, J. & Leskovec, J. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, 1–8 (Beijing, China, 2012).
31. Leskovec, J., Lang, K. J., Dasgupta, A. & Mahoney, M. W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.* **6**, 29–123 (2009).

32. Hall, B. H., Jaffe, A. B. & Tratjenberg, M. *The NBER patent citation data file: Lessons, insights and methodological tools* (Tech. Rep, National Bureau of Economic Research, 2001).
33. Yang, D., Zhang, D., Chen, L. & Qu, B. Nationtelescope: Monitoring and visualizing large-scale collective behavior in LBSNs. *J. Netw. Comput. Appl.* **55**, 170–180 (2015).
34. Backstrom, L., Huttenlocher, D., Kleinberg, J. & Lan, X. Group formation in large social networks: Membership, growth, and evolution. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 44-54 (Philadelphia, PA, USA, 2006).
35. Kagan, D., Elovichi, Y. & Fire, M. Generic anomalous vertices detection utilizing a link prediction algorithm. *Soc. Netw. Analy. Min.* **8**, 27 (2018).
36. Fire, M. & Guestrin, C. The rise and fall of network stars: Analyzing 2.5 million graphs to reveal how high-degree vertices emerge over time. *Information Processing & Management* **57**, 102041 (2020).
37. Stehlé, J. *et al.* High-resolution measurements of face-to-face contact patterns in a primary school. *PLoS ONE* **6**, e23176 (2011).
38. Zachary, W. W. An information flow model for conflict and fission in small groups. *J. Anthropol. Res.* **33**, 452–473 (1977).
39. Lusseau, D. *et al.* The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. Can geographic isolation explain this unique trait? *Behav. Ecol. Sociobiol.* **54**, 396–405 (2003).
40. Girvan, M. & Newman, M. E. J. Community structure in social and biological networks. *P. Natl. Acad. Sci. USA* **99**, 7821–7826 (2002).

Acknowledgements

This work has been supported in part by Slovenian Research Agency ARRS under the program P5-0168. We gratefully acknowledge use of the Shark cluster of the LUMC for computation time.

Author contributions

V.T. conceived the experiments, V.T. and L.Š. designed the algorithms, V.T. and L.Š. analysed the results, V.T. and L.Š. prepared the manuscript. Both authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to V.A.T.

Reprints and permissions information is available at www.nature.com/reprints.

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.