big data and cognitive computing

MDPI

Article

InfoFlow: A Distributed Algorithm to Detect Communities According to the Map Equation

Park K. Fung

Department of Ophthalmology, State University of New York Downstate Medical Center, Brooklyn, NY 11203, USA; fylixeoi@gmail.com

Received: 25 April 2019; Accepted: 18 July 2019; Published: 22 July 2019

Abstract: Formidably sized networks are becoming more and more common, including in social sciences, biology, neuroscience, and the technology space. Many network sizes are expected to challenge the storage capability of a single physical computer. Here, we take two approaches to handle big networks: first, we look at how big data technology and distributed computing is an exciting approach to big data storage and processing. Second, most networks can be partitioned or labeled into communities, clusters, or modules, thus capturing the crux of the network while reducing detailed information, through the class of algorithms known as community detection. In this paper, we combine these two approaches, developing a distributed community detection algorithm to handle big networks. In particular, the map equation provides a way to identify network communities according to the information flow between nodes, where InfoMap is a greedy algorithm that uses the map equation. We develop discrete mathematics to adapt InfoMap into a distributed computing framework and then further develop the mathematics for a greedy algorithm, InfoFlow, which has logarithmic time complexity, compared to the linear complexity in InfoMap. Benchmark results of graphs up to millions of nodes and hundreds of millions of edges confirm the time complexity improvement, while maintaining community accuracy. Thus, we develop a map equation based community detection algorithm suitable for big network data processing.

Keywords: graph; community detection; big data

1. Introduction

Formidably sized networks are becoming more and more common, including in social sciences, biology, neuroscience, and the technology space, where the number of nodes and edges may exceed millions or billions. In such cases, the sheer size of the network presents challenges in the processing, visualizing, understanding, or even storing the network [1–3].

When the network data exceeds the memory or disk storage capacity of a single computer, big data technology, including distributed filesystems and distributed processing techniques, can be used to store and process the data [1–3]. The caveat with big data technology is that parallel algorithms have to be designed and implemented in place of the original, serial algorithms [4].

Whilst big data technology provides a means to network storage and processing, for visualization and analytical purposes, smaller sized networks are much favored. Thus, coarse-grained representations of the big networks that preserve important network properties is paramount. For instance, many social systems show homophily in their network representations: nodes with similar properties tend to form highly connected groups called communities, clusters, or modules. Community detection algorithms [5–10] have been an active area of research, with ample algorithms to identify network communities.

Different approaches to community detection exists [5–10]. A more statistically oriented approach is the clustering approach, where sets of points (nodes in the graph; edges information tend not to be used) are categorized into groups called clusters, based on distance or density distributions in the state

space [11]. One popular algorithm is the k-means cluster algorithm, where each point is iteratively assigned to the cluster with the nearest arithmetic mean [11].

When graph edges are considered, community detection algorithms often have an implicit heuristic intra-community, wherein connections are more abundant than inter-community ones. Thus, the most popular approach is based on maximizing some measures of modularity that quantify the ratio of intra- and inter-community edges, relative to a random network, such as the Louvain algorithm [12]. However, such approaches suffer from problems of the "resolution limit", where communities that should be distinct are merged into a bigger community [13].

Another approach is the information theoretic approach, where we interpret the edges of the network as transportation or flow between nodes. The map equation [14] provides an associated information cost for describing the movements within the network, given a community partitioning. If a network has regions in which a PageRank random surfer tends to stay for a long time, minimizing the cost of movement as described by the map equation would identify such regions as communities. Thus, this approach takes advantage of the duality between finding community structure in networks and minimizing the description length of a random surfer's movements on a network. Compared to the modularity approach, which tends to view graph edges as structural connections, and where the detected communities reflect how the network formed, the information theoretic approach sees graph edges as flow, and detects communities reflect network dynamics [7]. In particular, the InfoMap algorithm [15] is a greedy algorithm that uses the map equation to partition a network into communities, performing well on both synthetic benchmarks and real networks [7]. Further, it does not suffer from any resolution limit problems [16]. It is undergoing active research with successful extensions to include those to capture higher-order flow, capturing time-dependent networks [17], overlapping communities [18], and multi-level communities [19].

In this paper, we adapt InfoMap into a distributed algorithm, given its strength in capturing network dynamics, and numerous extension possibilities. Similar projects exist to parallelize InfoMap [20–23], usually involving parallelizing a certain subset of the serial algorithm, with further assumptions of thread-locking or graph locality on top of InfoMap. In this paper, we propose two advancements: first, we develop discrete mathematics to adapt InfoMap into distributed computing framework. This is distinct from other existing works, since the entire algorithm and all data structures are parallelized and distributed, while keeping the algorithm identical to the serial InfoMap. Second, we further develop the mathematics for a greedy algorithm, InfoFlow, which has logarithmic time complexity, compared to the linear complexity in InfoMap. Benchmark results of graphs of up to millions of nodes and hundreds of millions of edges confirm the time complexity improvement, while maintaining community accuracy. Thus, we develop a map equation based community detection algorithm suitable for big network data processing.

