Fast Algorithm for Distance Dynamics-Based Community Detection

Maram Alsahafy and Lijun Chang(✉)

University of Sydney, Sydney, Australia
mals4485@uni.sydney.edu.au, lijun.chang@sydney.edu.au

Abstract. Community detection is a fundamental problem in graph-based data analytics. Among many models, the distance dynamics model proposed recently is shown to be able to faithfully capture natural communities that are of different sizes. However, the state-of-the-art algorithm Attractor for distance dynamics does not scale to graphs with large maximum vertex degrees, which is the case for large real graphs. In this paper, we aim to scale distance dynamics to large graphs. To achieve that, we propose a fast distance dynamics algorithm FDD. We show that FDD has a worst-case time complexity of $\mathcal{O}(T \cdot \gamma \cdot m)$, where T is the number of iterations until convergence, γ is a small constant, and m is the number of edges in the input graph. Thus, the time complexity of FDD does not depend on the maximum vertex degree. Moreover, we also propose optimization techniques to alleviate the dependency on T. We conduct extensive empirical studies on large real graphs and demonstrate the efficiency and effectiveness of FDD.

Keywords: Community detection · Distance dynamics · Power-law graphs

1 Introduction

Graph representation has been playing an important role in modelling and analyzing the data from real applications such as social networks, communication networks, and information networks. In the graph representation of these applications, community structures naturally exist [8], where entities/vertices in the same community are densely connected and entities/vertices from different communities are sparsely connected. For example, in social networks, users in the same community share similar characteristics or interests.

In view of the importance of community structures, many approaches have been proposed in the literature for identifying communities, *e.g.*, based on betweenness centrality [9], normalized cut [17], or modularity [12]. The betweenness centrality-based approach [9] divides a graph by iteratively removing from the graph the edge that has the largest betweenness centrality (and thus likely to be cross-community edges). The result is a dendrogram that compactly encodes

Z. Huang et al. (Eds.): WISE 2020, LNCS 12342, pp. 213–226, 2020.
https://doi.org/10.1007/978-3-030-62005-9_16

the division process, which is then post-processed based on the modularity measure [13] to identify the best division point. In addition, heuristic algorithms, *e.g.*, the Greedy algorithm [12] and the Louvain algorithm [3], have also been developed to directly optimize the modularity measure which results in an NP-hard optimization problem [4]. In particular, the Louvain algorithm has gained attention recently due to its low time complexity and fast running time. However, it is known that optimizing the modularity measure suffers from the resolution limit [8], *i.e.*, small communities cannot be identified as they are forced to join other communities to maximize the modularity.

The distance dynamics model was recently proposed in [16] to resolve the resolution limit, which does not optimize any specific qualitative measure. Instead, it envisions a graph as an adaptive dynamic system and simulates the interaction among vertices over time, which leads to the discovery of qualified communities. Specifically, each vertex interacts with its neighbors (*i.e.*, adjacent vertices) such that the distances among vertices in the same community tend to decrease while those in different communities increase; thus, the interaction information among vertices transfers through the graph. Finally, the distances will converge to either 0 or 1, and the graph naturally splits into communities by simply removing all edges with distance 1. It is shown in [16] that the distance dynamics model is able to extract large communities as well as small communities. However, the state-of-the-art algorithm Attractor [16] does not scale to graph with large maximum vertex degrees, while large real graphs are usually power-law graphs with large maximum vertex degrees [2]. Note that, Attractor is claimed in [16] to run in approximately $\mathcal{O}(m + k \cdot m + T \cdot m)$ time, where m is the number of edges in the input graph, T is the number of iterations until convergence, and k is the average number of exclusive neighbors for all pairs of adjacent vertices. However, we show in Sect. 3 that this is inaccurate, and in fact the time complexity of Attractor highly depends on deg_{max}—the maximum vertex degree of the input graph—which prevents Attractor from scaling to graphs with large maximum vertex degrees.

In this paper, we design a fast distance dynamics algorithm FDD to scale distance dynamics to large graphs. We first propose efficient techniques to compute the initial distances for all relevant vertex pairs in $\mathcal{O}(deg_{max} \cdot m)$ time, which is worst-case optimal. We then propose two optimization techniques to improve the efficiency of distance updating by observing that (1) converged vertex pairs can be excluded from the computation and (2) not the distances of all relevant vertex pairs need to be computed. Finally, we reduce the total time complexity to $\mathcal{O}(T \cdot \gamma \cdot m)$, where γ is a small constant. As a result, FDD can efficiently process large real graphs that have large maximum vertex degrees. Note that, our optimization techniques also alleviate the dependency on T such that FDD is more likely to run in $\mathcal{O}(\gamma \cdot m)$ time in practice.

Contributions. We summarize our main contributions as follows.

– We analyze the time complexity of the state-of-the-art distance dynamics algorithm Attractor, and show that it highly depends on deg_{max}. (Section 3)

– We design a fast distance dynamics algorithm FDD by proposing efficient initialization, optimization, and time complexity reduction techniques. (Section 4)
– We conduct extensive empirical studies on large real graphs and demonstrate the efficiency and effectiveness of FDD. (Section 5)

Related Works. Besides the methods mentioned above, there are many other community detection methods such as graph partitioning [10,19], hierarchical clustering [15,20], spectral algorithms [6,7], and clique percolation for overlapping communities [14]. A comprehensive survey about community detection can be found in [8].

