Scalable Flow-Based Community Detection for Large-Scale Network Analysis

Seung-Hee Bae*, Daniel Halperin*, Jevin West†, Martin Rosvall‡ and Bill Howe*
*Department of Computer Science and Engineering, University of Washington, Seattle, Washington
Email: {shbae,dhalperi,billhowe}@cs.washington.edu
†Information School, University of Washington, Seattle, Washington
Email: jevinw@uw.edu
‡Department of Physics, Umeå University, Umeå, Sweden
Email: martin.rosvall@physics.umu.se

***Abstract*—Community-detection is a powerful approach to uncover important structures in large networks. Since networks often describe flow of some entity, flow-based community-detection methods are particularly interesting. One such algorithm is called Infomap, which optimizes the objective function known as the map equation. While Infomap is known to be an effective algorithm, its serial implementation cannot take advantage of multicore processing in modern computers. In this paper, we propose a novel parallel generalization of Infomap called RelaxMap. This algorithm relaxes concurrency assumptions to avoid lock overhead, achieving 70% parallel efficiency in shared-memory multicore experiments while exhibiting similar convergence properties and finding similar community structures as the serial algorithm. We evaluate our approach on a variety of real graph datasets as well as synthetic graphs produced by a popular graph generator used for benchmarking community detection algorithms. We describe the algorithm, the experiments, and some emerging research directions in high-performance community detection on massive graphs.**

I. Introduction

Community detection in large graphs [1]–[3] is emerging as a first-class technique in a number of application domains, for example: finding functional similarity in biological networks [4], [5], identifying collaboration communities in research networks [2], and understanding the macro-structure of science through bibliometrics [6].

Community detection methods operate under the intuition that intra-community connections are more common than inter-community connections. Modularity ($\mathcal{Q}$) has been a popular formulation of the problem [7]: Given a partitioning of a graph, a high modularity score indicates that the number of partition-internal edges is higher than would be expected by chance. Modularity is easy to compute and widely applicable. However, modularity optimization methods suffer from a "resolution limit" that depends on the size and connectivity of the network [8]. Further, Guimerà, Sales-Pardo, and Amaral showed that random graphs have high-modularity subsets, suggesting that false structure may be found in practice [9]. Spectral and min-cut techniques have been shown to be effective at finding structure at all scales, but exhibit a bias such that aggressive maximization of certain community score functions can destroy intuitive notions of cluster quality [10].

In response to these limitations, Rosvall *et al.* proposed a flow-based and information-theoretic formulation of the community detection problem known as the *map equation* [11]. Flow in this context is modeled as a random walk through the graph as in the PageRank algorithm [12]. A graph partitioning is scored by finding a compressed modular representation of this flow, with high within-module flow and low between-modular flow [11]. This method has been shown to capture some intuitive notions of community and, when optimized with the search algorithm Infomap [13], to perform very well on real networks and on synthetic benchmarks [14].

Unfortunately, current implementations of Infomap are sequential and cannot scale to handle the graphs with millions and billions of edges that are becoming commonplace. The current sequential algorithm requires 45 minutes to detect communities in a graph with around 5 million vertices and 70 million edges (see Section IV)—a relatively long runtime for a relatively small graph, compared to web-scale graphs with billions of edges. To apply Infomap more broadly requires the ability to optimize the map equation in parallel. Our work represents the first such algorithm.

In this paper, we present *RelaxMap*, a parallel community detection algorithm to optimize the map equation. RelaxMap achieves parallelism over the inherently sequential Infomap by relaxing serial consistency constraints to significantly reduce lock contention, applying global locks only when applying updates to shared state. In particular, a) decisions to move vertices between modules are made in parallel without locking for checking consistency, but b) the algorithm acquires a global lock before applying each move to ensure that the shared module status and the objective function values are updated consistently. We show that these techniques offer significantly improved performance on modern multicore machines while achieving as good or better quality scores and similar convergence rates.

We offer the following contributions:

- We describe a novel parallel algorithm, called *RelaxMap*, which parallelizes the optimization of flow-compression for community detection. To the best of our knowledge, RelaxMap is the first parallel algorithm for flow-based community detection.
- Using synthetic benchmark graphs [14] and a variety of real graphs (where ground truth is unknown), we show that RelaxMap achieves 70% parallel efficiency

up to 12 cores while producing graph clusterings that match the quality of those produced by Infomap.

Section II summarizes the mathematics behind the map equation and the core Infomap algorithm. We describe RelaxMap in Section III, and we evaluate it on both synthetic and real graph datasets in Section IV. Section V contains a discussion of the related work, followed by our conclusions and proposals for future research in Section VI.

II. The Map Equation

Here, we briefly summarize the principles behind the map equation, a flow-based, information-theoretic objective function for evaluating the quality of a graph clustering—an assignment of vertices into modules, or communities. We then describe the core Infomap algorithm for optimizing the map equation over possible clusterings. Infomap is designed to cluster both weighted and directed networks, which arise in many important applications, and has been shown to be one of the most effective clustering techniques in objective third party benchmarks [14], [15]. In the next section, we will show how RelaxMap can optimize the map equation in a parallel fashion.

