Physica A 392 (2013) 5772–5785

Contents lists available at ScienceDirect

Physica A

journal homepage: www.elsevier.com/locate/physa

Personalized PageRank Clustering: A graph clustering algorithm based on random walks

Shayan A. Tabrizi [a], Azadeh Shakery [a,b,*], Masoud Asadpour [a,b], Maziar Abbasi [a], Mohammad Ali Tavallaie [a]

[a] School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran

[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), P.O. Box 19395-5746, Tehran, Iran

HIGHLIGHTS

- This algorithm employs random walks to accurately and efficiently cluster graphs.
- The top-down approach makes it a flexible solution for various clustering requirements.
- It is superior to most of the clustering algorithms in accuracy and running time.
- Plenty of performance optimizations are used to reduce the running time to linear.

ARTICLE INFO

Article history:
Received 16 June 2012
Received in revised form 10 May 2013
Available online 24 July 2013

Keywords:
Social networks
Clustering
Community detection
PageRank
Random walks

ABSTRACT

Graph clustering has been an essential part in many methods and thus its accuracy has a significant effect on many applications. In addition, exponential growth of real-world graphs such as social networks, biological networks and electrical circuits demands clustering algorithms with nearly-linear time and space complexity. In this paper we propose Personalized PageRank Clustering (PPC) that employs the inherent cluster exploratory property of random walks to reveal the clusters of a given graph. We combine random walks and modularity to precisely and efficiently reveal the clusters of a graph. PPC is a top-down algorithm so it can reveal inherent clusters of a graph more accurately than other nearly-linear approaches that are mainly bottom-up. It also gives a hierarchy of clusters that is useful in many applications. PPC has a linear time and space complexity and has been superior to most of the available clustering algorithms on many datasets. Furthermore, its top-down approach makes it a flexible solution for clustering problems with different requirements.

© 2013 Elsevier B.V. All rights reserved.

1. Introduction

Graphs play an essential role in system modeling. In an enormous number of domains such as biology, social networks and electrical engineering, graphs are used to model the real-world systems. So proposing more accurate and faster graph algorithms can help researchers in many domains obtaining better results.

One of the most important issues about graphs is how to detect clusters (communities). Clusters are subgraphs that are highly intra-connected and have a low inter-connectivity. The clustering problem can be seen as an optimization problem and for which many cost (or goal) functions have been proposed. In this paper we find clusters according to the people's

* Corresponding author at: School of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran. Tel.: +98 2182089722.

E-mail addresses: s.tabrizi@ut.ac.ir (S. A. Tabrizi), shakery@ut.ac.ir, shakery@gmail.com (A. Shakery), asadpour@ut.ac.ir (M. Asadpour), ma.abbasi@ut.ac.ir (M. Abbasi), ma.tavallaie@ece.ut.ac.ir (M.A. Tavallaie).

0378-4371/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.physa.2013.07.021

communication model. So in applications such as social networks it will obtain very good results because the clustering cost function is consistent with the underlying logic of the community formation, although it also achieves good results for other kinds of graphs. We will explain this more precisely in Section 4.

On the other hand, due to the explosive growth of the real-world graphs like social networks, classical algorithms are not sufficient anymore, since they usually have high space or time complexity. So developing new algorithms that scale linearly in the graph size (i.e. the number of vertices and the number of edges) can extend the domain of problems solvable in a reasonable time.

In this paper we introduce a novel approach to clustering that combines random walks and modularity to disclose the intrinsic clusters of an undirected graph, both weighted and unweighted. The reason why random walks are suitable for clustering is that a limited-length random walk tends to stay inside a cluster with high probability rather than transitioning to a different cluster. That is because the intra-connectivity of a good cluster is significantly larger than its inter-connectivity. So if we have many random walks starting from a single vertex within a precisely defined cluster most of them will probably get trapped in that cluster. Thus, vertices in other clusters will have considerably less visits than the vertices in the same cluster.

So we start many limited-length random walks from a vertex and sort the vertices according to the number of visits to obtain a list of vertices sorted by their cluster-similarity to the starting vertex. Then, if we find a suitable cutting point in the list, the graph could be partitioned into two subgraphs that are reasonably separated. So by doing this recursively until a certain condition is met a hierarchy of clusters is made.

We call our proposed method Personalized PageRank Clustering (PPC) and show that it has very nice properties and is superior to the existing algorithms in different aspects. We will show that PPC has a linear time complexity which means that it is scalable to very large graphs. We will also show that PPC is accurate in the sense that it uses not only the immediate neighbors of a vertex but also vertices with higher distances and thus its viewpoint is not local such as many other algorithms. On the other hand, unlike most of the fast graph clustering algorithms (e.g. Refs. [1,2]) PPC is a top-down (divisive) algorithm so it can have a holistic view of the graph and find global structures that are visible to a human observer better and thus it results in more reasonable clusters. In fact, PPC tends to find coarse-grained clusters that are significant from the viewpoint of a human observer and not fine-grained clusters that are only locally sensible. Its top-down approach also makes it suitable for interactive clustering. The user can view the result of each partitioning phase and go deeper and zoom in or zoom out, if desired, which reduces the time complexity of the algorithm when just a few phases are sufficient for the user.

Moreover, PPC can determine the number of clusters automatically which is crucial in many applications. Furthermore, because of the top-down approach of PPC it is very flexible. For example, PPC is easily applicable when the number of clusters is known or a certain threshold is desirable for the diameter of the clusters. In fact, its top-down approach lets us stop the algorithm whenever any needed conditions are met. In addition, PPC can further be extended to produce balanced clusters by partitioning the bigger clusters first.

The rest of this paper is organized as follows. Section 2 surveys major clustering algorithms and the algorithms using random walks. Section 3 discusses PageRank and personalized PageRank and their basic properties as well as a method for estimating personalized PageRank scores efficiently. In Section 4 the intuitions that PPC is based on are explained. Then, in Section 5 we introduce Forward Partitioning (FP) and Backward Partitioning (BP) and propose PPC based on a modified version of BP. Section 5 also covers the optimizations necessary for reducing the running time of the algorithm. In Section 6 PPC is thoroughly compared to other clustering algorithms and its running time is analyzed. Finally, this paper is concluded in Section 7 with some notes on PPC.

2. Related work

Clustering has a lot of applications and each application demands some specific features. Because of this, a globally-accepted definition of clustering does not exist. Thus, many algorithms have been proposed and each of them tries to solve the problem from its own perspective. Among the graph clustering algorithms whose main concerns are low time complexity and scalability, Refs. [1,3] are among the state-of-the-art ones [4]. The former uses a greedy approach based on modularity (see Definition 2) and the latter exploits information theory to cluster graphs. We will compare PPC to these algorithms in Section 6.1.

Many recent algorithms employ random walks [2,3,5–14]. The reason why random walks have recently been so popular in many applications is their ability to go beyond the local neighborhoods of the starting vertices that makes them ideal for getting rid of local decisions. We will further explain their characteristics that make them appropriate for clustering applications in Section 4. However, these algorithms have deficiencies that will be discussed in the rest of this section.

Hagen and Kahng [5] proposed the first method that uses random walks to detect clusters. This method uses random walks to effectively cluster circuits and is especially used in VLSI design. It defines a concept called cycle and by finding cycles tries to cluster graphs. The worst-case complexity of this algorithm is $O(n^3)$ and the space requirement is $O(n^2)$. Both of these complexities are infeasible for most of the real-world graphs.

Harel and Koren [6] introduce a definition of clustering, based on random walks. Then they propose two operators that change the weights of the edges and reduce the weights of inter-cluster edges. By enforcing them iteratively the inter-cluster edge weights are reduced until they are negligible. Afterwards, the cluster structure is revealed by omitting the near-zero-weight edges. This idea is similar to the idea used in Ref. [15] in which edge betweenness is used instead of random walks. One pitfall of this algorithm is that it limits the number of steps of each random walk. Although, this assumption is logical in

many cases, since the probability of short random walks is higher than long random walks, it might be inadequate in some cases and therefore worsen the results. It also assumes that graphs have bounded degree to achieve linear running time that is restrictive. Furthermore, as Ref. [16] states, divisive methods employing edge centrality based on walks tend to first remove the edges connected with a leaf vertex, and not the bridges between communities. So it is expected that the quality of this algorithm is not much remarkable. Ref. [16] is a good comparison of community detection methods based on edge centrality.