2 Preliminaries

In this paper, we focus on undirected and unweighted graphs $G = (V, E)$, where V represents the set of vertices and E represents the set of edges. We use n and m to denote the sizes of V and E, respectively. The edge between vertices u and v is denoted by $(u, v) \in E$. The set of neighbors of u is $N(u) = \{v \in V \mid (u, v) \in E\}$, and the degree of u is $deg(u) = |N(u)|$. The closed neighborhood $N[u]$ of u is the union of $\{u\}$ and its neighbors, *i.e.*, $N[u] = \{u\} \cup N(u)$; note that, $deg(u) = |N[u]| - 1$. In the remaining of the paper, we simply refer to closed neighborhood as neighborhood.

2.1 Distance Dynamics

The distance dynamics model is recently proposed in [16]. It consists of two stages: initial distance computation, and distance updating.

State-I: Initial Distance Computation. For each edge $(u, v) \in E$, the initial distance $d^{(0)}(u, v)$ between u and v is computed as

$$d^{(0)}(u, v) = 1 - s^{(0)}(u, v) \tag{1}$$

where $s^{(0)}(u, v)$ is the initial similarity between u and v which is measured in [16] by the Jaccard similarity between their neighborhoods, *i.e.*,

$$s^{(0)}(u, v) = \tfrac{|N[u] \cap N[v]|}{|N[u] \cup N[v]|} \tag{2}$$

Intuitively, the more common neighbors u and v have, the more similar they are and the smaller distance they have. It is easy to see that all the distance and similarity values range between 0 and 1.

Stage-II: Distance Updating. By Eq. (1), if the initial distance $d^{(0)}(u, v)$ is close to 0, then u and v are likely to belong to the same community; if $d^{(0)}(u, v)$ is close to 1, then u and v are likely to belong to different communities. However, if the initial distance is neither close to 0 nor close to 1, then it is hard to tell at the moment whether u and v should belong to the same community or not. To resolve this issue, distance dynamics [16] proposes to iteratively update the distance values for edges (*i.e.*, adjacent vertex pairs) based on the neighborhoods of the two end-points, until convergence (*i.e.*, become 0 or 1). Updating the distance/similarity for edge (u, v) is achieved by considering three forces: *influence from direct link* $DI(u, v)$ *, influence from common neighbors* $CI(u, v)$*, and influence from exclusive neighbors* $EI(u, v)$. Specifically,

$$s^{(t+1)}(u,v) = s^{(t)}(u,v) + DI^{(t+1)}(u,v) + CI^{(t+1)}(u,v) + EI^{(t+1)}(u,v) \tag{3}$$

and

$$d^{(t+1)}(u,v) = 1 - s^{(t+1)}(u,v) \tag{4}$$

where superscript $^{(t)}$ is used to refer to the corresponding values in iteration t.

1) Influence from Direct Link. Firstly, the similarity $s(u, v)$ will increase as a result of the influence of the direct edge between u and v. That is, $DI^{(t+1)}(u, v)$ is computed as

$$DI^{(t+1)}(u,v) = \frac{f\left(s^{(t)}(u,v)\right)}{deg(u)} + \frac{f\left(s^{(t)}(u,v)\right)}{deg(v)} \tag{5}$$

where $f(\cdot)$ is a coupling function and $\sin(\cdot)$ is used in [16].

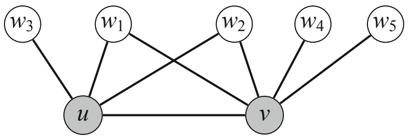


Fig. 1. Distance dynamics

2) Influence from Common Neighbors. Secondly, the similarity $s(u, v)$ will also increase as a result of the influence of the common neighbors of u and v. Let $CN(u, v) = N(u) \cap N(v)$ be the set of common neighbors of u and v; for example, $CN(u, v) = \{w_1, w_2\}$ in Fig. 1. Then, $CI^{(t+1)}(u, v)$ is computed as

$$CI^{(t+1)}(u,v) = \sum_{w \in CN(u,v)} \left(\frac{f\left(s^{(t)}(w,u)\right)}{deg(u)} \times s^{(t)}(w,v) + \frac{f\left(s^{(t)}(w,v)\right)}{deg(v)} \times s^{(t)}(w,u) \right) \tag{6}$$