The term "map" in *map equation* comes from the notion that clustering of real-world networks with respect to flow resembles cartography of traffic infrastructure for better navigation. Optimizing the map equation over possible module assignments identifies modules in which flow stays for a relatively long time, much like geographical maps identify cities as regions in which traffic stays for a relative long time. Flow can refer to real flow of, for example, passengers moving between airports, or flow of random walkers guided by the nodes and links of the network as a proxy for the real flow. But given the network structure, what is the optimal number of modules and the optimal assignment of nodes into those modules?

The map equation answers these questions using the fundamental principles of information theory. All regularities in data can be used to compress the data, such that the degree of compression becomes a measure for the success in finding regularities in the data. The map equation takes advantage of this *minimum description principle* by measuring the description length of a random walker (or of real flow) on a network with a modular codebook structure [11]. Each one of $|\mathsf{M}|$ *modular codebooks* describes movements between nodes assigned to the corresponding module. With $\mathcal{P}^m$ for the probability distribution of node visit rates p_α for $\alpha \in m$ in and exit rate $q_{m\curvearrowright}$ out of module codebook m, the average description length is given by the entropy $H(\mathcal{P}^m)$ according to Shannon's source coding theorem [16], [17]. The frequency of use of module codebook m is $p^m_{\circlearrowright} = q_{m\curvearrowright} + \sum_{\alpha\in m} p_\alpha$, the probability of staying in module m plus the exit rate. Moreover, a single *index codebook* describes movements between the module codebooks. With $\mathcal{Q}$ for the probability distribution of module entering rates $q_{m\curvearrowleft}$, the average description length is given by the entropy $H(\mathcal{Q})$. The frequency of use of the index codebook is $q_{\curvearrowleft} = \sum_{m\in\mathsf{M}} q_{m\curvearrowleft}$. Taken together, the map equation (Eq. 1) measures the average description length L given modular assignments M

$$L(\mathsf{M}) = q_{\curvearrowleft} H(\mathcal{Q}) + \sum_{m\in\mathsf{M}} p^m_{\circlearrowright} H(\mathcal{P}^m). \qquad (1)$$

Algorithm 1 Pseudo code for the serial Infomap algorithm [13]

1: **input:** Network $G = (V, E)$, where V = set of N vertices, E = set of edges. Per-iteration quality improvement threshold τ
2: Run PageRank to calculate steady state probability for each vertex.
3: $\mathsf{M} = \{\{v_i\} \mid v_i \in V\}$
4: $L = L(\mathsf{M})$ in Eq. 1
5: **repeat**
6: $L_{prev} = L$
7: R = random sequence of integers 1 to N
8: **for** i = 0; i < N; i++ **do**
9: m_{new} = bestNewModule(M, $v_{R[i]}$);
10: Move $v_{R[i]}$ to m_{new} module, and update M and L.
11: **end for**
12: **until** $L_{prev} - L < \tau$
13: **return** M

Minimizing the map equation over all possible module assignments gives the optimal modular structure for describing movements on the network, and therefore reveals important structures with respect to the dynamics on the network [11].

Algorithm 1 illustrates the *core algorithm* of the fast stochastic and recursive search algorithm implemented in Infomap [13]. The algorithm proceeds in two phases:

- Phase 1: (line 2) The visit probability (rank) of each vertex is computed in terms of the network flow.
- Phase 2: (lines 5- 12) The space of possible modularizations is greedily searched. The initial modules are singletons — one module per vertex. For each vertex v, the call **bestNewModule(**M**,** v**)** (line 9) checks neighboring modules and greedily selects the one that reduces the *minimum description length (MDL)* L by the largest amount. The algorithm stops when the change in MDL score in each iteration ($L_{prev} - L$) is less than a threshold τ.

In searching procedure for the best new module of a vertex v (line 9), the algorithm calculates the *in-flow* and *out-flow* between the vertex v and its neighbor modules. Finally, the improvement in the MDL for each candidate move can be calculated from the measured in-/out-flow information. The algorithm assigns the vertex v to whichever new module maximizes the MDL improvement. Details of the map equation and the Infomap community detection algorithm are available in the original paper [11] and a dynamic visualization of the technique is available online[1].

III. Parallelizing Flow-Based Community Detection

To parallelize Algorithm 1, we observe that *Phase 1* of that algorithm is the same as PageRank [12] for which there are many parallel and distributed implementations [18]–[21]. The remainder of this section describes how to parallelize *Phase 2*.

[1] http://www.mapequation.org

Algorithm 2 Pseudo code for a naïve lock-free algorithm

1: **for** (**in Parallel**) i = 0; i < N; i++ **do**
2: $\mathsf{M}_{\text{new}}[R[i]]$ = bestNewModule(M, $v_{R[i]}$);
3: **end for**
4: $\mathsf{M} = \mathsf{M}_{\text{new}}$, $L = L(\mathsf{M}_{\text{new}})$

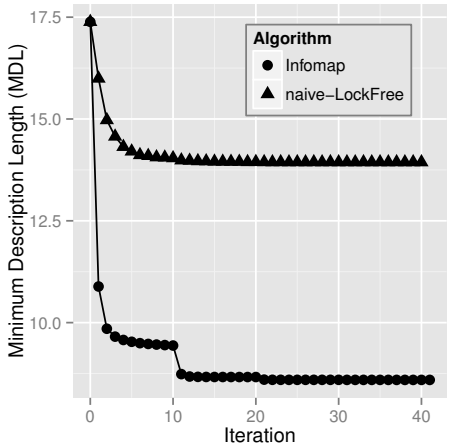


Fig. 1: The map equation values with respect to the iteration number of the naïve lock-free parallel scheme.

