On the Parallelization of MCMC for Community Detection

Frank Wanye
wanyef@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

Vitaliy Gleyzer
vgleyzer@ll.mit.edu
MIT Lincoln Laboratory
Lexington, Massachusetts, USA

Edward Kao
edward.kao@ll.mit.edu
MIT Lincoln Laboratory
Lexington, Massachusetts, USA

Wu-chun Feng
wfeng@vt.edu
Virginia Tech
Blacksburg, Virginia, USA

ABSTRACT

The rapid growth in size of real-world graph datasets necessitates the design of parallel and scalable graph analytics algorithms for large graphs. Community detection is a graph analysis technique with use cases in many domains from bioinformatics to network security. Markov chain Monte Carlo (MCMC)-based methods for performing community detection, such as the stochastic block partitioning (SBP) algorithm, are robust to graphs with a complex structure, but have traditionally been difficult to parallelize due to the serial nature of the underlying MCMC algorithm. This paper presents hybrid SBP (H-SBP), a novel hybrid method to parallelize the inherently sequential computation within each MCMC chain, for SBP. H-SBP processes a fraction of the most influential graph vertices serially and the remaining majority of the vertices in parallel using asynchronous Gibbs. We empirically show that H-SBP speeds up the MCMC computations by up to 5.6 × on real-world graphs while maintaining accuracy.

CCS CONCEPTS

• **Mathematics of computing** → **Graph algorithms**; **Gibbs sampling**; *Metropolis-Hastings algorithm*; • **Computing methodologies** → **Bayesian network models**; **Shared memory algorithms**.

This project was supported in part by NSF I/UCRC CNS-1822080 via the NSF Center for Space, High-performance, and Resilient Computing (SHREC).

ICPP '22, August 29-September 1, 2022, Bordeaux, France

ACM ISBN 978-1-4503-9733-9/22/08.
https://doi.org/10.1145/3545008.3545058

KEYWORDS

community detection, graph clustering, stochastic blockmodels, bayesian inference, asynchronous gibbs

ACM Reference Format:
Frank Wanye, Vitaliy Gleyzer, Edward Kao, and Wu-chun Feng. 2022. On the Parallelization of MCMC for Community Detection. In *51st International Conference on Parallel Processing (ICPP'22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3545008.3545058

1 INTRODUCTION

In real-world datasets, entities can often be grouped into clusters based on their similarities. In some datasets, these entities are linked via relationships, such as friendships or follows in social media datasets, and chemical interactions in protein-protein interaction datasets. In such datasets, often referred to as graphs, where entities are represented by graph vertices and relationships are represented by graph edges, such clusters are identified by looking for sets of vertices that have a high number of within-group edges and a low number of between-group edges. These clusters are called communities, and community detection is the process of identifying communities in a graph dataset. Community detection has a variety of use cases, ranging from social media analysis [17] to identifying functional groups in protein-protein interaction networks [21], which has led to lots of interest and research in this field.

The structure of graph datasets can be represented using stochastic blockmodels (SBMs) [11], a set of generative models that represent the structure of a graph via the graph's communities. In practice, this representation manifests as a sparse matrix B of size $C \times C$, where C is the number of communities and $B_{a,b}$ is either the number of edges that originate from community a and are incident on community b, or the probability of said edges occurring, depending on the chosen SBM representation. SBMs can be used to generate graphs with community structure, as well as to perform community detection in a given graph by fitting a variant of the SBM to the given data, often via optimizing a quality function.

Stochastic block partitioning (SBP) [18, 19] is a community detection algorithm based on inference over a variant of the SBM. SBP optimizes the minimum description length (MDL) of the SBM via the Metropolis-Hastings algorithm [8], a Markov chain Monte Carlo (MCMC) optimization technique. This algorithm works well on graphs that are traditionally hard to perform community detection on due to a high variation of community sizes and a high degree of between-community connectivity, but is slower and less

scalable than alternatives, especially those based on modularity maximization [7] and label propagation [14].

The scalability issues of SBP stem from the fact that the Metropolis-Hastings algorithm is inherently serial. There exists an alternative MCMC optimization technique, asynchronous Gibbs sampling [22], which is embarrassingly parallel, but does not always converge. Asynchronous Gibbs has been studied in various contexts [4, 5, 12], but we are unaware of any work that has studied its applications to community detection. In this manuscript, we present asynchronous stochastic block partitioning (A-SBP), a variation of SBP that uses asynchronous Gibbs instead of Metropolis-Hastings, and study how this change affects the speedup and quality of community detection.