Some methods try to use the benefits of random walks in other known algorithms. For example Refs. [7–9] use a random walk based distance in k-means or k-medoids algorithms and improve them considerably. But they have a superlinear time complexity that is infeasible for clustering large graphs.

A number of methods use random walks just to partition graphs locally around a specific vertex [10–12]. These algorithms try to find a cluster in which the vertex exists and has low connection with the rest of the graph. The importance of these algorithms is mainly because their complexity largely depends on the size of the small side of the cut that might be asymptotically smaller than that of the whole graph [10]. Refs. [10–12] find a cut with conductance at most ϕ whose small side has volume at least 2^b in time $O(2^b \log^3 m/\phi^2)$ and $O(2^b \log^2 m/\phi^2)$ respectively. Conductance [10,12] for a subset $S \subseteq V$ of vertices is defined to be

$$\Phi(S) = \frac{|\partial(S)|}{min(vol(S), 2m - vol(S))}, \tag{1}$$

in which $\partial(S) = \{\{x, y\} \in E | x \in S, y \notin S\}$ and volume of a set is:

$$vol(S) = \sum_{x \in S} \mathrm{d}(x). \tag{2}$$

Refs. [10–12] do not perform any experiments on any graphs, so their actual results on real graphs are unknown. In Ref. [13] an algorithm is proposed that outperforms similar methods in accuracy but increases the time complexity. This algorithm employs the multi-agent random walk instead of the ordinary random walk. In a multi-agent random walk several agents connected by a fixed rope of length l are used instead of a single agent. Using this kind of random walk, the clusters are found more accurately since it is less probable that several agents simultaneously travel over the bottleneck of a graph than only one agent.

Rosvall and Bergstrom [2] employ information theory to obtain a good clustering method based on flows. The method considers random walks as a "proxy for information flows in real system". It formalizes the clustering problem as an information theory problem and solves the problem using random walks. This algorithm is reasonably accurate as discussed in Ref. [4] and uses the information embedded in the graph very well. This paper uses a power method to calculate the PageRank scores that has a superlinear time complexity. In Ref. [3] they use a similar methodology but extend it to obtain a hierarchy of clusters. Since these methods are based on flows they perform well on graphs with links representing movement patterns but may not be suitable for graphs with links representing pairwise relationships. In addition, this method finds a lot of fine-grained clusters that may seem insignificant from a global perspective. Its time complexity is also bothering for large graphs as will be shown in Section 6.1.

In Ref. [14] an algorithm is proposed that approximates MaxCut [17] better than the greedy algorithm using random walks. But its time complexity is at least $O(n^{1.5})$ that is infeasible for big graphs.

As an example of the recent top-down algorithms, Ref. [18] which is based on relative link strength is considerable. It emphasizes on the strength of weak ties hypothesis [19], which states that the strength of a tie between two nodes A and B increases with the overlap of their friendship circles, which in turn results in the importance of weak ties in connecting communities. Thus, the paper measures the relative topological overlap (relative link strength) for each pair of vertices and uses the minimum link strength as a measure for determining if a cluster is coherent enough or not. Thereafter, it employs a top-down approach to cluster the graph. Its main problem, such as in many other top-down algorithms, is its time complexity, which is $O(mn)$. Besides, its NMI [20] drops relatively fast compared to PPC, which is its drawback. This will be further discussed in Section 6.2.

In addition, much recent research has been devoted to discovering overlapping structures (e.g. Refs. [21–23]), which is out of the scope of this paper, but is considered as a part of the future work. We just briefly mention the interesting idea in Ref. [23]. The paper employs a generalized version of modularity [24], which is based on the definition of stability [25], to cluster graphs. It then introduces random walking on links, which could intuitively model community overlaps, and exploits it, instead of normal random walks, to identify overlapping clusters.

3. Preliminaries

PageRank was introduced by Brin and Page [26]. It calculates the centrality of each vertex. The higher the number of inedges of a vertex, the more central it is. In addition, if the vertices that have outedges to this vertex are more central, it will also be more central.

There is another interpretation of PageRank that is more intuitive. This interpretation, called the random surfer model [26], assumes that an agent is walking on the graph. At each vertex it randomly selects one of the outedges and goes to

the vertex on the other side. In addition, it might jump to a random vertex in the graph, to which the current vertex is not necessarily connected, with a certain probability. The latter ensures that the algorithm will converge and also it will not trap in a subgraph without any outedges. The probability of the agent being in each vertex is the PageRank score of that vertex. This model tries to mimic the behavior of a person surfing a graph, especially the web.

Refs. [26,27] extend the definition of PageRank and propose personalized PageRank (a.k.a. Topic-Specific PageRank and Topic-Sensitive PageRank). In the basic PageRank algorithm, all vertices have the same chance to be chosen as the destination of the random jumps. But personalized PageRank (PPR) considers a probability vector $\vec{v}$ containing the probabilities of jumping to each vertex. So PPR vector $\vec{pr}(\alpha, \vec{v})$ for a graph with row-normalized adjacency matrix M is defined to be the unique solution of Eq. (3), in which α is the random jump probability:

$$\vec{pr}(\alpha, \vec{v}) = \alpha\vec{v} + (1-\alpha)M^T\vec{pr}(\alpha, \vec{v}). \tag{3}$$

A fundamental property of PPR is its linearity, as defined in Theorem 1, that is a vital property for scalability of our algorithm.

Theorem 1 (*Linearity [27]*). *For any probability vectors $\vec{v_1}$, $\vec{v_2}$ and positive constants β_1, β_2 with $\beta_1 + \beta_2 = 1$, Eq. (4) holds:*

$$\vec{pr}(\alpha, \beta_1\vec{r_1} + \beta_2\vec{r_2}) = \beta_1\vec{pr}(\alpha, \vec{r_1}) + \beta_2\vec{pr}(\alpha, \vec{r_2}). \tag{4}$$

Since $\alpha = 0.7$ is considered as a constant throughout this paper, we use Definition 1 for simplicity.

Definition 1. $p\vec{p}r_S$ where S is a vertex or a set of vertices is an abbreviation for a PPR score vector in which $p\vec{p}r_S(v)$ is the PPR score of v having S as the target of the random jumps and call it PPR scores from S.

Considering Theorem 1, we can easily add up PPR scores from the vertices of a set of vertices to obtain the PPR score from that set of vertices. This allows our algorithm to update PPR scores from a set of vertices incrementally when adding new vertices to the set.

We use a Monte Carlo method [28] to reduce the cost of this algorithm to linear. The idea is to use real random surfers to estimate the PageRank scores. Ref. [29] uses the same idea to make fully personalized PageRank scalable.

If we had an indefinite number of random surfers we could calculate PPR scores accurately. But this would take an unlimited time. So we limit the number of random surfers. For each vertex v, we start a number of random walks that is proportional to the degree of v, from it. For more accuracy, we start at least a minimum number of l random walks from each vertex, regardless of its degree, for some constant l. Thus, for each vertex v we start $max(l, k \times deg(v))$ random walks from v for some constants k, l. We use $k = 5$, $l = 50$ in our experiments. Thus the total number of random walks will be linear in the size of the graph (i.e. $\Theta(k \times m + n)$, in which n and m are the number of vertices and edges respectively). So the time complexity reduces to linear. The $p\vec{p}r_u(v)$ for two vertices u, v could be estimated as the number of visits random surfers starting from u make to v.

4. Intuitions

In this section we will clarify why PPR, that is the basis of PPC, is appropriate for exploring the cluster structure of graphs and explain how this property is used in other applications implicitly.

4.1. Spammers

Nowadays web spamming has emerged to gain economic benefits of high search rankings and disturbed the accuracy of those rankings. Among different methods of web spamming, link spam is strongly related to the importance of a site from the viewpoint of other sites [30]. If other sites have so many out-links to the site, it means that the site is important and it will move up in the rankings. With regard to this property, spammers make a farm of highly inter-connected web sites to raise their site rankings. Technically speaking, search engines use centrality measures like PageRank and PPR to determine the importance of each page. When a subgraph of the web is strongly dense it acts like a black hole and holds the PageRank, or personalized PageRank, score in itself. So its vertices get unfairly high scores and consequently raise in rank unfairly.