To compute the same result as the sequential algorithm in parallel, each thread attempting to move a vertex must acquire a lock on the module to which that vertex belongs as well as locks on all neighboring vertices (including itself) and modules. It is well-known that the lock contention and the corresponding loss of parallel efficiency is a problem for the performance of parallel algorithms [22], and we focused on an approach that minimally uses locks without loss of the output quality much.

A. *The Parallel Algorithm*

We can design a naïve lock-free scheme by checking each vertex independently in parallel and removing the necessity of interactions between neighboring vertices. The naïve lock-free algorithm runs steps 8 through 11 of Algorithm 1 for each vertex in separate threads, and finds new modules for vertices with respect to the module assignment of the previous iteration M_{t-1}. This algorithm is summarized in Algorithm 2. In this scheme, each candidate move is run independently since the previous module assignment for all vertices is already fixed, so the algorithm does not need to use any synchronization to make decisions for several vertices in parallel. This simple parallel algorithm can be implemented by adding a new array for holding current movement decision and updating new module decision and corresponding values at the end.

This approach will achieve perfect parallelism for the most expensive part of this application, from line 8 to line 11 in Algorithm 1, which has complexity $\mathcal{O}(\mathbf{E})$. The update at line 10 in Algorithm 1 now needs to be run after all parallel moves complete since the inner loop, which is lock-free, runs in parallel based on the results of the previous iteration, and each thread stores the moves in a new array. Therefore, it requires

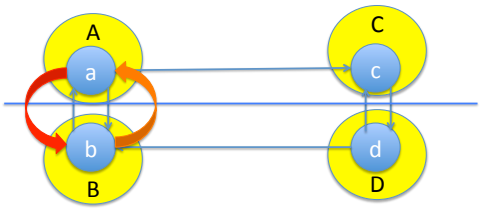


Fig. 2: An example of the cyclical movement in the naïve lock-free parallel algorithm; 4-vertex graphs running in 2-way parallelism.

Algorithm 3 The core-algorithm of the RelaxMap

1: **for** (**in Parallel**) i = 0; i < N; i++ **do**
2: m_{new} = bestNewModule(M, $v_{R[i]}$);
3: acquire a lock for the updates.
4: Move $v_{R[i]}$ to m_{new} module, and update M and L.
5: release lock.
6: **end for**

a procedure to update status values for each module and the new MDL at the end of each iteration (Algorithm 2).

The naïve method described is not competitive with the sequential method, however, as seen in Figure 1. To see why, consider the simple 4-vertex network in Figure 2. In the initial stage, each vertex is assigned to its own module, which we write as $a \in A$, ..., $d \in D$, where $a, \ldots, d$ represent vertex IDs and $A, \ldots, D$ are module IDs. One thread moves a to B, while an independent thread moves b to A, based on the network flow and the module information at the previous iteration (here initial stage). The two moves offset each other, causing the algorithm to make cyclical movements and converge prematurely (Figure 1).

To improve on the naïve method, we propose an algorithm **RelaxMap** based on the assumption that, in general, real network data is sparse, so a movement of a single vertex will typically only affect a small subset of the graph. As a result, if we consider a small number of random vertices concurrently, they are unlikely to influence each other, and the problems with the naïve method will be minimized.

In RelaxMap, each of p threads examines a vertex independently, then acquires a lock to apply the winning move and update the module information. When considering the p vertices, the move decisions are made with stale module information from the previous parallel round. In addition, the RelaxMap parallel algorithm avoids the cyclical-movement problem, shown in Figure 2. To see why, consider the case where one thread p_1 examines a and c and a different thread p_2 examines b and d as in Figure 2. Say two threads worked on a and d concurrently, deciding to move a to B and d to C. Then, as the two threads begin examining c and b respectively, they have access to the current module information; that $a, b \in B$ and $c, d \in C$. So thread p_2 would decide to keep b in B since it knows a is also in B, and no cyclical movement occurs.

There is still some possibility of problem: the two threads could examine a and b at first concurrently, moving a to B and b to A, but the probability of this case will be very low

TABLE I: Network datasets used for evaluating parallel RelaxMap algorithms.