The asynchronous Gibbs algorithm is a conceptually promising parallel approximation of MCMC because the graph structure, which including communities, encodes the sparse statistical dependencies between vertex community memberships. Our A-SBP results confirm that parallel MCMC can be done without sacrificing much inference accuracy when the statistical dependencies are sparse. Theoretic conditions based on the influence concept [4, 22] have been previously proposed for predicting whether or not the statistical dependencies are sparse, i.e. whether or not asynchronous Gibbs sampling would work just as well as the synchronous variants in non-community detection domains, but they are non-trivial to calculate in the SBM and community detection domain.

Based on intuitive assumptions about influence and previous work on the information content of graph edges [10], we propose a hybrid SBP variant, H-SBP, which divides up the vertices between a small set of important vertices that gets processed synchronously, and a large set of vertices that gets processed asynchronously. We empirically show that H-SBP improves upon on the accuracy of A-SBP in the cases where A-SBP fails to converge, at the expense of some speedup. We also show that H-SBP matches the accuracy of SBP in real-world graphs, while also speeding up the computation.

Our main contributions can be summed up as follows:

- We parallelize an inherently sequential MCMC method using approximate computation via asynchronous Gibbs, leading to 5.6 × speedup without a significant impact on accuracy.
- We provide an empirical evaluation of our parallel MCMC method in the context of community detection. We show that on real-world graphs, our novel hybrid stochastic block partitioning (H-SBP) algorithm matches the accuracy of SBP while speeding up the overall computation by up to 4.2 ×.

2 BACKGROUND AND RELATED WORK

In this section, we introduce background material and other work related to this manuscript.

2.1 Stochastic Blockmodels

Stochastic blockmodels (SBMs) [11] are a set of generative models for communities in a graph dataset. They can be used to generate graphs with community structure, or they can be fitted to existing graphs to infer community structure. A typical blockmodel is a sparse matrix $M_{C\times C}$, where C is the number of communities in the graph. Depending on whether the blockmodel is microcanonical or canonical, $M_{a,b}$ either holds the number of edges that originate from community a and are incident on community b, or the probability of an edge forming between communities a and b, respectively.

In order to use SBMs to perform community detection, a quality function is needed. One simple quality function is the log-likelihood of the graph G given the blockmodel B, $L(G|B)$. For the variant of blockmodel we focus on, the Degree-Corrected SBM (DCSBM), $L(G|B)$ is given by the following equation [9, 18]:

$$L(G|B) = \sum_{i,j} B_{i,j} \log\left(\frac{B_{i,j}}{d_i^{out} d_j^{in}}\right), \tag{1}$$

where $B_{i,j}$ is the number of edges between vertices in communities i and j, and d_k^{out} and d_k^{in} refer to the out- and in-edges incident on community k, respectively.

However, $L(G|B)$ as a quality function only works when the number of communities is known apriori, because it will always be maximized when every vertex is placed in a separate community. In order to prevent that from happening, the Minimum Description Length (MDL) is used instead. For the DCSBM, the MDL is given by the following equation [9, 18]:

$$MDL = Eh\left(\frac{C^2}{E}\right) + V \log C - L(G|B), \tag{2}$$

where $h(x) = (1 + x)\log 1 + x - x \log x$, and E, V, and C are the number of edges, vertices, and communities in the graph G, respectively.

2.2 Stochastic Block Partitioning

Stochastic block partitioning (SBP) [9, 18, 19] is an iterative community detection algorithm based on inference over the stochastic blockmodel. It comprises of two phases – the block merge phase and the MCMC phase, which is implemented via the serial Metropolis-Hastings algorithm [8]. In the block merge phase (see Algorithm 1), a merge is proposed for every community in B. The merges are then sorted by the resulting change in MDL, ΔMDL, and are applied one after another until the number of communities is halved (or otherwise reduced by a fixed amount). In the MCMC phase (see Algorithm 2), changes in community membership are proposed for each vertex in G. For each proposal, ΔMDL is calculated, and the Metropolis-Hastings acceptance ratio is computed from the ΔMDL. Based on this ratio, if the proposal is accepted, the blockmodel B is updated in place. The overall algorithm performs a fibonacci search by alternating between the two phases until the MDL is minimized, and the optimal number of communities is found. It is visually summarized in Fig. 1.