The key property that makes this kind of spamming work is that in a strongly dense subgraph candidates of a random surfer for being the next vertex are biased towards the vertices inside the subgraph. So a random surfer tends to stay inside a cluster rather than transitioning to a different cluster. This gives us an intuition why actually personalized PageRank can explore the cluster structure of graphs.

4.2. Friendships

PPR has an interesting property that is the basis of our algorithm. Studying the patterns of making friendship in societies, it is understood that the more common friends two persons have, the more probable they make a friendship (triadic closure

property [31]). That is because when two persons are more connected to each other they probably have more contacts with each other and so it is more probable that they come to know each other and make a friendship.

Also when two individuals are more connected they are probably more similar to each other than two individuals without any connections. So they are probably better candidates for being friends.

We can extend the concept of *common friends* by taking the number of friend-once-removeds, friend-twice-removeds and etc. into account. Actually PPR does it for us because the $\vec{ppr}_u(v)$ for a given vertex v is proportional to the number of common friends, friend-once-removeds, and etc. So when we want to determine which vertex is more similar to a given set of vertices we can choose either the vertex with the highest PPR score from the given vertices or the vertex that the given set of vertices take the highest sum of PPR scores from.

5. PPC Algorithm

PPC is a top-down algorithm based on partitioning. It initially considers the whole graph as a single cluster and then recursively partitions each cluster until further partitioning worsens the results. In this section, we first describe the structure of PPC as well as the basis of the partitioning algorithms. Afterwards, we introduce Forward Partitioning (FP) and describe its pros and cons. Then we propose Backward Partitioning (BP) that solves the main deficiency of FP, low accuracy. Afterwards, a modified version of BP is introduced which increases the quality of results considerably and is employed in PPC to partition the clusters.

BP and FP have the same basis described in Algorithm 1 and are based on a combination of random walks and modularity. They initially consider a random vertex as a community (cluster) and iteratively expand the community until the whole graph is considered as a single community. The best community formed during these iterations is considered as the inherent subgraph of the input graph. The measure used for determining the goodness of communities is modularity [32] as defined in Definition 2. An important property of modularity that makes it suitable for measuring clusters is that it can compare clusterings with different number of clusters.

Definition 2 (*Modularity [32]*)**.** The modularity of a graph with respect to a set of clusters is defined by Eq. (5):

$$Q = \frac{1}{4m} \sum_{ij} \left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j), \tag{5}$$

where A_{ij} represents the weight of the edge between vertices i and j, $k_i = \sum_j A_{ij}$, c_i is the cluster number to which vertex i is assigned, $\delta(x, y)$ is the Kronecker delta function that is 1 if $x = y$ and 0 otherwise, and $m = \frac{1}{2} \sum_{ij} A_{ij}$.

The basis of the partitioning algorithms is described in Algorithm 1. First, we put a randomly chosen vertex in set S and all other vertices in T. Then at each step we find the most similar vertex in T to the vertices in S by FIND-BEST-VERTEX(G, S, T) and move it to S. The two partitioning algorithms differ in this procedure and use different approaches to find the most similar vertex. In addition, modularity is updated in line 9 using Theorem 2 with amortized cost $O(2m/n)$ (i.e. the average degree of vertices).

We repeat these steps iteratively until all vertices are added to S. During these steps we store the best partitions found in lines 8–12 according to the modularity gain. The partitioning with the highest positive modularity gain, if any, is considered as the final partitioning of this phase.

Theorem 2 (*Modularity Gain*)**.** *The modularity gain obtained by moving a single vertex v from a cluster T into a cluster S is calculated by*

$$\Delta Q = \frac{1}{m} \left(\sum_{j \in S} A_{vj} - k_v \frac{\sum_{j \in S} k_j}{2m} \right) - \frac{1}{m} \left(\sum_{j \in T'} A_{vj} - k_v \frac{\sum_{j \in T'} k_j}{2m} \right),$$

where $T' = T - \{v\}$. This could be easily calculated in $O(2m/n)$ on average by saving $\sum_{j \in S} k_j$ and $\sum_{j \in T'} k_j$ and updating them at each step in $O(1)$.

PPC uses partitioning to cluster the whole graph in a top-down manner. It first partitions the input graph into two subgraphs (we call each partitioning a phase of the algorithm). Then at each phase the subgraph with the highest modularity gain after being partitioned is chosen and is partitioned. This procedure continues until there is no more increase in the modularity of the graph. The resulting subgraphs are considered as the final clusters of that graph.

5.1. Forward partitioning

As discussed in Section 5, Forward Partitioning and Backward Partitioning have the same basis. The only difference is in the FIND-BEST-VERTEX(G, S, T) procedure. To estimate the similarity of each vertex in T to the vertices in S, FP calculates the PPR score of each vertex in T from S. It then divides the score of each vertex by its degree so that the bias towards the

Algorithm 1 $\langle S, T\rangle \leftarrow$Partitioning($G$)

Require: $\vec{ppr}_u(v)$ = PPR score of v from u and $G = \langle V, E\rangle$

$v \leftarrow$ a randomly chosen vertex
$S \leftarrow \{v\}$
$T \leftarrow V - S$
$mod \leftarrow 0$
$mod_{max} \leftarrow 0$
while $T \neq \emptyset$ **do**
  $mod \leftarrow mod +$Modularity-Gain(G, v, S, T)
  if $mod > max_{mod}$ **then**
    $mod_{max} \leftarrow mod$
    $S_{max} \leftarrow S$
    $T_{max} \leftarrow T$
  end if
  $v \leftarrow$ Find-Best-Vertex(G, S, T)
  $S \leftarrow S \cup \{v\}$
  $T \leftarrow T - \{v\}$
end while
if $mod_{max} > 0$ **then**
  return $\langle S_{max}, T_{max}\rangle$
else
  return NIL
end if

vertices with high degree is removed. The higher this score is, the more similar the vertex is to the vertices in S as discussed in Section 4.2.

If the PPR scores are calculated accurately in FP, the overall time complexity of calculating them in each phase of PPC will be $O(mn \log n)$. So the total complexity of PPC will be $O(c(mn \log n))$, in which c is the number of final clusters, that is infeasible for many of the real-world graphs. To overcome this problem, we use the Monte Carlo estimation method described in Section 3. We start $min(l, k \times deg(v))$ random walks from each vertex v in S instead of performing a PPR from it. In fact this is done incrementally. At each step it is sufficient to start random walks only from the chosen vertex v and update $\vec{ppr}_S$ accordingly. So random walks are started from each vertex only once and thus the total number of random walks is linear in the size of the graph.

In addition, to select the most similar vertex in T to the vertices in S, we use a Fibonacci max-heap [33] that stores the estimated PPR scores. In fact, in order to be able to use the Increase-Key procedure we store the number of random walks themselves that are monotonically increasing, not the PPR scores. But we normalize them so that the starting vertices with different number of random walks are treated equally. So Insertion and Increase-Key take only $O(1)$ time and Extract-Max takes $O(\log n)$ time. Thus, the total time complexity of extracting the best vertex in each phase of PPC is $O(n \log n)$ that is practically $O(m + n)$ since in most of the real-world graphs such as social networks $m = \Omega(n \log n)$.

Therefore, PPC using FP has a worst-case complexity of $O(c(km + n \log n))$. Considering that the number of clusters is asymptotically lower than the size of the graph in most of the real-world applications, the time complexity is approximately linear in the graph size. Moreover, this complexity is the worst-case complexity and as will be shown in Section 6.3 the average complexity of PPC is $O((m + n) \log c)$ which is $O(m + n)$ practically. The problem with FP is its lower accuracy compared to BP, as will be discussed in Section 6.1.

5.2. Backward partitioning

Unlike Forward Partitioning, Backward Partitioning calculates the sum of PPR scores given to the vertices in S from each vertex v in T (i.e. $score_v = \sum_{u \in S} \vec{ppr}_v(u)$). BP then chooses the vertex v with the highest $score_v$. The logic of BP is similar to that of FP but they view the same problem from two opposite perspectives. Having the friendships' intuition in mind, FP selects the vertex that the group of friends, S, are more likely to make a friendship with while BP selects the vertex that is more eager to join the group of friends.