This paper is structured as follows: In Section 2, we adapt InfoMap into a distributed computing framework, and develop the InfoFlow distributed algorithm. In Section 3, we perform benchmark and review results. In Section 4, we summarize and discuss future directions.

2. Methodology

In this paper, we build on top of the map equation and InfoMap to found the distributed algorithm InfoFlow, which has improved runtime complexity and can be easily deployed and applied to big datasets. We quickly present the map equation in Section 2.1 for easy reference. Then, we develop the discrete mathematics in Section 2.2, which allow InfoMap to be adapted to distributed computing framework. In Section 2.3, we further develop the discrete math for the InfoFlow algorithm, which has logarithmic time complexity, as compared to the linear time complexity in InfoMap. In Section 3, I perform benchmarking for the two algorithms.

2.1. The Map Equation

For a given network partition, the map Equation [14] specifies the theoretical limit of how concisely we can describe the trajectory of a PageRank random surfer on the network. The underlying code

structure of the map equation is designed such that the description can be compressed if the network has regions in which the random walker tends to stay for a long time.

Given a network with n nodes indexed in Greek alphabets, we first perform PageRank analysis. The edges in the network will be interpreted as the transition probability, so that given an edge from node α to node β, with weight $\omega_{\alpha\beta}$, a PageRank random surfer has probability proportional to the edge weight to transit from node α to node β, not accounting teleportation probability yet. Since the edges signify transition probability, the edge weights are normalized with respect to the outgoing node, so that:

$$\sum_{\alpha} \omega_{\alpha\beta} = 1$$

then, each node will be associated with its ergodic frequency p_α.

The map Equation [14] specifies that, if we partition the network nodes into modules, where each module is indexed with Latin alphabets, then the network will have an information entropy, or codelength, associated with its PageRank random walk:

$$L = \text{plogp}\left(\sum_i q_i\right) - 2\sum_i \text{plogp}(q_i) - \sum_\alpha \text{plogp}(p_\alpha) \tag{1}$$

where:

$$\text{plogp}(x) = x \log_2 x \tag{2}$$

p_i is the ergodic frequency of the module. This is simply the sum of the ergodic frequencies of the nodes within the module:

$$p_i = \sum_{\alpha \in i} p_\alpha \tag{3}$$

and q_i is the probability of exiting the module, accounting for PageRank teleportation:

$$q_i = \tau \frac{n - n_i}{n-1} p_i + (1-\tau) \sum_{\alpha \in i} \sum_{\beta \notin i} p_\alpha \omega_{\alpha\beta} \tag{4}$$

n_i being the number of nodes within module i, and τ is the probability of PageRank teleportation.

2.2. InfoMap

Here, we develop mathematics to construct p_i and q_i, the ergodic frequency, and the exit probability of each module, thereby allowing calculation of Equation (1). In particular, the InfoMap algorithm [15] starts by having each node being its own module, and then in each iteration, merge two modules into one to reduce codelength. We develop maths to construct p_i and q_i, by providing formulae for the quantities in the merged module based on those in the two modules to be merged.

We can rewrite Equation (4) as:

$$q_i = \tau \frac{n - n_i}{n-1} p_i + (1-\tau) w_i \tag{5}$$

with:

$$w_i = \sum_{\alpha \in i} \sum_{\beta \notin i} p_\alpha \omega_{\alpha\beta} \tag{6}$$

being the exit probability without teleportation.

We can define a similar quantity, the transition probability without teleportation from module j to module k:

$$w_{jk} = \sum_{\alpha \in j} \sum_{\beta \in k} p_\alpha \omega_{\alpha\beta} \tag{7}$$

Now, if we merge modules j and k into a new module with index i, the exit probability would be follow Equation (5) with:

$$n_i = n_j + n_k \tag{8}$$

$$p_i = p_j + p_k \tag{9}$$

and the exit probability without teleportation can be calculated via:

$$w_i = \sum_{\alpha \in i} \sum_{\beta \notin i} p_\alpha \omega_{\alpha\beta} \tag{10}$$

$$= \sum_{\alpha \in i \text{ or } \alpha \in k} \sum_{\beta \notin i \text{ and } \beta \notin k} p_\alpha \omega_{\alpha\beta} \tag{11}$$

$$= \sum_{\alpha \in j} \sum_{\beta \notin i \text{ and } \beta \notin k} p_\alpha \omega_{\alpha\beta} + \sum_{\alpha \in k} \sum_{\beta \notin i \text{ and } \beta \notin k} p_\alpha \omega_{\alpha\beta} \tag{12}$$

since we are looking at the exit probability of a module, there are no self-connections within modules, so that the specification of $p_\alpha\ \omega_{\alpha\beta}$ given $\alpha \in i, \beta \notin i$ is redundant. Then we have:

$$\omega_i = \sum_{\alpha \in j} \sum_{\beta \notin k} p_\alpha \omega_{\alpha\beta} + \sum_{\alpha \in k} \sum_{\beta \notin j} p_\alpha \omega_{\alpha\beta} \tag{13}$$

which conforms with intuition, that the exit probability without teleportation of the new module is equal to the exit probability of all nodes without counting for the connections from j to k, or from k to j.