Dataset	Number of vertices ($\|V\|$)	Number of edges ($\|E\|$)	Avg. degree ($2 \times \|E\|/\|V\|$)	Max degree
directNet-1k (Synth.)	1,000	19,849	39.70	69
directNet-5k (Synth.)	5,000	98,313	39.33	69
directNet-10k (Synth.)	10,000	196,414	39.29	69
web-BerkStan	685,230	7,600,595	22.18	84290
web-NotreDame	325,729	1,497,134	9.19	10721
web-Stanford	281,903	2,312,497	16.41	38626
soc-LiveJournal1	4,847,571	68,993,773	28.47	22887
soc-Pokec	1,632,803	30,622,564	37.51	20518
wiki-Talk	2,394,385	5,021,410	4.19	100032

if the number of vertices is large. And, even if some cyclical movements happen during one iteration, they will be fixed at later iterations with high probability if the MDL decreases enough to continue. For instance, among N vertices, there are two strongly connected vertices. If we run p-way parallel RelaxMap ($p << N$ in general), the probability that those two vertices are examined concurrently is $(p-1)/N \approx p/N$. If the algorithm stops after t iterations, then the probability that those two vertices executed at the same time through all the t iterations is $(p/N)^t$ because the algorithm searches new modules for vertices in random order at each iteration. The two vertices will not be run concurrently at least once among t iterations in $1-(p/N)^t \approx 1$ probability. Therefore, we argue that the cyclical movement problem will not be an issue in practice. In worst case, such as $p \geq N$ and each vertex takes the same amount of time for finding a new module, all N nodes examined at the same time and this algorithm would still work as same as the naïve method, at least.

The main difference between the naïve lock-free algorithm and the RelaxMap is the following: RelaxMap considers batches of p vertices concurrently, while the naïve method considers batches of N vertices concurrently. Since $p << N$, the probability that two or more of the p vertices are interdependent is low, and the algorithm can converge almost as quickly as the serial verison.

The RelaxMap algorithm is described in Algorithm 3. One specific feature of the RelaxMap (and the Infomap) algorithm is that the MDL (L) value and the statistics of modules (i.e. exit-rates ($q_{m\curvearrowright}$) and sum of the visit-rates ($\sum_{\alpha \in m} p_\alpha$) for each module m) are always correct with respect to the current module assignment for the consistent and efficient calculation of L and correct algorithmic procedure. For achieving the correctness of those values, we use a global lock in lines 3 and 5 of Algorithm 3. We tested RelaxMap without acquiring the global lock in lines 3 and 5 of Algorithm 3 to see how much it affects the algorithm. This approach was ineffective, for two reasons. First, the inconsistency caused the algorithm to converge very slowly, in some cases slower than the sequential algorithm. Second, the incorrect module information results in incorrect counting for active module numbers, which causes race conditions and subsequent memory faults.

IV. Experimental Analysis

We study RelaxMap experimentally to answer the following two questions: First, *does RelaxMap produce clusterings that match the quality of state of the art flow-based clustering (Infomap) despite RelaxMap's relaxed consistency constraints?* We study both synthetic graphs, for which ground truth is available, and real graph datasets (Section IV-B). Second, *does RelaxMap significantly improve performance over the serial algorithm?* We evaluate performance over the same graph datasets using two different machines and up to 12 cores (Section IV-C). Our results in this section answer both questions in the affirmative: communities identified by RelaxMap match the quality of the state of the art, and RelaxMap achieves 70% parallel efficiency in the machines tested.

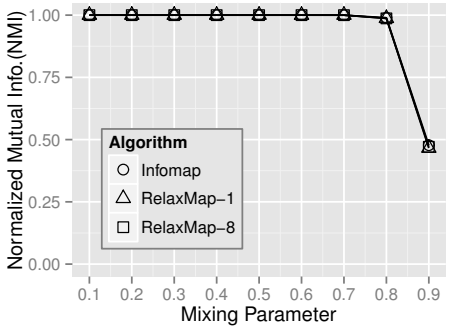
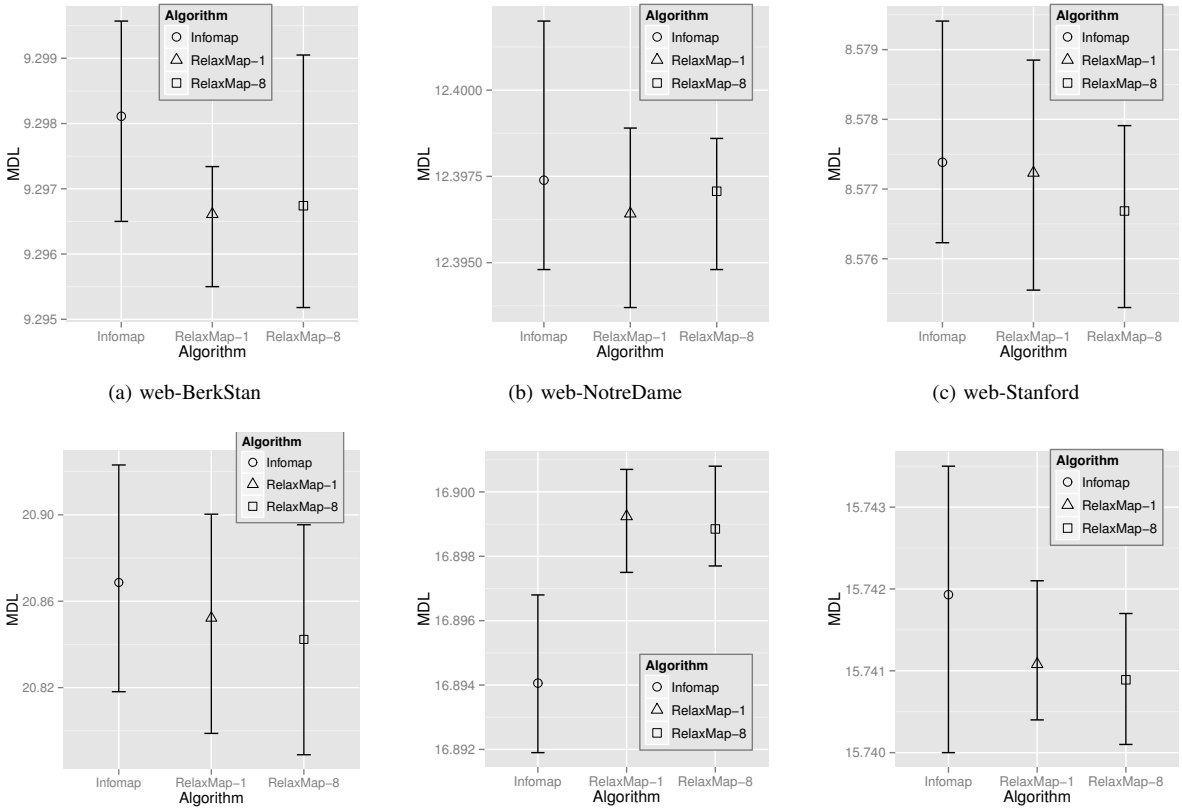