3) Influence from Exclusive Neighbors. Thirdly, the similarity $s(u, v)$ is also affected by the influence of the **exclusive neighbors** $EN_v(u)$ of u with respect to v, and the exclusive neighbors $EN_u(v)$ of v with respect to u, where $EN_v(u) = N(u)\backslash CN(u, v)$. For example, $EN_v(u) = \{w_3\}$ and $EN_u(v) = \{w_4, w_5\}$ in Fig. 1. However, whether the influence of an exclusive neighbor w on $s(u, v)$ is positive or negative will depend on the similarity between w and the other vertex [16]. Specifically, $EI^{(t+1)}(u, v)$ is computed as

$$EI^{(t+1)}(u,v) = \sum_{w \in EN_v(u)} \left(\frac{f\left(s^{(t)}(w,u)\right)}{deg(u)} \times \rho(w,v) \right) + \sum_{w \in EN_u(v)} \left(\frac{f\left(s^{(t)}(w,v)\right)}{deg(v)} \times \rho(w,u) \right) \tag{7}$$

where $\rho(w,u) = s^{(0)}(w,u)$ if $s^{(0)}(w,u) \geq \lambda$, and $\rho(w,u) = s^{(0)}(w,u) - \lambda$ otherwise; note that, in the latter case, $\rho(w,u) < 0$. Here, λ is a parameter and is recommended in [16] to be in the range $[0.4, 0.6]$.

3 Time Complexity of Attractor

The Attractor algorithm, along with the distance dynamics model, is proposed in [16] for computing the distance dynamics. The pseudocode of Attractor is shown in Algorithm 1, which is self-explanatory and directly follows from the distance dynamics model in Sect. 2. It is worth pointing out that, although only the distances for edges (*i.e.*, adjacent vertex pairs) are required in the community detection and in the initial distance computation, Attractor also computes and stores $\rho(w,u)$ for vertex pairs w and u that are not directly connected but have common neighbors (see Line 2). This is because these $\rho(w,u)$ values will be used later by Eq. (7) for updating the distances.

Algorithm 1: Attractor(G) [16]

```
1 Compute $d^{(0)}(u,v)$ for every edge $(u,v) \in E$;
2 Compute $\rho(w,u)$ for every pair of vertices that are not directly connected but
  have common neighbors;
3 while not converge do
4     for each edge $(u,v) \in E$ do
5         if $0 < d^{(t)}(u,v) < 1$ then
6             Compute $d^{(t+1)}(u,v)$ from $d^{(t)}(u,v)$ by using Equations (3) – (7);
```

Attractor is claimed in [16] to approximately run in $\mathcal{O}(m + k \cdot m + T \cdot m)$ time, where k is the average number of exclusive neighbors for all pairs of adjacent vertices, and T is the total number of iterations (*i.e.*, t). Specifically, it is claimed that Line 1 runs in $\mathcal{O}(m)$ time, Line 2 runs in $\mathcal{O}(k \cdot m)$ time, and Lines 3–6 run in $\mathcal{O}(T \cdot m)$ time. However, we find that this analysis is not accurate, which is also evidenced by its poor performance on graphs with large maximum vertex degrees (see our experiments in Sect. 5).

Issue-1. Firstly, Line 1 computes $d^{(0)}(u,v)$ for all m edges in the graph, which cannot be conduct in $\mathcal{O}(m)$ time by assuming the triangle detection conjecture in [1]. This is because the number of triangles in a graph can be directly obtained from the values $d^{(0)}(u,v)$ of all edges in the graph in linear time (see Sect. 4.1).

Issue-2. Secondly, Line 2 computes $\rho(u,v)$ for $k \cdot m$ non-adjacent pairs of vertices. It is unlikely that this can be conducted in $\mathcal{O}(k \cdot m)$ total time.

Issue-3. Thirdly, an iteration of distance updating (Lines 4–6) may update m edges in the worst case. It is unlikely that this can be achieved in $\mathcal{O}(m)$ time for an iteration.

We illustrate Issue-3 by analyzing the time complexity of an iteration of distance updating (*i.e.*, Lines 4–6). From Eqs. (5), (6), and (7), we can see that updating $d(u,v)$ for a specific edge (u,v) takes $2|CN(u,v)|+|N_v(u)|+|N_u(v)| = deg(u)+deg(v)$ time, by assuming that the values of $\rho(w,u)$ are precomputed and each $\rho(w,u)$ value can be retrieved in constant time. Thus, the time complexity of an iteration of distance updating is $\mathcal{O}\left(\sum_{(u,v)\in E}\left(deg(u)+deg(v)\right)\right) = \mathcal{O}\left(\sum_{v\in V}\left(deg(v)\right)^2\right) = \mathcal{O}\left(deg_{max}\cdot m\right)$, where deg_{max} is the maximum vertex degree in G. Note that, $\mathcal{O}(deg_{max}\cdot m)$ cannot be replaced by $\mathcal{O}(deg_{ave}\cdot m)$, where deg_{ave} is the average vertex degree in G. In fact, $\mathcal{O}(deg_{max}\cdot m)$ can be n times larger than $\mathcal{O}(deg_{ave}\cdot m)$ in extreme cases; for example, in a star graph we have $deg_{max}=m=n-1$ and $\sum_{v\in V}(deg(v))^2=(n-1)\cdot n$, while $deg_{ave}<2$.

Moreover, as we will show in Sect. 4.1, computing $\rho(u,v)$ naively at Line 2 has an even higher time complexity than $\mathcal{O}\left(deg_{max}\cdot m\right)$. As a result, Attractor will not be able to process large real graphs as it is well known that most of the large real graphs, although have small average degrees, are usually power-law graphs with a few vertices of extremely high degrees [2]. This is also confirmed by our empirical studies in Sect. 5.