We can further simplify the math's by expanding the non-inclusive set specification:

$$\omega_i = \sum_{\alpha \in j} \left[\sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{\beta \in k} p_\alpha \omega_{\alpha\beta} \right] + \sum_{\alpha \in k} \left[\sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{\beta \in j} p_\alpha \omega_{\alpha\beta} \right] \tag{14}$$

Expanding gives:

$$\omega_i = \sum_{\alpha \in j} \sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{\alpha \in j} \sum_{\beta \in k} p_\alpha \omega_{\alpha\beta} + \sum_{\alpha \in k} \sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{\alpha \in k} \sum_{\beta \in j} p_\alpha \omega_{\alpha\beta} \tag{15}$$

which by definition is:

$$\omega_i = \omega_j - \omega_{jk} + \omega_k + \omega_{kj} \tag{16}$$

So that now, we can calculate ω_i and by Equation (5) we can calculate q_i.

We can do similar for ω_{il}, if we merged modules j and k into i, and l is some other module:

$$w_{il} = \sum_{\alpha \in i} \sum_{\beta \in l} p_\alpha \omega_{\alpha\beta} \tag{17}$$

$$= \sum_{\alpha \in j \text{ or } \alpha \epsilon k} \sum_{\beta \in l} p_\alpha \omega_{\alpha\beta} \tag{18}$$

$$= \sum_{\alpha \in j} \sum_{\beta \in l} p_\alpha \omega_{\alpha\beta} + \sum_{\alpha \in k} \sum_{\beta \in l} p_\alpha \omega_{\alpha\beta} \tag{19}$$

$$= \omega_{jl} + \omega_{kl} \tag{20}$$

and similarly for ω_{li}:

$$\omega_{li} = \omega_{lj} + \omega_{lk} \tag{21}$$

With these calculations, we are able to construct the modular properties after each pair-wise merging, where the relevant properties include n_i, p_i, w_i, and w_{ij}. We can forget about the actual nodal properties; after each merge, we only need to keep track of modular properties.

Now, we can implement the InfoMap algorithm, where initially each node is its own module, and then in each iteration, we merge the two modules that offer the greatest reduction in codelength, with the new module having modular properties according to Equations (8), (9), (16), (5), (20), and (21). The algorithm terminates when no more merges are possible to reduce codelength. Since the maximum number of merges is $e - 1$ merges, where e is the number of edges in the network, the number of merges have complexity $O(e)$. A graphical illustration of the InfoMap algorithm is shown in Figure 1.