The problem of time complexity is even more severe in BP since calculating the scores accurately leads to a time complexity of $O(c(mn^2 \log n))$. To solve this problem, we use the Monte Carlo method mentioned in Section 3 again. At the beginning of the BP, we start $min(l, k \times deg(v))$ random walks from each vertex v, and in each vertex u store the number of visits random surfers starting from v make to u. Then, at each step when a vertex v is moved from T to S, the scores of the vertices in T could be updated efficiently by just updating the scores of the vertices whose random walks have visited v. Thus, since the total number of visits is linear in the size of the graph, the total time complexity of updating PPR scores in each phase of PPC is linear. In addition, using a Fibonacci max-heap makes the selection of the best vertex possible in a

Algorithm 2 $\langle S, T\rangle \leftarrow$REVISED-BP($G$)

Require: $G = \langle V, E\rangle$
1: SURF-AND-SAVE(G)
2: $v \leftarrow$ a randomly chosen vertex
3: $S \leftarrow \{v\}$
4: $T \leftarrow V - S$
5: $mod \leftarrow 0$
6: $mod_{max} \leftarrow 0$
7: $H \leftarrow$ INITIALIZE-FIBONACCI-HEAP-BP(v)
8: **while** $T \neq \emptyset$ **do**
9: $mod \leftarrow mod +$MODULARITY-GAIN(G, v, S, T)
10: **if** $mod > mod_{max}$ **then**
11: $mod_{max} \leftarrow mod$
12: $S_{max} \leftarrow S$
13: $T_{max} \leftarrow T$
14: **end if**
15: $v \leftarrow$ FIND-BEST-VERTEX-BP(G, H)
16: $S \leftarrow S \cup \{v\}$
17: $T \leftarrow T - \{v\}$
18: **end while**
19: **if** $mod_{max} > 0$ **then**
20: $\langle S_{max}, T_{max}\rangle \leftarrow$ REARRANGE-VERTICES(G, S_{max}, T_{max})
21: **return** $\langle S_{max}, T_{max}\rangle$
22: **else**
23: **return** *NIL*
24: **end if**

linear time and thus the total time complexity of each phase reduces to linear. So similar to the PPC with FP, the total time complexity reduces to $O((m + n)\log c)$ in practice. Moreover, the space complexity remains linear since the total number of random walks is linear in the number of edges, with a small constant factor.

5.3. Personalized PageRank Clustering

Personalized PageRank Clustering employs Backward Partitioning to cluster graphs. Moreover, one additional step is added to reduce the effect of noise, which might be the result of estimations used throughout the algorithm. The pseudocode is shown in Algorithm 2, in which REARRANGE-VERTICES(G, S_{max}, T_{max}) corresponds to the added step. This procedure examines each vertex to determine whether moving the vertex to the other partition increases the modularity, and if yes, moves the vertex to that partition. To ensure that the time complexity remains unchanged, each vertex is moved at most once in each partitioning phase. As we will see in Section 6.1, although an additional step is added to the algorithm, the running time reduces since resulting partitions are less noisy and thus fewer partitioning phases are required to cluster the whole graph. Other red-marked lines in Algorithm 2 (Lines 1, 7, and 15) are the steps modified to efficiently implement Backward Partitioning.

The code of this algorithm could be freely downloaded from http://khorshid.ut.ac.ir/~s.tabrizi/. In Section 6 we will compare PPC with other state-of-the-art algorithms and discuss its results.

6. Experiments

In this section, we first examine the quality of the results of PPC and compare it to that of the state-of-the-art algorithms in Section 6.1. Then, LFR benchmarks [34] are employed to analyze the PPC quality more thoroughly. Thereafter, we will analyze PPC to find out why it is practically fast despite its top-down approach, in Section 6.3.

6.1. Clustering results

In this section PPC, that uses the modified Backward Partitioning, is applied to some benchmark graphs and the quality of the resulting clusters is examined. In addition, it is compared to PPC with Forward Partitioning and PPC with original Backward Partitioning that we call them FPC and BPC algorithms respectively, Infohiermap algorithm [3] and Louvain algorithm [1]. Algorithms [1,3] are relatively fast, acceptedly accurate and among the state-of-the-art algorithms [4].

In Fig. 1 we have visualized the result of our algorithm on the graph produced by the algorithm proposed in Ref. [34] with default parameters. In this paper the graphs have been laid out using the Force Atlas 2 algorithm, of the open source software Gephi [35]. It can be seen that PPC finds clusters remarkably accurately.

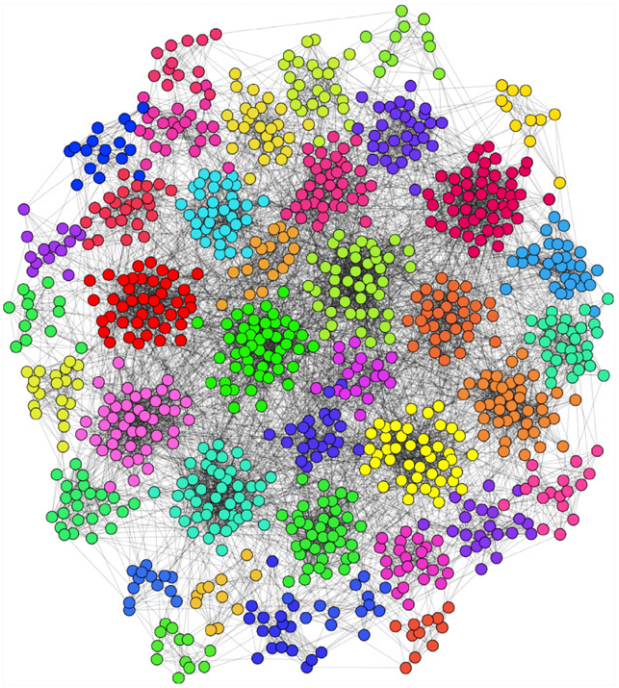


Fig. 1. The visualization of the PPC clusters of the graph produced by Ref. [34]. Vertices with the same color correspond to clusters.

Table 1
Modularity and running time of FPC, BPC, PPC, Louvain and Infohiermap for different datasets.

Dataset	n	m	FPC	BPC	PPC	Louvain	Infohiermap
Karate	34	78	0.374/0s	0.406/0s	0.419/0s	0.419/0s	0.402/0s
Dolphins	62	159	0.505/0s	0.502/0s	0.519/0s	0.517/0s	0.525/0s
Lesmis	77	254	0.477/0s	0.544/0s	0.544/0s	0.565/0s	0.536/0s
PolBooks	105	441	0.523/0s	0.502/0s	0.516/0s	0.527/0s	0.527/0s
AdjNoun	112	425	0.254/0s	0.273/0s	0.285/0s	0.297/0s	0.009/0s
Football	115	613	0.600/0s	0.589/0s	0.600/0s	0.604/0s	0.601/0s
PolBlogs	1K	17K	0.423/1s	0.425/0s	0.426/s	0.426/0s	0.425/1s
Yeast	5K	17K	0.469/1s	0.529/0s	0.554/0s	0.562/0s	0.494/2s
Arxiv	9K	24K	0.755/0s	0.783/0s	0.802/1s	0.814/0s	0.794/4s
CiteData	81K	482K	0.714/13s	0.746/43s	0.758/36s	0.754/1s	0.731/311s
NotreDam	326K	1M	0.929/28s	0.934/63s	0.939/47s	0.937/6s	0.929/340s
WebGoogle	876K	4M	0.925/116s	0.957/259s	0.969/259s	0.978/14s	0.976/30m

The graphs we consider here in this section for comparisons include the famous karate club network [36], a network of 62 bottlenose dolphins [37], a network of co-appearances of characters in Victor Hugo's novel "Les Miserables" [38], a network of co-purchasing of books about US politics by the same buyers [39], a network of common adjective and noun adjacencies for the novel "David Copperfield" by Charles Dickens [40], a network of American football games between Division IA colleges during regular season Fall 2000 [15], a network of political blogs around the time of the 2004 presidential election [41], a dataset of yeast protein–protein interaction [42], a network of 9000 scientific papers and their citations [43], CiteData collection of academic articles extracted from CiteULike and CiteSeer repositories [44], a webpage network of a few hundred thousand webpages from nd.edu domain [45], and a webgraph released by Google as a part of Google Programming Contest [46]. Edge directions, if any, are ignored in these graphs.