4 Fast Distance Dynamics

In this section, we design a fast distance dynamics algorithm FDD. We first propose techniques to compute $d^{(0)}(u,v)$ and $\rho(u,v)$ for all relevant vertex pairs in $\mathcal{O}\left(deg_{max}\cdot m\right)$ total time in Sect. 4.1, and then develop optimization techniques to improve the efficiency of distance updating in Sect. 4.2. Finally, we reduce the total time complexity of FDD to $\mathcal{O}(T\cdot\gamma\cdot m)$ for a small constant γ in Sect. 4.3.

4.1 Efficient Initialization

A straightforward approach for initialization (*i.e.*, computing $d^{(0)}(u,v)$ and $\rho(u,v)$ at Lines 1–2 of Algorithm 1) is to compute the values independently for all relevant vertex pairs by conducting an intersection of $N[u]$ and $N[v]$ in $deg(u)+deg(v)$ time. Then, the total time complexity of computing $d^{(0)}(u,v)$ for all edges (*i.e.*, all adjacent vertex pairs) is $\mathcal{O}\left(\sum_{(u,v)\in E}\left(deg(u)+deg(v)\right)\right) = \mathcal{O}\left(deg_{max}\cdot m\right)$. Let E_2 be the set of vertex pairs that are not directly connected but share common neighbors (*i.e.*, E_2 is the set of vertex pairs whose $\rho(u,v)$ values need to be computed), and $deg_2(u)$ be the number of vertex pairs in E_2 containing u (equivalently, the number of 2-hop neighbors of u in the graph G). Then, the total time complexity of computing $\rho(u,v)$ for all $(u,v)\in E_2$ will be $\mathcal{O}\left(\sum_{(u,v)\in E_2}\left(deg(u)+deg(v)\right)\right) = \mathcal{O}\left(\sum_{v\in V}\left(deg(v)\cdot deg_2(v)\right)\right)$. This can be much larger than $\mathcal{O}\left(\sum_{v\in V}deg(v)\cdot deg(v)\right) = \mathcal{O}\left(deg_{max}\cdot m\right)$; consider a tree where every vertex except leafs has exactly x neighbors, we have $deg_2(v)\approx deg(v)\cdot x$.

Algorithm 2: Initialization of FDD

```
1  Initialize an empty hash table $\mathcal{C}$ for storing common neighbor counts;
2  for each vertex $u \in V$ do
3      for each pair of vertices $\{v, w\} \subseteq N(u)$ with $v < w$ do
4          if $(v, w) \notin \mathcal{C}$ then $\mathcal{C}(v, w) \leftarrow 1$;
5          else $\mathcal{C}(v, w) \leftarrow \mathcal{C}(v, w) + 1$;
6  for each edge $(u, v) \in E$ do
7      $s^{(0)}(u, v) \leftarrow \frac{\mathcal{C}(u,v)+2}{deg(u)+deg(v)-\mathcal{C}(u,v)}$;
8      $d^{(0)}(u, v) \leftarrow 1 - s^{(0)}(u, v)$;
9  for each vertex pair $\{u, v\} \in \mathcal{C} \backslash E$ do
10     $s^{(0)}(u, v) \leftarrow \frac{\mathcal{C}(u,v)}{deg(u)+deg(v)-\mathcal{C}(u,v)+2}$;
```

In this subsection, we propose efficient techniques to compute $d^{(0)}(u, v)$ and $\rho(u, v)$ for all relevant vertex pairs (equivalently, compute $s^{(0)}(u, v)$ for all vertex pairs in $E \cup E_2$) in $\mathcal{O}\left(deg_{max} \cdot m\right)$ total time. The general idea is to incrementally count the number of common neighbors for all vertex pairs of $E \cup E_2$ simultaneously. Let $c(u, v)$ be the number of common neighbors of u and v (*i.e.*, $c(u, v) = |N(u) \cap N(v)|$). It is easy to see that we have

$$s^{(0)}(u, v) = \begin{cases} \frac{c(u,v)\,+\,2}{deg(u)\,+\,deg(v) - c(u,v)} & \text{if } (u, v) \in E \\ \frac{c(u,v)}{deg(u)\,+\,deg(v) - c(u,v)\,+\,2} & \text{if } (u, v) \notin E \end{cases} \tag{8}$$

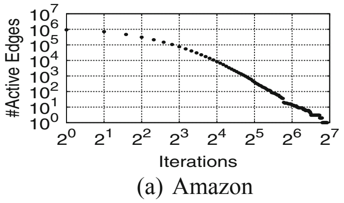
Thus, after computing $c(u, v)$, each $d^{(0)}(u, v)$ and $\rho(u, v)$ can be calculated in constant time. In order to efficiently compute $c(u, v)$, we enumerate all wedges in the graph, where a wedge is a triple (v, u, w) such that $(u, v) \in E$ and $(u, w) \in E$. The set of all wedges can be obtained by enumerating all vertex pairs in the neighborhood of each vertex, and for each wedge (v, u, w), we increase $c(v, w)$ by 1. The pseudocode is shown in Algorithm 2, where a hash table $\mathcal{C}$ is used for efficiently accessing the counts $c(u, v)$.