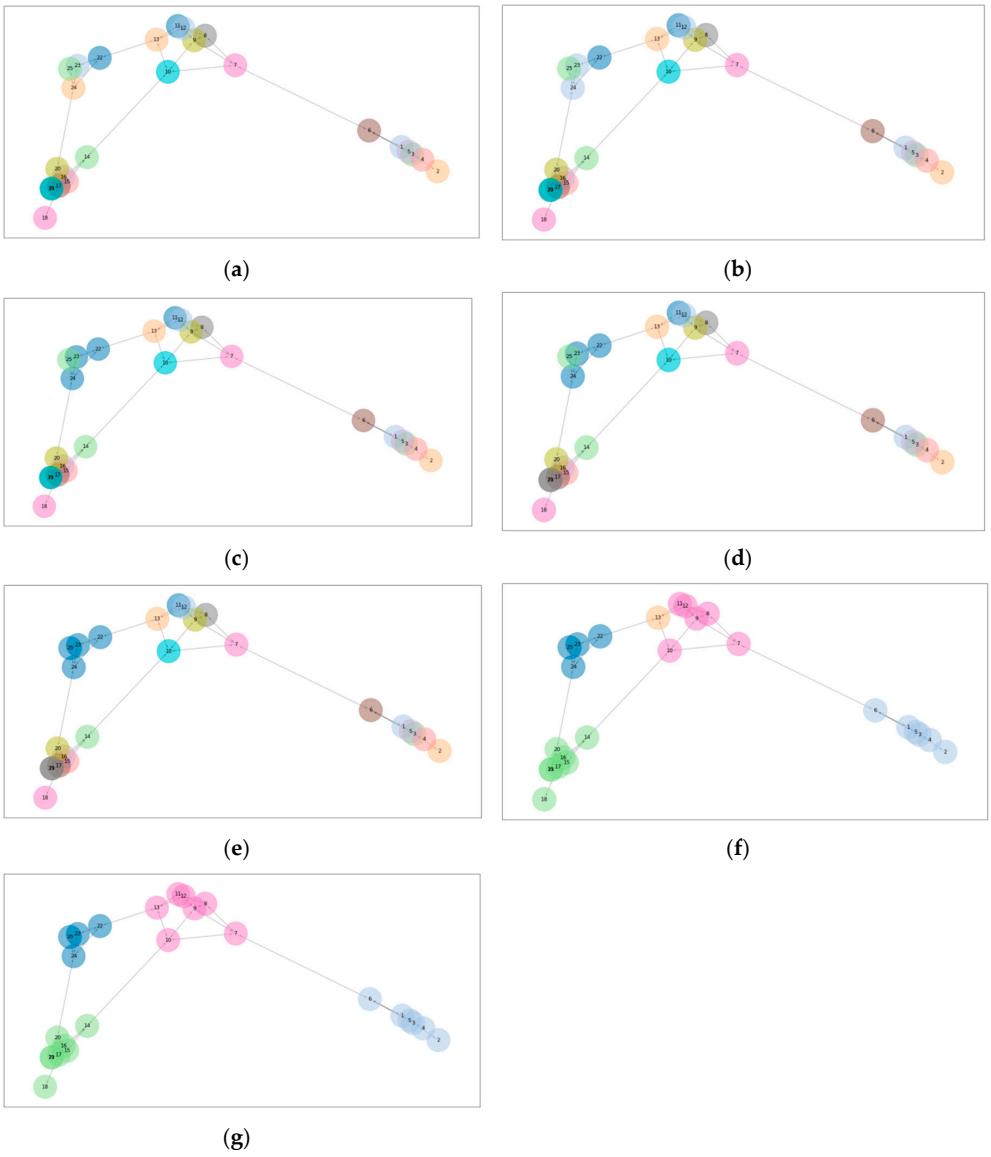

Figure 1. Graph with 25 nodes as visual illustration of InfoMap algorithm. Each node is labeled with a node number for reference and is colored according to its assigned community. We show the first four merges, and last two merges according to the InfoMap algorithm. In each step, two communities are merged into one. There are 21 merges in total. For brevity, only the first four and last two merges are shown. (**a**) In the initial graph, each node is its own module; (**b**) nodes 23 and 24 are merged; (**c**) node 22 is merged with module 23–24; (**d**) nodes 17 and 19 are merged; (**e**) node 25 is merged with module 22–23–24; (**f**,**g**) for brevity, we skip to the last two merges, where we are left with a final partitioning of four communities.

2.3. InfoFlow

InfoMap merges two modules on each iteration, giving $O(e)$ loops. One obvious improvement possibility is to perform multiple merges per loop. However, this idea is not compatible with the idea of performing pair-wise merges, unless we can make sure no module is involved with more than one merge at once.

Here, rather than focusing on making sure that no module is involved with more than one merge at once, we can explore the idea of merging multiple modules at once. Thus, we can perform parallel merges in the same loop iteration, where possibly all modules are involved in some merge.

Consider multiple modules $\sim M_i$ merging into a module M. Another way to express this equivalently is to say that a module M is partitioned into i non-overlapping subsets:

$$M = \sum_i M_i \tag{22}$$

Then we can expand the nodal sum over module M into the sum over all nodes in all submodules M_i, the exit probability of the merged module M becomes:

$$\omega_M = \sum_{M_i} \sum_{\alpha \in M_i} \sum_{\beta \notin M} p_{\alpha\omega_{\alpha\beta}} \tag{23}$$

$$= \sum_{M_i} \sum_{\alpha \in M_i} \left[\sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{\beta \in M} p_{\alpha\omega_{\alpha\beta}} \right] \tag{24}$$

$$= \sum_{M_i} \sum_{\alpha \in M_i} \sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{M_i} \sum_{\alpha \in M_i} \sum_{\beta \in M_i} p_{\alpha\omega_{\alpha\beta}} \tag{25}$$

$$= \sum_{M_i} \sum_{\alpha \in M_i} \sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{M_i} \sum_{\alpha \in M_i} \sum_{M_j} \sum_{\beta \in M_j} p_{\alpha\omega_{\alpha\beta}} \tag{26}$$

where we expand the second term with respect to the M_j's to give:

$$\omega_M = \sum_{M_i} \sum_{\alpha \in M_i} \sum_{\beta} p_{\alpha\omega_{\alpha\beta}} - \sum_{M_i} \sum_{\alpha \in M_i} \sum_{M_j \neq M_i} \sum_{\beta \in M_j} p_{\alpha\omega_{\alpha\beta}} - \sum_{M_i} \sum_{\alpha \in M_i} \sum_{\beta \in M_{ii}} p_{\alpha\omega_{\alpha\beta}} \tag{27}$$