In Table 1 the results of PPC are compared to those of FPC, BPC, Louvain and Infohiermap with respect to modularity and running time. As is clear, the modularities of FPC are considerably lower than those of PPC on average. That is because division of the PPR scores by the degree of the vertices distances FP from the natural cluster exploratory of random walks, which in turn reduces the accuracy of FPC. Furthermore, the quality of BPC is affected by the estimation noises, and therefore although its results are comparable with other state-of-the-art algorithms, PPC outperforms it by overcoming the noise problem through the additional step. Since other analyses confirmed the superiority of PPC to FPC and BPC, in the rest of this chapter we only focus on evaluating PPC for brevity.

On the other hand, the modularities of PPC and Louvain are approximately in the same range in most of the datasets while that of Infohiermap is much lower in several datasets. An important point that should be noted is that the objective function of Louvain is just modularity and thus is biased towards high modularities. So it is not surprising that Louvain reaches higher

Table 2
The modularities of CNM, PL, WT and PPC algorithms for Arxiv dataset.

Algorithm	Modularity
CNM	0.772
PL	0.757
WT	0.761
PPC	0.802

modularities in some datasets and this should not be considered its superiority without thoroughly examining the results. However, although the modularity of PPC is lower than that of Louvain for some of the datasets, it is still much higher than those of many other methods. For example, in the Arxiv dataset PPC has lower modularity than Louvain. But when the results are compared with the results reported in the Louvain paper, Table 2, it becomes clear that PPC outperforms CNM [47], PL [48] and WT [49] algorithms with respect to modularity.

In addition, the running times of PPC and Louvain are asymptotically similar. Although PPC has a longer running time, its running time still grows linearly in the size of the graph, similar to Louvain, and thus it is applicable to large graphs. On the other hand, the running time of Infohiermap grows very fast to the size of the graph. This makes it impossible to be used for large graphs. Furthermore, the Infohiermap paper suggests that the algorithm is repeated about 100 times to get close to the global optimum. But, Infohiermap with repetition takes much more time, which makes it completely infeasible practically. It is noteworthy that the random walk based approach of PPC allows it to be used in a distributed setting, as will be discussed in Section 7, which can make it scalable to huge graphs.

In Fig. 2 the results of the three algorithms are visualized on the dolphins and the political blogs datasets. In the dolphins dataset PPC partitions the graph into 5 partitions while Louvain and Infohiermap partition it into 6 partitions. The results of both PPC and Louvain seem reasonable while it is clear that Infohiermap has a partition that is too small to be considered significant from a global perspective.

Furthermore, as Fig. 2(b) shows, PPC and Louvain partition the political blogs dataset into 9 partitions whereas Infohiermap produces 42 partitions. It is clear that 9 partitions seem to be more reasonable from a holistic view and the result of Infohiermap is too localized in some situations. Furthermore, in the PPC result 98.53% of the vertices lie in the two big clusters, while for Louvain it is just 96.16%, which shows that Louvain is more noisy. Some parts of the results that seem noisy are marked in the figure which confirms that the result of PPC is much less noisy.

In conclusion, the experiments state that PPC and Louvain are superior to Infohiermap in most of the datasets. However, it is still hard to make a final judgment about the superiority of either PPC or Louvain and we will postpone answering this question to Section 6.2. But, Louvain seems faster practically, although there is no theoretical analysis of its time complexity which can make it useless for real-time applications. On the other hand, the random walk based approach of PPC allows it to be employed in a distributed setting, as will be discussed in Section 7, which can make it an ideal solution for clustering very large graphs. Moreover, the top-down approach of PPC makes it much more flexible than bottom-up methods. For example, in a top-down approach, the user can force the algorithm to divide the largest clusters first, which can make the final clusters approximately balanced. Nevertheless, bottom-up approaches cannot merge the two smallest clusters first since they might be completely irrelevant.

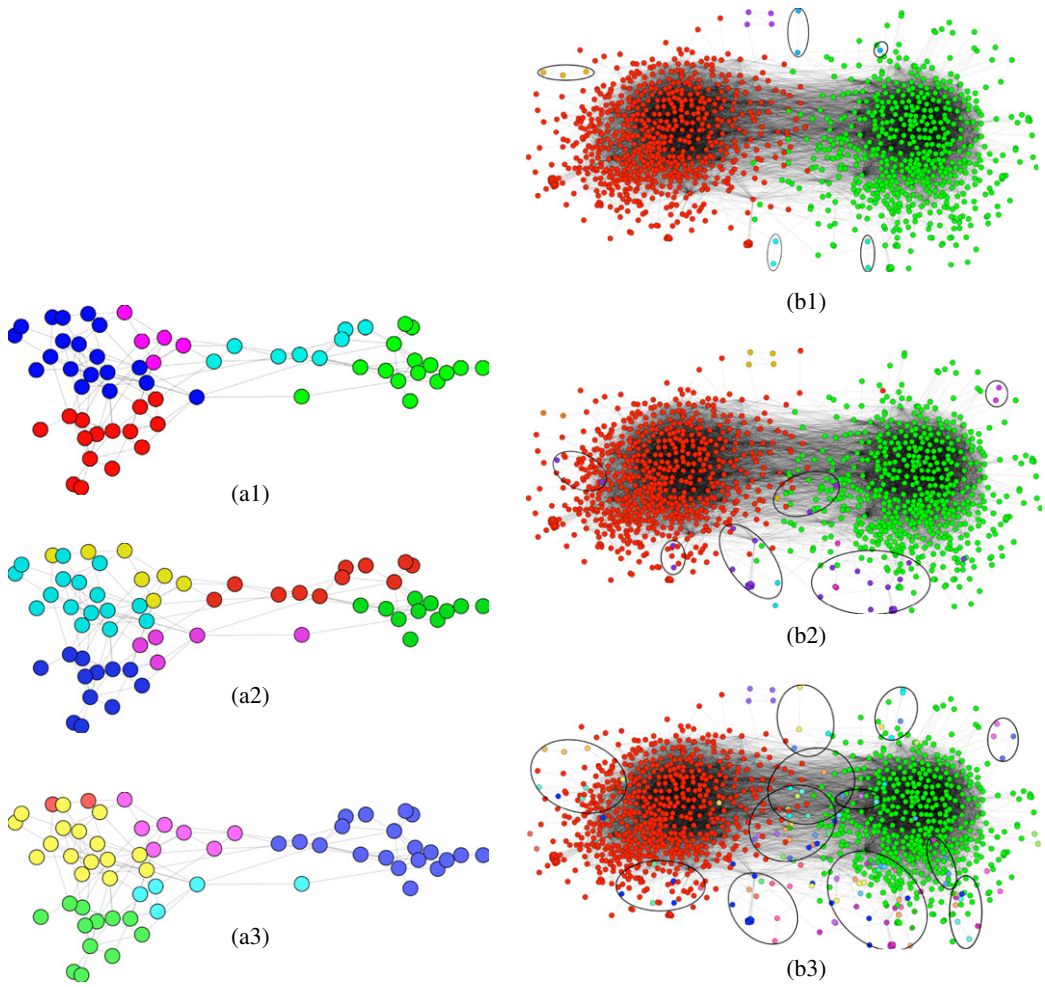
6.2. LFR benchmark analysis

In this section, we employ the comparative analysis proposed in Ref. [50] to evaluate PPC. The paper uses the LFR benchmark [34] to produce benchmark graphs with a ground truth. Afterwards, to evaluate the quality of clustering results according to the ground truth, it exploits *normalized mutual information* (NMI) [20]. The NMI of a clustering X with respect to a ground truth Y is defined as Eq. (6).

$$I_{norm}(X, Y) = \frac{2I(X, Y)}{H(X) + H(Y)}, \tag{6}$$

in which, $I(X, Y)$ is the *mutual information* of the two patitionings X and Y and $H(X)$ is the Shannon entropy of X as defined in Ref. [50]. The higher the $I_{norm}(X, Y)$ is, the more similar clustering X is to the ground truth Y, and hence the more precise it is.