Fig. 3: The average normalized mutual information (NMI) as a function of the mixing parameter comparing **Infomap** and **RelaxMap** for the *directNet-5k* synthetic dataset. This implies that the **RelaxMap** generates outputs that are very similar to the outputs of the **Infomap** algorithm.

A. Experimental Setup

Algorithms. We compare RelaxMap against Infomap, the state of the art serial algorithm to optimize the map equation [11]. We used the open-source implementation of Infomap from Rosvall *et al.* (www.mapequation.org). We implemented RelaxMap using OpenMP [23] for shared memory parallel environments, and ran RelaxMap with up to p concurrent threads, where p is the number of cores on the test machine.

Datasets. We used three different synthetic graphs from a standard clustering benchmark network generator, directNet [14] to evaluate clustering when ground truth communities are available. We follow the parameters given by Lancichinetti *et al.* [14, Section VI-A] to generate graphs of 1000, 5000, and 10000 vertices with "small" communities between 10 and 50 vertices each. We also used six real network datasets from the Stanford



(a) web-BerkStan (b) web-NotreDame (c) web-Stanford

(d) wiki-Talk (e) soc-Pokec (f) soc-LiveJournal1

Fig. 4: The output quality comparison between the **Infomap** and **RelaxMap** algorithm (1-way and 8-way parallelism) with six real-world datasets in Table I. In each plot, the x-axis represents three different algorithms and the y-axis is the MDL score. The output qualities of the RelaxMap-1 and RelaxMap-8 are similar to the output quality of the Infomap with all the tested real datasets except soc-Pokec dataset.

Network Analysis Project (SNAP) [24]. You can find a summary of salient properties of the datasets in Table I, and detailed information on the SNAP website [24].

Test machines. We use two different multicore computers to perform our parallel, shared-memory experiments with RelaxMap. One 8-core machine, *Machine-I*, has two Intel Xeon X5355 quad-core processors (2.66 GHz, 8 MB L2 Cache) and 16 GB of main memory. The 12-core machine, *Machine-II*, has two six-core Intel Xeon E5-2430L processors (2.00 GHz, 15 MB Intel Smart Cache) and 64 GB of main memory.

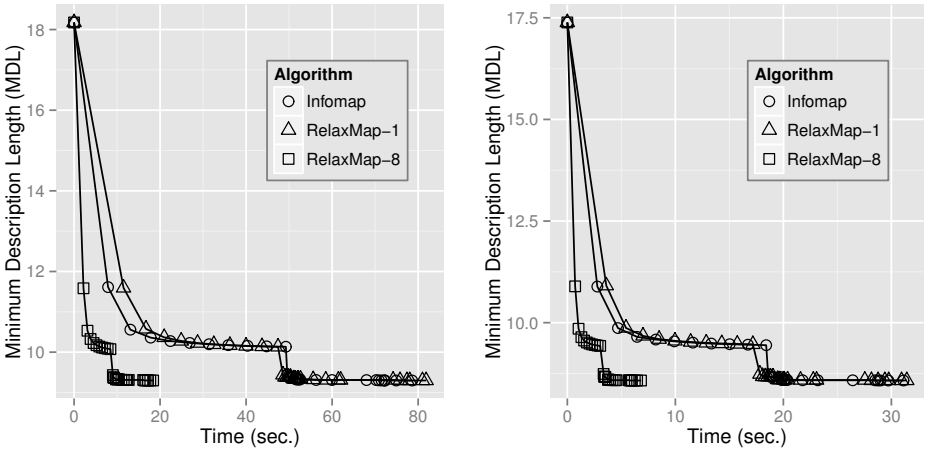
Randomized trials. Except where noted otherwise, all results are based on 10 experimental runs with different random seeds.

B. Clustering Quality Analysis

We first seek to understand whether RelaxMap finds good clusterings even though it uses a relaxed consistency model. We begin with a standard network benchmark, in which synthetic graphs with known communities are constructed randomly according to a mixing parameter that describes how likely inter-community edges are present relative to inter-community edges [14]. Given ground truth communities, the standard score for a clustering is its normalized mutual information (NMI) [25], which equals 1 if it produces the exact same communities as ground truth, and 0 if the two are completely unrelated. Infomap was determined to be the best-performing algorithm in an objective benchmark study [14]; we do not want RelaxMap to compromise on this excellent quality.