Combining the first and third terms,

$$\omega_M = \sum_{M_i} \sum_{\alpha \in M_i} \sum_{\beta \notin M_i} p_{\alpha\omega_{\alpha\beta}} - \sum_{M_i} \sum_{\alpha \in M_i} \sum_{M_j \neq M_i} \sum_{\beta \in M_j} p_{\alpha\omega_{\alpha\beta}} \tag{28}$$

which we can recognize as:

$$\omega_M = \sum_{M_i} \omega_{M_i} - \sum_{M_i} \sum_{M_j \neq M_i} \omega_{M_i M_j} \tag{29}$$

$$\omega_{M_i} = \sum_{\alpha \in M_i} \sum_{\beta \notin M_i} p_\alpha \omega_{\alpha\beta} \tag{30}$$

$$\omega_{M_i M_j} = \sum_{\alpha \in M_i} \sum_{\beta \in M_j} p_\alpha \omega_{\alpha\beta} \tag{31}$$

where we can immediately see that Equation (29) is a linear generalization of Equation (16), while Equations (30) and (31) are identical to previous definitions, and may be calculated iteratively as the previous algorithm. We can calculate $\omega_{M_iM_j}$ by expanding on the partitioning:

$$\omega_{M_iM_j} = \sum_{M_k \in M_i} \sum_{\alpha \in M_k} \sum_{M_{k'} \in M_j} \sum_{\beta \in M_{k'}} p_\alpha \omega_{\alpha\beta} \tag{32}$$

$$= \sum_{M_k \in M_i} \sum_{M_{k'} \in M_j} \omega_{M_kM_{k'}} \tag{33}$$

so that when we merge a number of modules together, we can calculate its connections with other modules by aggregating the existing modular connections. This is directly analogous to Equation (20).

Thus, the mathematical properties of merging multiple modules into one are identical to that of merging two modules. This is key to developing my multi-merge algorithm, InfoFlow.

As InfoMap, each node is initially its own module. During each iteration, each module seeks to merge with a connected module that offers the greatest reduction in codelength, if only the merging of the two modules are considered. If no such merge exists, the module does not seek to merge. Then, the weakly connected modular components connected via the merge seeking are merged into one module, according to Equations (29) and (33). This is repeated seek-merging and bulk-merging is iterated until the codelength cannot be reduced. A graphical illustration of InfoFlow is shown in Figure 2.

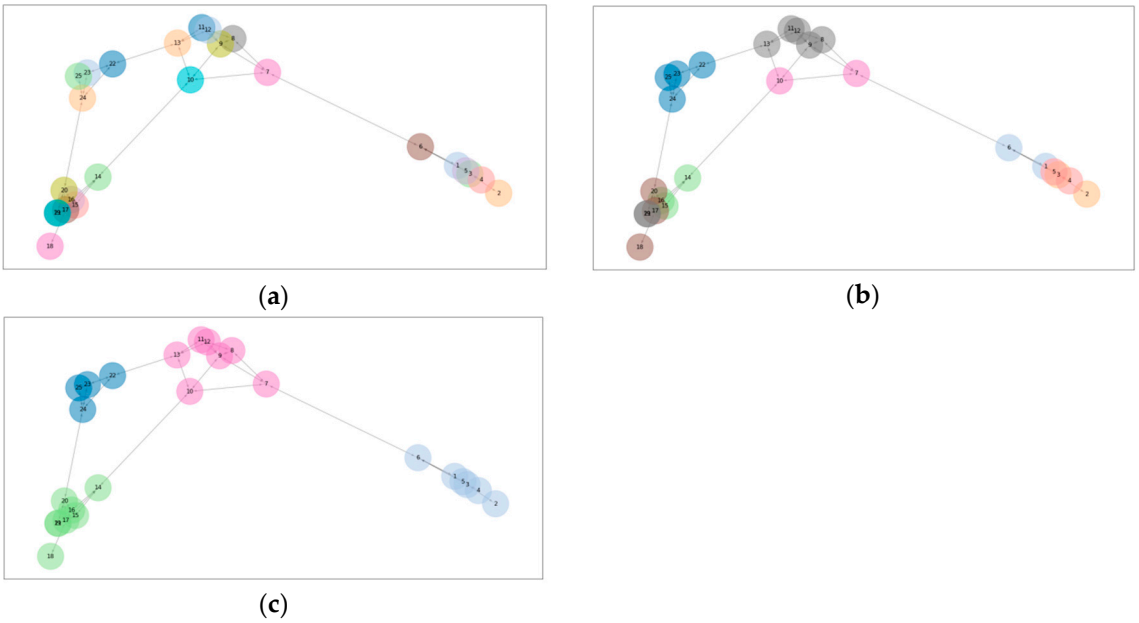

Figure 2. The same 25-node graph, showing all merges according to the InfoFlow algorithm. There are totally 2 loops, compared to 21 in InfoMap. (**a**) The initial graph is shown for easy reference; (**b**) after one loop, the 25 communities are merged into 9, which is roughly the geometric mean between 25 and 4; (**c**) The final partitioning is identical to that of InfoMap.