To produce the benchmark graphs, input parameters are considered the same as those of Ref. [50]. For the left subfigures of Fig. 3, which illustrates the NMI plots of the three algorithms, the average degree, the maximum degree, the exponent of the degree distribution, and the exponent of the community size distribution are $20, 50, -2$, and -1, respectively. Furthermore, in each subfigure, four curves are shown that correspond to two different graph sizes (1000 and 5000 vertices) and, for each size, to two different ranges for the community sizes. To maintain a consistent style of naming with Ref. [50], we will indicate community size ranges by letters S and B. S (stands for "small") means that community sizes range between 10 and 50 vertices, and B (stands for "big") means that community sizes range between 20 and 100 vertices. On the other hand, the right side of Fig. 3 represents the results of the algorithms on the LFR benchmarks with 50 000 and 100 000 vertices. Community sizes are considered to be between 20 and 1000 vertices, and the maximum degree is fixed to be 200.



(a) Dolphins graph. (b) PolBlogs graph.

Fig. 2. The visualization of the results of the three algorithms. (1) The dolphins dataset: (1a) PPC clusters. (1b) Louvain clusters. (1c) Infohiermap clusters. (2) The political blogs dataset: (2a) PPC clusters. (2b) Louvain clusters. (2c) Infohiermap clusters.

Furthermore, the LFR benchmark has another input variable, named *mixing parameter* μ, which expresses the ratio between the external degree of a node with respect to its cluster, and the total degree of the node. The lower the mixing parameter, the more obvious the clustering structure and thus the easier it is to identify the correct clustering structure. Hence, most of the algorithms perform well for low values of μ, but fail to identify the underlying clusters when μ grows.

Fig. 3 illustrates the NMI of the three algorithms on the benchmark graphs produced for different values of μ. Each point in the left-hand side curves corresponds to an average over 100 realizations of the benchmarks, while the points of the right curves correspond to a single realization (because of the high running time) of the benchmarks. As Fig. 3 demonstrates, the NMI of PPC seems to drop sharply earlier, compared to that of Louvain and Infohiermap. But if we take a further look at the results, we will identify a nice property that is the evidence of our claim about the superiority of top-down methods. In fact, PPC tends to identify clusters that are significant from the viewpoint of a human observer, and not the clusters that just make sense through heuristic formulations of a cluster.

We compared the results of PPC with those of Louvain and examined the averaged results of the algorithms over 100 realizations of the benchmarks with 1000 and 5000 vertices, and values of μ smaller than 0.5. Among the 32 averaged points, in 28 of the points PPC yielded better or equal NMIs compared to Louvain and in 4 of the points Louvain yielded better NMIs. Moreover, in 24 of the points among 30 non-equal NMIs, the results give very strong evidence ($P < 0.00001$) that PPC performs better, according to the one-sample t-test. Thus, PPC performs better when $\mu < 0.5$. On the other hand, the literature calls communities with $\mu < 0.5$ a community in a strong sense [51]. Thus, PPC performs better when the communities are strongly sensible and significant. Therefore, PPC performs better in a context of strong clusters since it can examine the graph holistically and does not identify very local clusters that are not significant from the viewpoint of a global observer. A similar reasoning is also true about comparing PPC and Infohiermap. As μ approaches one, weak clusters become more probable to be produced, and therefore the observable clustering structure diverges from the ground truth.

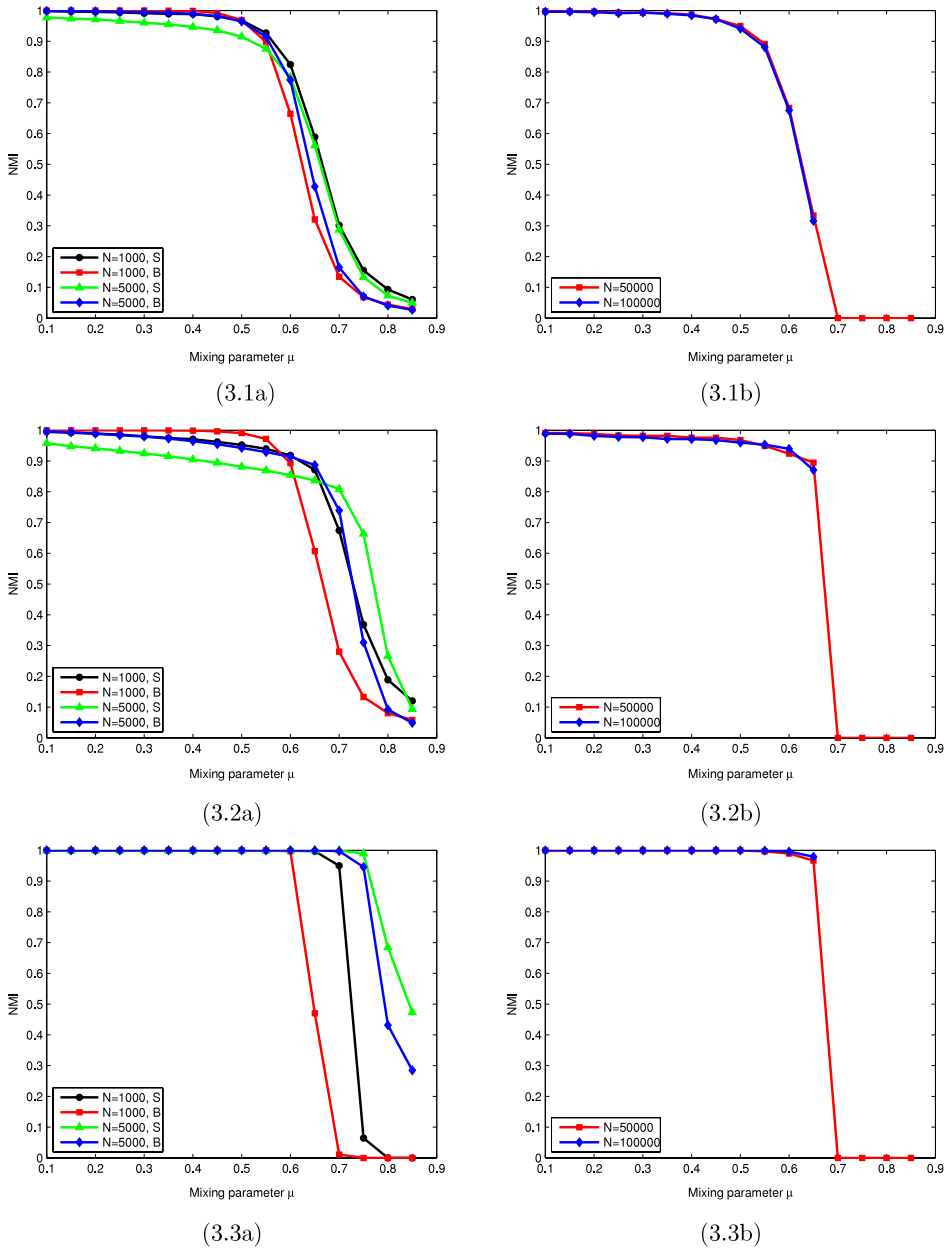


Fig. 3. Tests of the algorithms on the LFR benchmarks, having (a) 1000 and 5000 vertices and two cluster sizes ranges S, B (b) 50 000 and 100 000 vertices and with cluster sizes ranging between 20 and 1000. (a1) & (b1) are the test results for PPC, (a2) & (b2) are the test results for Louvain, and (a3) & (b3) correspond to Infohiermap test results.

So, for low values of μ PPC performs like Infohiermap and better than Louvain. As μ increases to 0.5, and the malformed clusters become more probable to be produced, PPC diverges from the ground truth while Infohiermap is still identifying the ground truth, although it is not significant completely, anymore. Afterwards, as μ increases from 0.5 to 1, the strong clusters gradually diminish, but Louvain and Infohiermap are still detecting them, while PPC tends to find only the meaningful ones, and when μ gets close to one, even weak clusters diminish and all the three algorithms diverge from the ground truth. The right-hand side subfigures of Fig. 3 also infer the very same conclusion.