Figure 3 shows the NMI of clusterings produced by Infomap, a serial version of RelaxMap that runs with only one thread (RelaxMap-1), and RelaxMap running with 8 threads (RelaxMap-8), over 100 runs for graphs generated with varying mixing parameter. The parallel and serial algorithms achieve the same results: RelaxMap and Infomap find clusterings of equivalent NMI, and are able to identify the ground truth communities with mixing parameter below 0.8. Results are shown for graphs with 5000 nodes, but were identical for other sizes: We conclude that clusterings identified by RelaxMap in parallel are as good as those found by the sequential Infomap algorithm.

In addition to the comparison of the output quality of synthetic datasets, we investigated the output quality of the real-world datasets, as well. Since there is no ground truth communities for the real-world datasets, there is no direct metric for the quality of the communiteis, such as NMI value. However, we found that minimizing MDL value is correlated



(a) web-BerkStan (b) web-Stanford

Fig. 5: The MDL values with respect to the elapsed time for the **RelaxMap** algorithm compared to the **Infomap** algorithm with (a) *web-BerkStan* and (b) *web-Stanford* datasets. Each point in these plots represents the corresponding MDL and time at each iteration for an experiment.

to maximizing corresponding NMI value from synthetic dataset results, so we assume that the less MDL means the better quality on real datasets for which no ground truth exists. Figure 4 compares final output qualities of the RelaxMap algorithm and the Infomap algorithm in terms of the final MDL code length for six real-world datasets in Table I. In Figure 4, each point means an average MDL code length of an experiment and each error-bar represents the minimum and maximum values of the experiment. As illustrated in Figure 4, the quality of the outputs from the RelaxMap algorithm are mostly matched to the quality of the outputs from the Infomap algorithm.

In Figure 4, the result for *soc-Pokec* dataset is different with other results. Although the absolute quality difference in average is negligible (about 0.005), it shows that the RelaxMap algorithm does not capture the same MDL in soc-Pokec dataset case. Both 1-way parallel (**RelaxMap-1**) and 8-way parallel (**RelaxMap-8**) running results from the RelaxMap algorithm are very similar to each other as shown in Figure 4-(e). Since 1-way parallel RelaxMap is logically the same algorithm with the Infomap algorithm, the main reason for this result is not the consistency relaxation feature of the RelaxMap.

Although the core algorithm of the 1-way RelaxMap is logically identical to the Infomap, how to implement those in real programming languages could be slightly different. We have not compared in detail our RelaxMap implementation with the Infomap implementation yet. However, we found that the number of sub-modules generated by the 1-way RelaxMap is larger than by the Infomap out of the similar number of intermediate modules. This implies that the detailed implementations of how to generate sub-modules for both algorithms are different each other, although the concept of generating sub-modules from each module (a.k.a community) is identical. After sub-module finding procedure is done, the RelaxMap-1 experiment and the sequential Infomap algorithm start showing slightly different convergence behaviors in each iteration for the soc-Pokec dataset, unlikely other datasets in Table I, which results in different output quality.

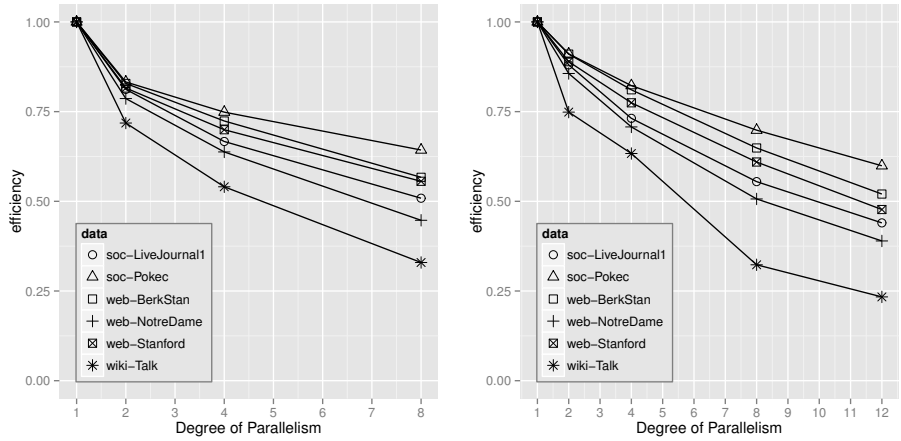
C. Parallel Performance Analysis

In Section IV-B, we discuss the output quality of the proposed parallel algorithm compared to the sequential algorithm. In this section, we would like to analyze parallel performance of the RelaxMap algorithm; 1) how quickly it can be done in parallel, and 2) how efficient it is.

First, we investigate how fast RelaxMap converges to a good clustering. Figure 5 shows the MDL values per unit of elapsed time where each algorithm uses the same random seed. As shown in Figure 5, **RelaxMap** converges in the same number of iterations as Infomap for both the *web-BerkStan* and *web-Stanford* datasets. The sequential version of RelaxMap (*RelaxMap-1*) exhibited the same performance as *Infomap*, and the 8-way parallel RelaxMap (*RelaxMap-8*) is about 4-times faster.