Next, n nodes are partitioned into m final modules according to InfoFlow. If we assume in each loop, k modules merge into one on average, and that there are l loop, we have:

$$n\, k^{-l} = m \tag{34}$$

$$k^l = \frac{n}{m} \tag{35}$$

$$l = \log_k n - \log_k m \tag{36}$$

so that we have $O(\log_k n)$ merges, while within each merge, there is $O(k)$ time complexity related to the connected component with $O(k)$ modules. Thus, the overall average time complexity is $O(k \log_k n)$.

The worst case complexity comes in two cases: first, when we degenerate into InfoMap, i.e., we have $O(e)$ loops, each loop we merge only two modules into one; second when $l = 1$ and $k = n/m$, and the overall complexity is $O(k) = O(n/m)$.

3. Simulation and Results

InfoMap and InfoFlow are implemented on Apache Spark [24] using the Scala language. The code, which is open source and can be accessed online [25] is set up and run on a local desktop computer, the Dell Precision T3610, running Windows 7 Professional. For easy referencing, the specifications for the computer is Intel Xeon CPU E5-1620 v2 @3.70 GHz 3.70 GHz, with 64 Gb RAM. Apache Spark 2.1.1 and Hadoop 2.7 are used. When possible, RAM disk was used to speed up simulations.

First, we ran real world data of small to moderately sized graphs from [26–28], with both InfoMap and InfoFlow, and compared the runtime performance and resultant communities. The results are tabulated in Table 1, listing the nodes and edges of the graph, the number of loops ran before the algorithms completed, the runtime (for the sake of comparison, common runtime between the two algorithms, including initialization time and PageRank runtime, are not counted), and final partitioning codelength. The normalized mutual information between the community partitioning's given by the two algorithms is also tabulated.

Table 1. Benchmarking results for InfoMap and InfoFlow, for datasets of various size, on the Dell Precision T3610. To aid comparison we do not include common performance runtime, including file reading time and PageRank runtime from the table. Runtime in unit of seconds. In these test cases, while runtime complexity is linear in the case of InfoMap, it is pseudo-constant in InfoFlow, while keeping very high accuracy in the final partitioning, according to normalized mutual information (NMI). The datasets are real world graphs taken from [26–28].

Nodes	Edges	InfoMap			InfoFlow			Normalized Mutual Information
		Loops	Time	Codelength	Loops	Time	Codelength	
396	994	90	7	3.449	2	3	3.875	94%
674	613	325	16	3.380	2	3	3.408	99%
1059	4919	568	53	5.617	2	6	6.206	86%
1490	19,090	1006	86	7.786	3	6	8.044	85%
2114	2277	1082	72	4.532	2	4	4.862	95%
3084	10,413	1763	193	6.041	3	8	6.748	87%
4470	12,731	1915	232	4.780	3	7	5.042	95%
4941	13,188	4449	423	6.953	3	7	7.204	90%
6752	54,233	4661	1057	7.291	4	10	7.811	85%
7343	11,898	4133	422	4.897	3	8	5.011	100%
8843	41,601	4741	997	6.285	4	13	6.752	92%
10,617	72,168	4571	1194	7.206	3	9	7.354	95%
13,308	148,035	10,298	3623	9.013	1	10	9.395	82%
13,356	120,238	12,642	3492	9.968	3	11	10.78	76%
23,219	325,593	7803	7120	8.231	3	16	8.419	95%
27,770	352,807	25,166	12,352	9.823	4	25	10.47	74%

We see that, for InfoMap, the number of loops and runtime follow a linear complexity relationship with the size of the graph, while the number of loops in InfoFlow is kept within 4, and the runtime within 30 s, so that a pseudo-constant, or logarithmic complexity relationship with the size of the graph, along the estimation of Equation (36), is confirmed. As a solid example of the speedup in InfoFlow, for the graph with 27,770 nodes and 352,807 edges, there are 1000 times less loops in InfoFlow, and the runtime is nearly 3.5 h for InfoMap, and 25 s for InfoFlow, representing nearly a 500-time speedup.

As analyzed in Equations (34)–(36), this speedup in runtime and complexity is a direct consequence of going from pair-wise merge in InfoMap to the multi-merging in InfoFlow. In terms of partitioning accuracy, it might be a concern whether the bulk-merging of InfoFlow might sacrifice partitioning accuracy. From the results of Table 1, we see that the codelength difference between the two algorithms are very similar, rarely exceeding a 5% difference. The normalized mutual information (NMI) is often kept higher than 80%, so the accuracy in community detection is not compromised when going from InfoMap to InfoFlow.

Having compared the runtime and accuracy between InfoMap and InfoFlow, we now apply InfoFlow to bigger data, with graphs going up to millions of nodes and hundreds of millions of edges [26–28], until the limit of the computing resource is challenged on the Dell Precision T3610. The results of the simulations are tabulated in Table 2. We see that the number of loops is kept within 20, so a pseudo-constant or logarithmic complexity is well followed, while the runtime is kept within a few hours. The runtime does not follow any obvious relationship with the number of nodes or edges, since the processing of the dataset challenges the limit of the computer, and complex performance issues with memory caching and paging comes into play. Importantly, the runtime is in the same order of magnitude with PageRank calculation time. Since the latter is a prerequisite for the map equation approach, the benchmarking results indicate we have optimal runtime complexity within the map equation approach to community detection.