To support our argument, the result of Louvain algorithm for an LFR graph with $\mu = 0.7$, having 5000 vertices and the small cluster sizes range ($N = 5000, S$), is visualized in Fig. 4. According to the result of the visualization algorithm, Force Atlas 2, it is intuitive that there are not many well-defined clusters in the graph. Besides, the dispersion of the colors corresponding to clusters confirms the claim of low number of meaningful clusters. However, Louvain and Infohiermap produce 38 and 201 clusters respectively, while PPC only identifies only 12 clusters. Similarly, for the LFR graph with $\mu = 0.7$, having 5000 vertices and the big cluster sizes range ($N = 5000, B$), the results are similar and PPC produces

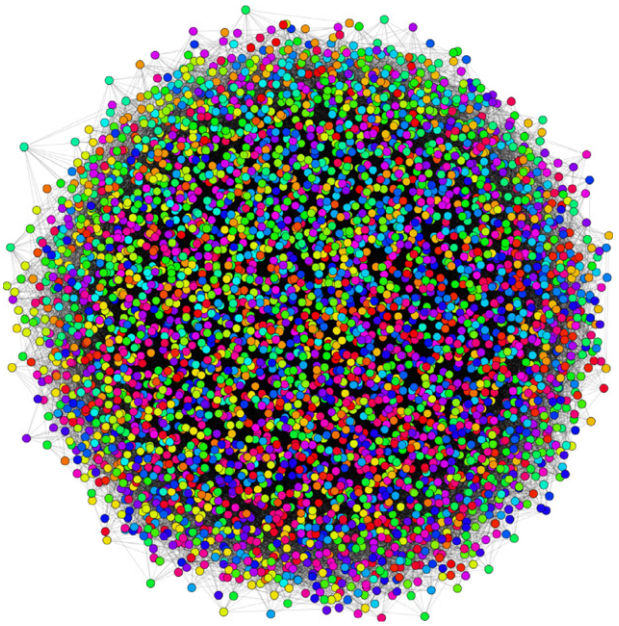


Fig. 4. The result of Louvain algorithm on the LFR benchmark with $\mu = 0.7$, having 5000 vertices and the small cluster sizes range.

Table 3
Statistics about the two subgraphs produced by PPC in the first phase for 100 randomly generated graphs by Ref. [34]. The second column shows statistics about the number of vertices in the smaller subgraph. The third column shows statistics about the percentage of final clusters contained in the subgraph with lower number of final clusters.

	Vertices	Clusters
Min	403	0.414
Max	500	0.500
Avg	475.86	0.472
Stdev	19.27	0.021

only 9 clusters, while Louvain and Infohiermap produce 32 and 101 clusters respectively. This confirms that Louvain and Infohiermap identify many insignificant clusters that just make sense through heuristic formulations and are meaningless from a global observer's perspective.

6.3. Running time analysis

Many of the top-down clustering algorithms suffer from high time complexity since if the partitioning algorithm divides each cluster into a very small cluster and a very large cluster, the time complexity is multiplied by c, the number of final clusters. On the contrary, the experiments of Section 6.1 show that PPC is remarkably fast. In addition to the performance optimizations done in PPC, there is another reason why PPC is fast: PPC tends to detect more coarse-grained clustering structures first.

Table 3 shows some statistics about the two subgraphs produced in the first phase of PPC for 100 randomly generated graphs. Ref. [34], that is used to generate the graph of Fig. 1, is employed with default parameters to generate these 1000-vertex graphs. The second column shows minimum, maximum, average and standard deviation of the number of vertices in smaller subgraph. As is clear, PPC strictly tends to partition the root graph into two balanced subgraphs. The third column, which shows the statistics about the percentage of final clusters contained in the subgraph with lower number of final clusters, also confirms this conclusion. In other words, if the input graph has c clusters with approximately the same characteristics, modified BP partitions it into two subgraphs with almost $c/2$ clusters each, instead of for example a one-cluster subgraph and a subgraph with $c - 1$ clusters.

Balanced partitioning property reduces the multiplier of time complexity from c to almost $\log c$ which can be considered as a constant in practice. It should be noted that this does not force the algorithm to necessarily produce balanced clusters, since in the lower levels some subgraphs could still continue splitting while others do not, although modifications to produce balanced clusters are also possible.

Furthermore, balanced partitioning property is a crucial property that lets PPC produce hierarchies with rational structures. Otherwise, for example it would produce a hierarchy for a network of scientific papers that has a subgraph of a very detailed sub-domain in a special domain and a subgraph of the rest of the network, in its first level. For instance, it would split scientific papers into the string theory related ones and others in the first level, instead of splitting them into the natural sciences papers and the formal sciences ones.

7. Conclusions and future directions

In this paper we proposed the PPC clustering algorithm based on a mixture of random walks and modularity. PPC is a top-down (divisive) method which recursively partitions subgraphs until there is no more modularity gain. At each partitioning phase an ordered list of vertices is obtained by using random walks. Then an appropriate cutting point is determined by modularity and the subgraph is partitioned at the point.

PPC is superior to most of the fast clustering algorithms since it is top-down and it can have a holistic view of the graph. Also, because of the linear time complexity, it is fast enough to be employed for clustering very large real-world networks. In addition, PPC is very flexible because of its top-down approach. For example, changing the algorithm in order to produce balanced clusters or obtain a predefined number of clusters is as easy as pie.

Moreover, because of the plenty of works done in computing PageRank in a distributed setting (e.g. Refs. [52–55]), PPC has the potential to be used distributedly which can make it an ideal solution for clustering at a scale that is infeasible with any single-machine implementation. Thus, distributed PPC remains an important area for future work.

On the other hand, the current version of this algorithm partitions each subgraph into two subgraphs. So in the cases where multi-partitioning is desirable it might not be satisfactory, although this problem can be partially solved by using methods of converting bi-partitioning to multi-way partitioning. A comparison of some of these methods could be found in Ref. [56].

Furthermore, one direction of the future work could be using other measures instead of modularity that are more compatible with the underlying logic of random walks. An example of such measures could be found in Ref. [57]. Moreover, employing measures such as the stability-based modularity generalization proposed in Ref. [24], can help overcome the resolution limit problem of modularity [58]. Besides, taking the recent interest in overlapping community detection into account (e.g. Refs. [21–23]), another obvious direction for future work is extending PPC to identify overlapping clusters.

Acknowledgments

This research was in part supported by ITRC (Research Institute for ICT) and IPM (Nos. CS1390-4-15 and CS1390-4-06) grants and the researchers wish to express their appreciation.

In addition, the authors would like to thank Mohammad Hossein Mousavi Shoushtari for his useful guidelines on the research trend.

References

[1] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, Journal of Statistical Mechanics: Theory and Experiment (10) (2008) P10008.
[2] M. Rosvall, C.T. Bergstrom, Maps of random walks on complex networks reveal community structure, Proceedings of the National Academy of Sciences 105 (4) (2008) 1118–1123.
[3] M. Rosvall, C.T. Bergstrom, Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems, PLoS ONE 6 (4) (2011) e18209.
[4] S. Fortunato, Community detection in graphs, Physics Reports 486 (3–5) (2010) 75–174.
[5] L. Hagen, A.B. Kahng, A new approach to effective circuit clustering, in: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, 1992, pp. 422–427.
[6] D. Harel, Y. Koren, On clustering using random walks, in: Proceedings of the 21st Conference on Foundations of Software Technology and Theoretical Computer Science, 2001, pp. 18–41.
[7] L. Yen, D. Vanvyve, F. Wouters, F. Fouss, M. Verleysen, M. Saerens, Clustering using a random walk based distance measure, in: 13th European Symposium on Artificial Neural Networks, 2005, pp. 317–324.
[8] K. Avrachenkov, V. Dobrynin, D. Nemirovsky, S.K. Pham, E. Smirnova, Pagerank based clustering of hypertext document collections, in: Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2008, pp. 873–874.
[9] A. Azran, Z. Ghahramani, A new approach to data driven clustering, in: Proceedings of the 23rd International Conference on Machine Learning, 2006, pp. 57–64.
[10] R. Andersen, F. Chung, K. Lang, Local graph partitioning using PageRank vectors, in: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, 2006, pp. 475–486.
[11] R. Andersen, F. Chung, K. Lang, Using PageRank to locally partition a graph, Internet Mathematics 4 (1) (2007) 35–64.
[12] R. Andersen, F. Chung, Detecting sharp drops in PageRank and a simplified local partitioning algorithm, in: Proceedings of the 4th International Conference on Theory and Applications of Models of Computation, 2007, pp. 1–12.
[13] M. Alamgir, U. von Luxburg, Multi-agent random walks for local clustering on graphs, in: Proceedings of the IEEE International Conference on Data Mining, 2010, pp. 18–27.
[14] S. Kale, C. Seshadhri, Combinatorial approximation algorithms for MaxCut using random walks, in: Data Structures and Algorithms, 2011, pp. 367–388.
[15] M. Girvan, M.E.J. Newman, Community structure in social and biological networks, Proceedings of the National Academy of Sciences 99 (12) (2002) 7821–7826.
[16] P.G. Sun, Y. Yang, Methods to find community based on edge centrality, Physica A: Statistical Mechanics and its Applications 392 (9) (2013) 1977–1988.
[17] R.M. Karp, Reducibility among combinatorial problems, in: Complexity of Computer Computations, Springer, 1972, pp. 85–103.