It is easy to see that the time complexity of Algorithm 2 is $\mathcal{O}\left(\sum_{u \in V} deg^2(u)\right) = \mathcal{O}\left(deg_{max} \cdot m\right)$, by assuming that each access to the hash table $\mathcal{C}$ takes constant time. Note that, this time complexity (for computing $s^{(0)}(u, v)$ for all vertex pairs in $E \cup E_2$) is worst-case optimal. For example, consider a star graph with n vertices, the time complexity is $\mathcal{O}(n^2)$, and $E \cup E_2$ is the set of all vertex pairs (*i.e.*, $|E \cup E_2| = \binom{n}{2} = \frac{n(n-1)}{2}$).

From the above discussions, it can be verified that $\sum_{(u,v) \in E} c(u, v)$ equals the total number of triangles in the graph. That is, the total number of triangles can be obtained from the $s^{(0)}(u, v)$ values of all $(u, v) \in E$ in $\mathcal{O}(m)$ time. As a result, by assuming the triangle detection conjecture in [1], computing $s^{(0)}(u, v)$ (or $d^{(0)}(u, v)$) for all pairs of directed connected vertices cannot be conducted in $\mathcal{O}(m)$ time, which invalidates the claim in [16] regarding the time complexity of Line 1 of Algorithm 1.

Algorithm 3: Distance Updating of FDD

```
1  for each edge $(u, v) \in E$ do
2      if $0 < d^{(0)}(u, v) < 1$ then
3          Push $(u, v)$ into a queue $\mathcal{Q}^{(0)}$;
4  Initialize $t \leftarrow 0$, and a disjoint-set data structure $\mathcal{S}$ for $V$;
5  while $\mathcal{Q}^{(t)} \neq \emptyset$ do
6      while $\mathcal{Q}^{(t)} \neq \emptyset$ do
7          $(u, v) \leftarrow$ pop an edge from $\mathcal{Q}^{(t)}$;
8          if $u$ and $v$ are in different sets in $\mathcal{S}$ then
9              Compute $d^{(t+1)}(u, v)$ from $d^{(t)}(u, v)$ by using Equations (3) – (7);
10             if $0 < d^{(t+1)}(u, v) < 1$ then Push $(u, v)$ into $\mathcal{Q}^{(t+1)}$;
11             if $d^{(t+1)}(u, v) > 1$ then $d^{(t+1)}(u, v) \leftarrow 1$;
12             if $d^{(t+1)}(u, v) < 0$ then Union $u$ and $v$ in $\mathcal{S}$, and $d^{(t+1)}(u, v) \leftarrow 0$;
13         else $d^{(t+1)}(u, v) \leftarrow 0$;
14     $t \leftarrow t + 1$;
```



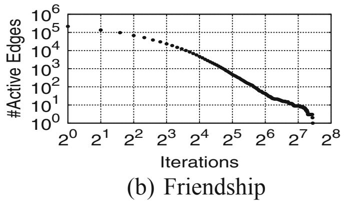


Fig. 2. Number of active edges in iterations

4.2 Optimizing Distance Updating

For distance updating, Attractor blindly tests each edge to update its distance if it is not converged (see Line 4 in Algorithm 1), which is inefficient. We propose two optimization techniques to improve the efficiency of distance updating.

Active-Edge Queue Optimization. We observe that although the total number of iterations (*i.e.*, T) that is needed for all edges to converge could be large (*e.g.*, can be up-to a hundred), most of the edges actually converge after the first few iterations. For example, Fig. 2 demonstrates the number of active (*i.e.*, not converged) edges in the iterations for graphs "Amazon" and "Friendship"; please refer to Sect. 5 for the descriptions of these graphs. Thus, it is a waste of time to check the converged (*i.e.*, non-active) edges. Motivated by this, we propose to maintain a queue for all the active edges, and only check and update the active edges in each iteration. The pseudocode of using queue to maintain active edges is shown in Lines 1–3, 5–7, and 10 of Algorithm 3.

Transitivity Optimization. Our next optimization is based on the following observation. In the distance dynamics model [16], after the distances of all edges converge, the communities are formed by the connected components of the graph after removing all edges of distance 1. Thus, during the iterations of distance dynamics, if the distance between u and v does not yet converge but there is a path between u and v consisting only of edges of distance 0, then u and v must be in the same community in the final result. As a result, we can directly set the distance between u and v to be 0 without following the updating equations. We call this transitivity optimization. To efficiently test whether there is a path between u and v consisting only of edges of distance 0 regarding the current distance values of the graph, we use a disjoint-set data structure [5] to maintain the connected components formed by edges of distance 0. Whenever the distance of an edge (u, v) converges to 0, we union the two corresponding connected components of u and v in the disjoint-set data structure. Note that, according to the distance dynamics model [16], once the distance of an edge becomes 0, it will never increase again. The pseudocode of transitivity optimization is shown in Lines 4, 8, and 12 of Algorithm 3.