Table 2. Benchmarking results for InfoFlow, on the Dell Precision T3610, for datasets that challenge the limit the computer resource. The runtime for graph reading and PageRank calculations are listed to add perspective on community detection speed. Importantly, the number of loops are very small, indicating logarithmic complexity, and that the community detection runtime is on the same order of magnitude as PageRank. Since the latter is a prerequisite for the map equation approach to community detection, InfoFlow may have optimal runtime complexity for this class of algorithm. Runtime unit in seconds. The datasets are real world graphs taken from [26–28].

Nodes	Edges	Read Time	PageRank Time	InfoFlow	
				Loops	Time
50,515	819,306	6	33	7	63
82,670	133,445	3	36	4	68
325,729	1,497,134	9	138	8	420
281,903	2,312,497	20	160	5	1269
685,230	1,600,595	51	449	7	2618
1,632,803	30,622,564	239	1298	19	3816
3,744,768	16,518,948	139	497	11	3134
4,847,571	68,993,773	635	9531	20	21,540
9,845,725	57,156,537	574	9531	7	70,379
98,303	100,245,742	452	1183	1	1437
5,154,859	99,199,551	2079	13,531	8	9276

4. Conclusions

With a view of developing a distributed community detection algorithm, we developed discrete mathematics on the map equation to provide formulae for the modular properties for merged pairwise modules, which enabled the implementation of InfoMap algorithm on distributed computing. We then generalized this from a pairwise merge to merging arbitrary number of modules, which prompted the algorithm InfoFlow. Benchmarking results on an Apache Spark implementation confirmed that the runtime complexity of InfoFlow had logarithmic runtime complexity, compared to the linear time complexity of InfoMap, while retaining accuracy in the community results.

Similar projects to develop distributed community detection algorithms, in particular InfoMap, exist [20–23]. These projects parallelize certain segments of the algorithm while keeping other segments and data structures in serial, with assumptions on thread locking or graph locality. In contrast, in this

paper, we developed discrete math to adapt InfoMap into distributed computing framework, while keeping the algorithm identical, with no additional assumptions being made. In other words, we developed the mathematical formulation that enables parallel and distributed computing, rather than developing an inherently parallel algorithm. This is a significant development in InfoMap. InfoFlow was built with only one additional development of multi-merging, as opposed to the pair-wise merging in InfoMap. Benchmark results showed that this development improves runtime complexity while retaining result accuracy. Thus, the mathematics is a significant contribution to the research in InfoMap, which will be valuable future research in InfoMap extensions such as hierarchical structures, overlapping structures, and higher-order Markov dynamics [17–19].

The coding implementation [25] is open source and implemented in Apache Spark, which is actively maintained, with proven performance, reliability and scalability, with contributions from companies such as Google, Facebook, and IBM [29]. It can be easily configured and deployed on clusters and cloud platforms. This is in contrast to, for example, the implementation in [23], which used the GraphLab PowerGraph library [30] which was not actively maintained. Another example is [20], which used the Thrill library [31], which is still in the experimental phase. The choice of distributed computing library framework, along with computational environment, is one of the major factors affecting runtime, memory consumption and other performance metrics. Meanwhile, algorithmic performance, which is agnostic to the coding implementation and library environment, may be measured via theoretical space and time complexity, where the logarithmic runtime complexity of InfoFlow shines.

Funding: This research received no external funding.

Acknowledgments: The author thank Meena Rajani and Martin Rosvall for fruitful discussions.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, C.L.P.; Zhang, C.Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Inf. Sci.* **2014**, *275*, 314–347. [CrossRef]
2. Oussous, A.; Benjelloun, F.Z.; Lahcen, A.A.; Belfkih, S. Big Data technologies: A survey. *J. King Saudi Univ. Comput. Inf. Sci.* **2018**, *30*, 431–448. [CrossRef]
3. Hashem, I.; Yaqoob, I.; Mokhtar, Y.A.S.; Gani, N.A.; Khan, S. The rise of "big data" on cloud computing: Review and open research issues. *Inf. Syst.* **2015**, *47*, 98–115. [CrossRef]
4. Hwu, W.M. What is ahead for parallel computing. *J. Parallel Distrib. Comput.* **2014**, *74*, 2574–2581. [CrossRef]
5. Girvan, M.; Newman, M.E.J. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA* **2002**, *99*, 7821–7826. [CrossRef] [PubMed]
6. Fortunato, S. Community detection in graphs. *Phys. Rep.* **2010**, *486*, 75–174. [CrossRef]
7. Lancichinetti, A.; Fortunato, S. Community detection algorithms: A comparative analysis. *Phys. Rev. E* **2009**, *80*, 056117. [CrossRef] [PubMed]
8. Aldecoa, R.; Marin, I. Exploring the limits of community detection strategies in complex networks. *Sci. Rep.* **2013**, *3*, 2216. [CrossRef]
9. Fortunato, S.; Hric, D. Community detection in networks: A user guide. *Phys. Rep.* **2016**, *659*, 1–44. [CrossRef]
10. Javed, M.A.; Younis, M.S.; Latif, S.; Qadir, J.; Baig, A. Community detection in networks: A multidisciplinary review. *J. Netw. Comput. Appl.* **2018**, *108*, 87–111. [CrossRef]
11. Saxena, A.; Prasad, M.; Gupta, A.; Bharill, N.; Patel, O.P.; Tiwari, A.; Joo, E.M.; Weiping, D.; Chin-Teng, L. A review of clustering techniques and developments. *Neurocomputing* **2017**, *267*, 664–681. [CrossRef]
12. Newman, M.E.J.; Girvan, M. Finding and evaluating community structure in networks. *Phys. Rev. E* **2004**, *69*, 026113. [CrossRef] [PubMed]
13. Fortunato, S.; Barthelemy, M. Resolution limit in community detection. *Proc. Natl. Acad. Sci. USA* **2007**, *104*, 36–41. [CrossRef] [PubMed]