In fact, the proposed parallel algorithm and the original sequential algorithm show the same convergence pattern of the MDL improvement with respect to the number of iteration as in Figure 5 with all the test datasets in Table I. Based on Figure 5 and our test results, we conclude that RelaxMap provides a compatible convergence ratio to Infomap, though it breaks the original algorithm's serial consistency property by lock-free parallel design during decision making. This convergence ratio result implies that our assumptions for applying lock-free mechanism to the proposed parallel algorithm are sensible.

The average, minimum, and maximum running times of Infomap, and 1-way and 8-way parallel RelaxMap with all real-world test datasets on *Machine-I* are in Table II. These running times are corresponding to the experiments of Figure 4. Infomap and 1-way parallel RelaxMap experiments show similar running times, and 8-way parallel RelaxMap is typically about 4 to 5 times faster than the corresponding sequential ones.



(a) Machine-I (b) Machine-II

Fig. 6: Parallel efficiency of the **RelaxMap** algorithm on (a) *Machine-I* and (b) *Machine-II* with tested datasets in Table I.

TABLE II: Running times for Infomap and 1-way and 8-way parallel RelaxMap with various datasets on *Machine-I*.

Dataset	Algorithm	Average Runtime (s)	Minimum Runtime (s)	Maximum Runtime (s)
web-Stanford	Infomap	32.6	32.1	37.7
	RelaxMap-1	30.5	30.1	31.4
	RelaxMap-8	6.9	6.7	7.0
web-NotreDame	Infomap	19.4	19.1	20.1
	RelaxMap-1	19.5	18.9	20.1
	RelaxMap-8	5.5	5.2	5.7
web-BerkStan	Infomap	79.4	76.6	86.5
	RelaxMap-1	81.2	78.8	83.4
	RelaxMap-8	17.9	17.1	18.4
soc-Pokec	Infomap	1127.4	1045.5	1210.4
	RelaxMap-1	1266.2	1225.0	1317.1
	RelaxMap-8	246.2	237.0	260.0
soc-LiveJournal1	Infomap	2653.7	2610.4	2688.3
	RelaxMap-1	1972.4	1949.4	1999.7
	RelaxMap-8	484.7	473.3	495.0
wiki-Talk	Infomap	258.8	162.1	380.1
	RelaxMap-1	236.6	178.1	316.8
	RelaxMap-8	89.8	72.8	102.4

Figure 6 illustrates the parallel efficiency of the RelaxMap algorithm on (a) *Machine-I* and (b) *Machine-II*, correspondingly. In Figure 6, except lower average degree datasets, most of the test cases show from 50% to 70% parallel efficiency in 8-way parallelism on both test environments. This offers a significant improvement over the state of the art.

Although the *wiki-Talk* dataset is larger than the *web-Stanford* dataset in terms of the number of vertices and edges in Table I, the parallel efficiency of the RelaxMap algorithm with the *wiki-Talk* dataset is much lower than the *web-Stanford* dataset in Figure 6. The reason is from the average degree of both datasets. RelaxMap runs in parallel per each vertex of the for-loop in Algorithm 3, and the workload of each vertex is highly correlated to the degree of the vertex since each neighbor must be evaluated. As shown in Table I, the average degree of the *web-Stanford* dataset is much larger than that of the *wiki-Talk* dataset. We find that the overall efficiency is generally correlated with the average degree, by comparing Figure 6 with Table I.

V. Related Work

The community detection problem for a given network (or graph) is a well-known and challenging problem for network structure analysis study. One of the most well-known metrics for the community detection problem is the modularity [7], and the modularity maximization method [26] is one of the mostly used methods for the problem. Riedy *et al.* [27], [28] worked on parallel modularity maximization algorithm under shared-memory environments, like a server with several multi-core processors or Cray-XMT systems, and they modified the original algorithm to achieve better parallel performance.

Rather than the modularity metric, Zhang *et al.* [29] proposed another metric for community detection, called *propinquity*, and provided a parallel algorithm for community detection based on the *propinquity* metric. By utilizing an incremental design for the propinquity calculation, which avoids unnecessary recalculation of the propinquity values per each iteration, they achieved better efficiency in parallel.

Niu *et al.* suggested an interesting *lock-free* scheme for parallel stochastic gradient descent (SGD) algorithms, called *HogWild!* [22], In their paper, Niu *et al.* proved that the HogWild! approach will converge in optimal ratio, which is similar to its original algorithm, given a sparse dataset, although the HogWild! approach allows overwrites on decision variables for the SGD optimization due to not using lock mechanism in shared memory parallelism. Since most of the real network graphs are usually sparse, in this paper, we proposed a parallel flow-based community detection algorithm in terms of the *map equation* [11] metric, by applying lock-free scheme as similar as HogWild! for SGD algorithms.

VI. Conclusion and Future Work

We proposed a parallel flow-based community detection algorithm, called *RelaxMap*. Due to the original algorithm's

sequential and dependent nature, it is difficult to implement an efficient parallel algorithm which follows the sequential properties as exactly same as the original algorithm. The proposed algorithm applies lock-free parallel mechanism in searching new modules of vertices, for achieving better efficiency based on the sparsity assumption which is frequently occurred in real-world networks.