4.3 Reducing Time Complexity

Following Sects. 3 and 4.1, the (worst-case) time complexity of FDD is $\mathcal{O}(T \cdot deg_{max} \cdot m)$, where T is the number of iterations of distance updating until convergence. Note that, the two optimization techniques in Sect. 4.2 do not increase nor decrease the time complexity. Specifically, the two operations of the disjoint-set data structure, Find (Line 8 of Algorithm 3) and Union (Line 12 of Algorithm 3), have amortized time complexity of $\mathcal{O}(\alpha(n))$, where $\alpha(\cdot)$ is a extremely slow-growing function that is at most 4 for all practical values of n [5]; thus, we consider them as constant operations in the analysis.

As observed in [2] and also demonstrated in Table 1 in Sect. 5, most real graphs are power-law graphs. That is, although the average degree is very small, the maximum degree deg_{max} could be very large or even in the same order of magnitude as n. Thus, the time complexity $\mathcal{O}(T \cdot deg_{max} \cdot m)$ is still too high for large real graphs that have large deg_{max}. In order to scale FDD to large real graphs, we propose to firstly remove from the graphs all vertices whose degrees are larger than a threshold γ. Consequently, FDD is only run on the resulting graph whose deg_{max} is at most γ, and the time complexity becomes $\mathcal{O}(T \cdot \gamma \cdot m)$. Moreover, as a result of the two optimizations proposed in Sect. 4.2, the term T in the time complexity is also largely alleviated. This is because later iterations take an insignificant amount of time due to the small number of active edges (see Fig. 2). Thus, FDD is more likely to run in $\mathcal{O}(\gamma \cdot m)$ time in practice.

5 Experiments

In this section, we conduct empirical studies to demonstrate the effectiveness and efficiency of our techniques. We evaluate our fast distance dynamics algorithm FDD against two existing algorithms: the existing distance dynamics algorithm Attractor [16], and the popular modularity-based community detection algorithm

Table 1. Statistics of real graphs, where deg_{max} is the maximum degree and $\#Community$ is the number of provided ground-truth communities.

Graphs	n	m	deg_{max}	$\#Community$	Source
Karate	34	78	17	2	Network Repository
Dolphins	62	159	12	2	Network Repository
Polbooks	105	441	25	3	Network Repository
Football	115	613	12	12	Network Repository
Email-Eu	986	16,064	345	42	SNAP
Polblogs	1,224	16,715	351	2	KONECT
AS	23,752	58,416	2,778	176	KONECT
Cora	23,166	89,157	377	70	KONECT
Amazon	334,863	925,872	549	5,000	SNAP
Youtube	1,134,890	2,987,624	28,754	5,130	SNAP
Collaboration	9,875	25,973	65	-	SNAP
Friendship	58,228	214,078	1,134	-	SNAP
RoadNet	1,088,092	1,541,898	9	-	SNAP
Live-Journal	3,997,962	34,681,189	14,815	-	SNAP

Table 2. Running time of FDD against Attractor and Louvain (in seconds)

Graphs	FDD	Attractor	Louvain	Graphs	FDD	Attractor	Louvain
Karate	0.001	0.003	0.001	Cora	1.706	14.949	0.235
Dolphins	0.001	0.004	0.002	Amazon	14.988	93.256	5.508
Polbooks	0.006	0.014	0.002	Youtube	22.667	–	>2hrs
Football	0.006	0.012	0.002	Collaboration	0.414	1.628	0.107
Email-Eu	0.113	5.572	0.012	Friendship	2.738	119.988	0.624
Polblogs	0.110	7.681	0.013	RoadNet	11.975	27.399	18.623
AS	0.346	222.784	0.202	Live-Journal	271.555	–	546.271

Louvain [3]. For FDD and Attractor, we set $\lambda = 0.5$ as suggested in [16]. In addition, we choose $\gamma = 50$ by default for FDD. All algorithms are implemented in C++, and all experiments are conducted on a machine with an Intel Core 2.6 GHz CPU and 8 GB memory.

We evaluate the algorithms on 14 real graphs that are widely used in the existing studies. The graphs are downloaded from Network Repository, Stanford Network Analysis Platform (SNAP), and the Koblenz Network Collection (KONECT). Statistics of these graphs are shown in Table 1. The first ten graphs come with ground-truth communities, while the last four graphs do not have ground-truth communities and are mainly used for efficiency testings. In addition to these real graphs, we also generated synthetic graphs based on the LFR benchmark [11] for evaluating the sensitivity of Attractor and FDD to the maximum degree deg_{max}.

Eval-I: Evaluate the Efficiency of FDD Against Attractor and Louvain. The results are shown in Table 2. We can see that the running time of Attractor generally correlates to the maximum degree deg_{max} and it does not scale to graphs with large deg_{max}, while FDD is not affected by deg_{max} and is much faster than Attractor. For example, Attractor runs out-of-memory on Youtube and Live-Journal where $deg_{max} > 10^4$. On the other two graphs AS and Friendship that have $10^3 < deg_{max} < 10^4$, FDD is 643x and 43.8x faster than Attractor, respectively. When compared with Louvain, the running time of FDD is generally similar to that of Louvain on these graphs, and is smaller on Youtube and Live-Journal; FDD computes the communities for Youtube in 22 s while Louvain does not finish within 2 h. This demonstrates the efficiency-superiority of FDD over existing algorithms Attractor and Louvain.