[18] K. Li, X. Gong, S. Guan, C.-H. Lai, Efficient algorithm based on neighborhood overlap for community identification in complex networks, Physica A: Statistical Mechanics and its Applications 391 (4) (2012) 1788–1796.
[19] P. Csermely, Weak Links: Stabilizers of Complex Systems from Proteins to Social Networks, Springer, 2006.
[20] L. Danon, A. Daz-Guilera, J. Duch, A. Arenas, Comparing community structure identification, Journal of Statistical Mechanics: Theory and Experiment (09) (2005) P09008.
[21] J. Xie, B.K. Szymanski, X. Liu, SLPA: uncovering overlapping communities in social networks via a speaker–listener interaction dynamic process, in: Proceedings of the IEEE 11th International Conference on Data Mining Workshops, 2011, pp. 344–349.
[22] X. Fu, L. Liu, C. Wang, Detection of community overlap according to belief propagation and conflict, Physica A: Statistical Mechanics and its Applications 392 (4) (2013) 941–952.
[23] T.S. Evans, R. Lambiotte, Line graphs, link partitions, and overlapping communities, Physical Review E 80 (1) (2009) 016105.
[24] R. Lambiotte, J.C. Delvenne, M. Barahona, Laplacian dynamics and multiscale modular structure in networks, 2009. http://arxiv.org/abs/0812.1770.
[25] J.-C. Delvenne, S.N. Yaliraki, M. Barahona, Stability of graph communities across time scales, Proceedings of the National Academy of Sciences 107 (29) (2010) 12755–12760.
[26] L. Page, S. Brin, R. Motwani, T. Winograd, The PageRank Citation Ranking: Bringing Order to the Web, in: Tech. Rep., Stanford Digital Library Technologies Project, 1998.
[27] T.H. Haveliwala, Topic-sensitive PageRank: a context-sensitive ranking algorithm for web search, IEEE Transactions on Knowledge and Data Engineering 15 (4) (2003) 784–796.
[28] H.L. Anderson, Metropolis, Monte Carlo, and the MANIAC, Los Alamos Science 14 (1986) 96–108.
[29] D. Fogaras, B. Rcz, K. Csalogny, T. Sarls, Towards scaling fully personalized PageRank: algorithms, lower bounds, and experiments, Internet Mathematics 2 (3) (2005) 333–358.
[30] M.R. Henzinger, R. Motwani, C. Silverstein, Challenges in web search engines, SIGIR Forum 36 (2) (2002) 11–22.
[31] G. Simmel, K.H. Wolff, The sociology of Georg Simmel, Free Press (1950).
[32] M.E.J. Newman, Modularity and community structure in networks, Proceedings of the National Academy of Sciences 103 (23) (2006) 8577–8582.
[33] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Journal of ACM 34 (3) (1987) 596–615.
[34] A. Lancichinetti, S. Fortunato, F. Radicchi, Benchmark graphs for testing community detection algorithms, Physical Review E 78 (4) (2008) 046110.
[35] M. Bastian, S. Heymann, M. Jacomy, Gephi: an open source software for exploring and manipulating netwoberks, in: Proceedings of the 3rd International Conference on Weblogs and Social Media, 2009, pp. 361–362.
[36] W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33 (4) (1977) 425–473.
[37] D. Lusseau, K. Schneider, O.J. Boisseau, P. Haase, E. Slooten, S.M. Dawson, The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations, Behavioral Ecology and Sociobiology 54 (4) (2003) 396–405.
[38] D.E. Knuth, The Stanford GraphBase: A Platform for Combinatorial Computing, Addison-Wesley, 1993.
[39] V. Krebs, Books about US politics, 1988. http://www.orgnet.com (online; accessed 22.04.12).
[40] M.E.J. Newman, Finding community structure in networks using the eigenvectors of matrices, Physical Review E 74 (3) (2006) 036104.
[41] L.A. Adamic, N. Glance, The political blogosphere and the 2004 U.S. election: divided they blog, in: Proceedings of the 3rd International Workshop on Link Discovery, 2005, pp. 36–43.
[42] I. Xenarios, L. Salwnski, X.J. Duan, P. Higney, S.-M. Kim, D. Eisenberg, DIP, the database of interacting proteins: a research tool for studying cellular networks of protein interactions, Nucleic Acids Research 30 (1) (2002) 303–305.
[43] Cornell KDD Cup, 2003. http://www.cs.cornell.edu/projects/kddcup/ (online; accessed 22.04.12).
[44] A. Harpale, Y. Yang, S. Gopal, D. He, Z. Yue, CiteData: a new multi-faceted dataset for evaluating personalized search performance, in: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, 2010, pp. 549–558.
[45] R. Albert, H. Jeong, A.L. Barabasi, The diameter of the world wide web, Nature 401 (1999) 130–131.
[46] J. Leskovec, K.J. Lang, A. Dasgupta, M.W. Mahoney, Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters, Internet Mathematics 6 (1) (2009) 29–123.
[47] A. Clauset, M.E.J. Newman, C. Moore, Finding community structure in very large networks, Physical Review E 70 (6) (2004) 066111.
[48] P. Pons, M. Latapy, Computing communities in large networks using random walks, Journal of Graph Algorithms and Applications 10 (2) (2006) 191–218.
[49] K. Wakita, T. Tsurumi, Finding community structure in a mega-scale social networking service, in: Proceedings of IADIS International Conference on WWW/Internet, 2007, pp. 153–162.
[50] A. Lancichinetti, S. Fortunato, Community detection algorithms: a comparative analysis, Physical Review E 80 (5) (2009) 056117.
[51] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, D. Parisi, Defining and identifying communities in networks, Proceedings of the National Academy of Sciences 101 (9) (2004) 2658–2663.
[52] Y. Wang, D.J. DeWitt, Computing PageRank in a distributed internet search system, in: Proceedings of the Thirtieth International Conference on Very Large Data Bases, 2004, pp. 420–431.
[53] Y. Zhu, S. Ye, X. Li, Distributed PageRank computation based on iterative aggregation–disaggregation methods, in: Proceedings of the 14th ACM International Conference on Information and Knowledge Management, 2005, pp. 578–585.
[54] H. Ishii, R. Tempo, Distributed randomized algorithms for the PageRank computation, IEEE Transactions on Automatic Control 55 (9) (2010) 1987–2002.
[55] G. Malewicz, M.H. Austern, A.J. Bik, J.C. Dehnert, I. Horn, N. Leiser, G. Czajkowski, Pregel: a system for large-scale graph processing, in: Proceedings of the ACM SIGMOD International Conference on Management of data, 2010, pp. 135–146.
[56] M. Wang, S. Lim, J. Cong, M. Sarrafzadeh, Multi-way partitioning using bi-partition heuristics, in: Proceedings of the Asia and South Pacific Design Automation Conference, 2000, pp. 667–672.
[57] D. Lai, H. Lu, C. Nardini, Enhanced modularity-based community detection by random walk network preprocessing, Physical Review E 81 (6) (2010) 066118.
[58] S. Fortunato, M. Barthlemy, Resolution limit in community detection, Proceedings of the National Academy of Sciences 104 (1) (2007) 36–41.