Although the RelaxMap algorithm allows to identify new module of a vertex based on stale information, we believe that it might happen rarely and will not affect the convergence ratio of the algorithm, when the given network is sparse. Empirically, we show the convergence ratio of the RelaxMap algorithm is as fast as the original sequential algorithm.

In addition to the fast convergence ratio, our proposed parallel algorithm provides high-quality outputs which is compatible with outputs of the sequential algorithm, and achieves acceptable efficiency of the parallel executions on both experimental environments.

For future work, we think there is a room for the improvement of the parallel efficiency by considering fine-grained locking structures to reduce lock contention. Also, extending the proposed RelaxMap parallel algorithm to the distributed-memory environment would be a very interesting work which is essential for finding community structure of much more large scale of networks by using hundreds or thousands of parallel units.

ACKNOWLEDGMENT

This work is sponsored in part by a subcontract from Pacific Northwest National Labs and by the National Science Foundation through the collaborative SI2-S2I2 grants 1216726, 1216754, 1216872, 1216879, 1216884. M. Rosvall was supported by the Swedish Research Council grant 2012-3729. We would like to thank Andrea Lancichinetti for assistance with the synthetic graph generator. The authors would also like to thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[2] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[3] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, "Link communities reveal multiscale complexity in networks," *Nature*, vol. 466, no. 7307, pp. 761–764, 2010.

[4] R. Guimera and L. A. N. Amaral, "Functional cartography of complex metabolic networks," *Nature*, vol. 433, no. 7028, pp. 895–900, 2005.

[5] A.-C. Gavin, P. Aloy, P. Grandi, R. Krause, M. Boesche, M. Marzioch, C. Rau, L. J. Jensen, S. Bastuck, B. Dümpelfeld *et al.*, "Proteome survey reveals modularity of the yeast cell machinery," *Nature*, vol. 440, no. 7084, pp. 631–636, 2006.

[6] M. Rosvall and C. T. Bergstrom, "Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems," *PloS one*, vol. 6, no. 4, p. e18209, 2011.

[7] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[8] S. Fortunato and M. Barthélemy, "Resolution limit in community detection," *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, Jan. 2007. [Online]. Available: http://dx.doi.org/10.1073/pnas.0605965104

[9] R. Guimerà, M. Sales-Pardo, and L. A. N. Amaral, "Modularity from fluctuations in random graphs and complex networks," *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, vol. 70, no. 2, 2004. [Online]. Available: http://dx.doi.org/10.1103/physreve.70.025101

[10] J. Leskovec, K. J. Lang, and M. Mahoney, "Empirical comparison of algorithms for network community detection," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 631–640. [Online]. Available: http://doi.acm.org/10.1145/1772690.1772755

[11] M. Rosvall, D. Axelsson, and C. T. Bergstrom, "The map equation," *The European Physical Journal Special Topics*, vol. 178, no. 1, pp. 13–23, 2009.

[12] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1, pp. 107–117, 1998.

[13] 2013, the source code is available here: www.mapequation.org.

[14] A. Lancichinetti and S. Fortunato, "Community detection algorithms: A comparative analysis," *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, vol. 80, no. 056117, 2009.

[15] R. Aldecoa and I. Marín, "Exploring the limits of community detection strategies in complex networks," *Scientific reports*, vol. 3, no. 2216, 2013.

[16] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, July 1948.

[17] ——, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 623–656, October 1948.

[18] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, "Distributed graphlab: a framework for machine learning and data mining in the cloud," *Proc. VLDB Endow.*, vol. 5, no. 8, pp. 716–727, Apr. 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2212351.2212354

[19] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative mapreduce," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 810–818.

[20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012, pp. 2–2.

[21] D. F. Gleich and L. Zhukov, "Scalable computing with power-law graphs: Experience with parallel PageRank," in *SuperComputing 2005*, November 2005, poster.

[22] F. Niu, B. Recht, C. Ré, and S. J. Wright, "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent," in *NIPS*, 2011. [Online]. Available: http://books.nips.cc/papers/files/nips24/NIPS2011_0485.pdf

[23] OpenMP Architecture Review Board, "OpenMP application program interface version 3.0," http://www.openmp.org/mp-documents/spec30.pdf, May 2008.

[24] J. Leskovec, "Stanford large network dataset collection," http://snap.stanford.edu/data/index.html, [Online; accessed June-19-2013].

[25] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 09, p. P09008, 2005.

[26] A. Clauset, M. E. Newman, and C. Moore, "Finding community structure in very large networks," *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[27] E. J. Riedy, H. Meyerhenke, D. Ediger, and D. A. Bader, "Parallel community detection for massive graphs," in *Proceedings of the 9th international conference on Parallel Processing and Applied Mathematics-Volume Part I*. Springer-Verlag, 2011, pp. 286–296.

[28] J. Riedy, D. A. Bader, and H. Meyerhenke, "Scalable multi-threaded community detection in social networks," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. IEEE, 2012, pp. 1619–1628.

[29] Y. Zhang, J. Wang, Y. Wang, and L. Zhou, "Parallel community detection on large networks with propinquity dynamics," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 997–1006.