Table 3. NMI and purity (abbreviated as Pur) of FDD against Attractor and Louvain

Graphs	FDD		Attractor		Louvain		FDD-P	
	NMI	Pur	NMI	Pur	NMI	Pur	NMI	Pur
Karate	**0.782**	**1**	**0.782**	**1**	0.602	0.971	**0.782**	**1**
Dolphins	0.201	0.677	0.201	0.677	**0.516**	**0.968**	0.201	0.677
Polbooks	0.520	**0.886**	0.520	**0.886**	**0.537**	0.848	0.520	**0.886**
Football	**0.931**	**0.939**	**0.931**	**0.939**	0.856	0.800	**0.931**	**0.939**
Email-Eu	**0.726**	**0.824**	0.564	0.585	0.309	0.329	0.707	0.788
Polblogs	0.195	**0.963**	0.201	**0.963**	**0.647**	0.952	0.200	**0.963**
AS	**0.412**	**0.821**	0.362	0.651	0.480	0.509	0.411	0.818
Cora	**0.555**	**0.701**	0.547	0.667	0.459	0.314	0.554	0.700
Amazon	**0.873**	**0.050**	**0.873**	**0.050**	0.794	0.023	**0.873**	**0.050**
Youtube	**0.828**	0.030	–	–	–	–	0.794	**0.031**

Table 4. Running time (in seconds) of our algorithms

Graphs	Attractor	FDD$_1$	FDD$_2$	FDD$_3$	FDD
AS	222.784	33.368	31.093	30.477	0.346
Cora	14.949	4.190	3.568	3.355	1.706
Amazon	93.256	25.717	21.918	19.422	14.988
Youtube	–	–	–	–	22.667
Collaboration	1.628	0.561	0.480	0.442	0.414
Friendship	119.988	24.870	22.889	20.664	2.738
RoadNet	27.399	14.988	12.844	12.242	11.975
Live-Journal	–	–	–	–	271.555

Eval-II: Evaluate the Effectiveness of FDD Against Attractor and Louvain. We run the algorithms on the ten graphs that come with ground-truth communities. We measure two well-known metrics, Normalize Mutual Information (NMI) and Purity [18], for the communities extracted by the algorithms with respect to the ground-truth communities. For both metrics, the larger the value, the better the quality. The results are shown in Table 3, where best results are highlighted by bold font. We can see that the communities extracted by FDD in general has the highest quality regarding both NMI and purity. This confirms the effectiveness of the distance dynamics model [16].

Recall that FDD removes from the graph the vertices that have degrees larger than γ and does not assign these vertices into communities. Nevertheless, the extracted communities actually has a no worse quality than Attractor, as shown in Table 3. One possible reason could be that the high-degree vertices may mislead the community extraction algorithms due to connecting to too many other vertices. In Table 3, we also include the algorithm FDD-P which is FDD with post-processing that assigns the removed vertices to communities; specifically, each removed vertex is assigned to the community that contains most of its neighbors. We can see that the community quality is similar to that of FDD. This suggests that it may be a good idea to first ignore high-degree vertices.

Eval-III: Evaluate Our Techniques. We implemented another three variants of FDD, $\text{FDD}_1, \text{FDD}_2$, FDD_3, by adding the techniques one-by-one. FDD_1 is improved from Attractor by adding the efficient initialization proposed in Sect. 4.1. FDD_2 is improved from FDD_1 by adding the active-edge queue optimization. FDD_3 is improved from FDD_2 by adding the transitivity optimization. Note that, FDD then is the improvement from FDD_3 by adding the technique in Sect. 4.3. The results are shown in Table 4. We can see that each of our techniques contributed to the efficiency of FDD.