14. Rosvall, M.; Axelsson, D.; Bergstrom, C.T. The map equation. *Eur. Phys. J. Spec. Top.* **2009**, *178*, 13–23. [CrossRef]
15. Rosvall, M.; Bergstrom, C.T. Maps of random walks on complex networks reveal community structure. *Proc. Natl. Acad. Sci. USA* **2008**, *105*, 1118–1123. [CrossRef] [PubMed]
16. Kawamoto, T.; Rosvall, M. Estimating the resolution limit of the map equation in community detection. *Phys. Rev. E* **2015**, *91*, 012809. [CrossRef] [PubMed]
17. Rosvall, M.; Bergstrom, C.T. Mapping change in large networks. *PLoS ONE* **2010**, *5*, e8694. [CrossRef]
18. Esquivel, A.V.; Rosvall, M. Compression of flow can reveal overlapping modular organization in networks. *Phys. Rev. X* **2011**, *1*, 021025.
19. Rosvall, M.; Bergstrom, C.T. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PLoS ONE* **2011**, *6*, e18209. [CrossRef]
20. Hamann, M.; Strasser, B.; Wagner, D.; Zetiz, T. Distributed Graph Clustering Using Modularity and Map Equation. In *Euro-Par 2018: Parallel Processing Page*; Springer: Cham, Switzerland, 2018; pp. 688–702.
21. Bae, S.H.; Halperin, D.; West, J.D.; Rosvall, M.; Howe, B. Scalable Flow-Based Community Detection for Large-Scale Network Analysis. In Proceedings of the 2013 IEEE 13th International Conference on Data Mining Workshops, Dallas, TX, USA, 7–10 December 2013.
22. Bae, S.H.; Halperin, D.; West, J.D.; Rosvall, M.; Howe, B. Scalable and Efficient Flow-Based Community Detection for Large-Scale Graph Analysis. *ACM Trans. Knowl. Discov. Data* **2017**, *11*, 32. [CrossRef]
23. Bae, S.H.; Howe, B. GossipMap: A Distributed Community Detection Algorithm for Billion-Edge Directed Graphs. In Proceedings of the SC'15: International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, TX, USA, 15–20 November 2015.
24. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster Computing with Working Sets. *HotCloud* **2010**, *10*, 95.
25. Fung, P.K. InfoFlow: An Apache Spark Implementation of the InfoMap Community Detection Algorithm. Available online: https://github.com/felixfung/InfoFlow (accessed on 21 July 2019).
26. Batagelj, V.; Mrvar, A. Pajek Datasets. Available online: http://vlado.fmf.uni-lj.si/pub/networks/data/ (accessed on 21 July 2019).
27. Leskovec, J.; Krevl, A. SNAP Datasets: Stanford Large Network Dataset Collection. Available online: http://snap.stanford.edu/data (accessed on 21 July 2019).
28. Davis, T.A.; Hu, Y. The university of Florida sparse matrix collection. *ACM Trans. Math. Softw. (TOMS)* **2011**, *38*, 1. [CrossRef]
29. Apache Software Foundation. Apache Spark Committers. Available online: https://spark.apache.org/committers.html (accessed on 21 July 2019).
30. Gonzalez, J.E. GraphLab PowerGraph. Available online: https://github.com/jegonzal/PowerGraph (accessed on 21 July 2019).
31. Bingmann, T.; Axtmann, M.; Jobstl, E.; Lamm, S.; Nguyen, H.C.; Noe, A.; Schlag, S.; Stumpp, M.; Sturm, T.; Sanders, P. Thrill: High-performance algorithmic distributed batch data processing with C++. In Proceedings of the 2016 IEEE International Conference on Big Data (Big Data), Washington, DC, USA, 5–8 December 2016.