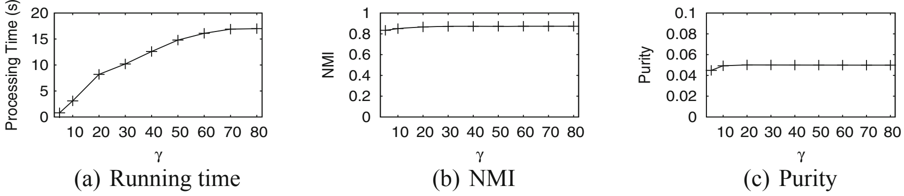


Fig. 3. Varying γ on Amazon

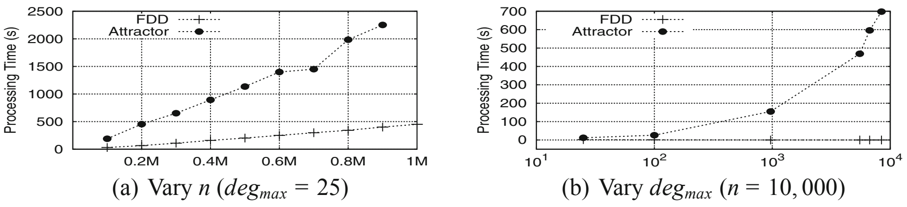


Fig. 4. Running time of FDD and Attractor on LFR graphs

Eval-IV: Evaluate the Effect of γ. We in this testing run FDD on Amazon by varying γ from 5 to 80. The results are shown in Fig. 3. Firstly, when γ increases, the running time of FDD also increases; this conforms with our theoretical analysis of FDD, *i.e.*, its time complexity is $\mathcal{O}(T \cdot \gamma \cdot m)$. Secondly, NMI increases slightly along with the increasing of γ. Thirdly, the purity also increases slightly with respect to γ as more vertices are assigned to communities. Based on these results, we need to make a trade-off between the running time and the quality of extracted communities; we observe that $\gamma = 50$ works well in practice.

Eval-V: Evaluate the Effect of deg_{max}. In order to evaluate the performance of FDD and Attractor more thoroughly on graphs with different maximum degrees (deg_{max}), we also generated synthetic graphs based on the LFR benchmark [11]. We fix the average degree to be 20, and vary either the number of vertices n or the value of deg_{max}. The results of varying n from 0.2×10^6 to 10^6 is shown in Fig. 4(a), where deg_{max} is fixed at 25. We can see that the running time of both FDD and Attractor increases as expected. However, the running time of Attractor increases much faster than that of FDD. The results of varying deg_{max} from 25 to 10^4 is shown in Fig. 4, where n is fixed at 10^4. We can see that, when the maximum degree increases, the running time of Attractor also increases significantly. On the other hand, the running time of our algorithm FDD remains almost the same. This confirms our theoretical analysis in Sects. 3 and 4 that the time complexity of Attractor depends on deg_{max} while the time complexity of FDD is not related to deg_{max}. As large real graphs usually have large maximum degrees, FDD is more suitable than Attractor for processing large real graphs.

6 Conclusion

In this paper, we first showed that the time complexity of the state-of-the-art distance dynamics algorithm Attractor highly depends on the maximum vertex degree, and then developed a fast distance dynamics algorithm FDD to scale distance dynamics to large real graphs that have large maximum vertex degrees. In FDD, we proposed efficient techniques to compute the initial distance for all relevant vertex pairs in $\mathcal{O}(deg_{max} \cdot m)$ time, as well as optimization techniques to improve the practical efficiency of distance updating. Moreover, we also reduced the time complexity to $\mathcal{O}(T \cdot \gamma \cdot m)$ for a small constant γ. Experimental results on large real graphs demonstrated the efficiency and effectiveness of FDD.

References

1. Abboud, A., Williams, V.V.: Popular conjectures imply strong lower bounds for dynamic problems. In: Proceeding of FOCS 2014, pp. 434–443 (2014)
2. Barabasi, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**, 509–512 (1999)
3. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10), 10008 (2008)

4. Brandes, U., et al.: On Modularity-NP-Completeness and Beyond. Universität Fak. für Informatik, Bibliothek (2006)
5. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to algorithms. MIT press, Cambridge (2009)
6. Donetti, L., Munoz, M.A.: Detecting network communities: a new systematic and efficient algorithm. J. Stat. Mech. Theory Exp. **2004**(10), 10012 (2004)
7. Eriksen, K.A., Simonsen, I., Maslov, S., Sneppen, K.: Modularity and extreme edges of the internet. Phys. Rev. Lett. **90**(14), 148701 (2003)
8. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**(3–5), 75–174 (2010)
9. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proc. Int. Acad. Sci. **99**(12), 7821–7826 (2002)
10. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970)
11. Lancichinetti, A., Fortunato, S., Radicchi, F.: Benchmark graphs for testing community detection algorithms. Phys. Rev. E: Stat., Nonlin, Soft Matter Phys. **78**(4), 046110 (2008)
12. Newman, M.E.: Fast algorithm for detecting community structure in networks. Phys. Rev. E **69**(6), 066133 (2004)
13. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**(2), 026113 (2004)
14. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**(7043), 814 (2005)
15. Ravasz, E., Barabási, A.L.: Hierarchical organization in complex networks. Phys. Rev. E **67**(2), 026112 (2003)
16. Shao, J., Han, Z., Yang, Q., Zhou, T.: Community detection based on distance dynamics. In: Proceedings of SIGKDD 2015, pp. 1075–1084 (2015)
17. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **22**(8), 888–905 (2000)
18. Strehl, A., Ghosh, J., Mooney, R.: Impact of similarity measures on web-page clustering. In: Workshop on Artificial Intelligence for Web Search (AAAI 2000), vol. 58, p. 64 (2000)
19. Suaris, P.R., Kedem, G.: An algorithm for quadrisection and its application to standard cell placement. IEEE Trans. Circuits Syst. **35**(3), 294–303 (1988)
20. Wilkinson, D.M., Huberman, B.A.: A method for finding communities of related genes. Proc. Nat. Acad. Sci. **101**(1), 5241–5248 (2004)