Computing ΔMDL and the subsequent updates to B are the two main computational bottlenecks of SBP. The block merge phase is embarrassingly parallel until the sort and application of merges. The MCMC phase, on the other hand, is inherently serial because it runs a single MCMC chain until convergence. Within this chain, every new proposal depends on the up-to-date state of the blockmodel. Using stale versions of the blockmodel could lead to inaccurate proposals. Parallelizing the MCMC chain is therefore non-trivial. Parallelizing a single MCMC chain would typically incur a lot of communication overhead, while running multiple chains in parallel on the same data may lead to significant repeated "burn-in" time as the MCMC chain stabilizes on each process [24]. There

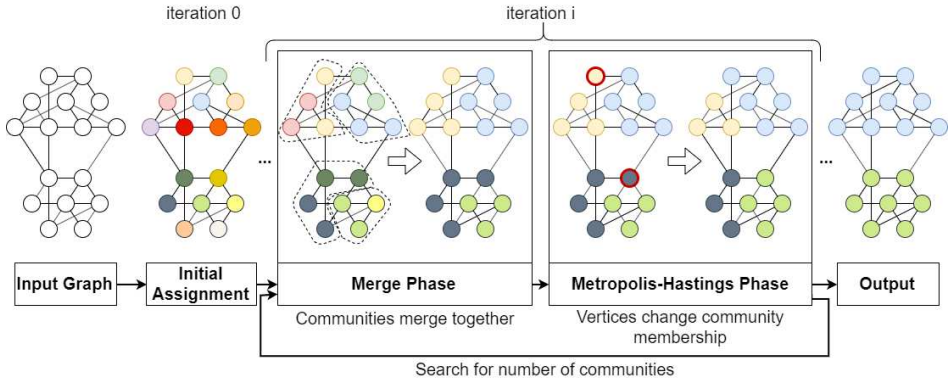


Figure 1: Snapshots of a graph at various stages of the stochastic block partitioning algorithm.

Data: Graph G, Blockmodel B, int x
Result: Updated Blockmodel B
initialize best_merges container;
for *community* $c \in B$ **do in parallel**
  repeat x **times**
    Propose a new community c' to merge with c;
    Calculate ΔMDL when c is merged with c';
    if ΔMDL *is best obtained so far for* c **then**
      Store c' and ΔMDL for c in best_merges;
    end
  end
end
sort best_merges on ΔMDL;
repeat
  c, c' = best_merges.pop();
  Merge c into c';
until *number of communities is halved*;

Algorithm 1: Block Merge Phase

Data: Graph G, Blockmodel B, double t, int x
Result: Updated Blockmodel B
compute MDL of B;
repeat
  foreach *vertex* $v \in G$ **do**
    propose new community c for v;
    compute ΔMDL for proposed move;
    compute Metropolis-Hastings ratio from ΔMDL;
    if *move is accepted* **then**
      move v to c and update B;
    end
  end
  compute MDL of B;
until $\Delta MDL < t \times MDL$ *or* x *times*;

Algorithm 2: MCMC Phase

has been some work in parallelizing MCMC by dividing the data between worker threads/processes and then combining the samples once all chains are sufficiently mixed [15], but combining multiple blockmodels is a non-trivial proposition. Consequently, the MCMC phase in SBP is run serially, and generally takes up the majority of algorithm runtime, as seen in Fig. 2.

2.3 Asynchronous Gibbs

Asynchronous Gibbs [22] is an alternative, parallel MCMC algorithm for sampling from an unknown distribution where multiple workers propose samples for the same Markov chain. From a computational perspective, there are two main differences between asynchronous Gibbs and Metropolis-Hastings. The first is that the asynchronous Gibbs algorithm accepts all proposals. The second is that changes to the underlying distribution are communicated asynchronously, and may not be received by other workers. That is, each worker has an incomplete and partially stale view of the current state of the Markov chain. This limitation breaks the dependency chain of MCMC algorithms, and it has been shown that the asynchronous Gibbs is not guaranteed to converge at the optimal solution.

There has been a substantial amount of work into understanding when asynchronous Gibbs will converge quickly. In [4], it was shown that asynchronous Gibbs will converge quickly when the total influence on the system, $\alpha < 1$. The value of α can be computed using Equation 3.

$$\alpha = \max_{i \in I} \sum_{j \in I} \max_{(X,Y) \in B_j} ||\pi_i(\cdot|X_{I\{i\}}) - \pi_i(\cdot|Y_{I\{i\}})||_{TV}, \qquad (3)$$

where π is a probability distribution of the set of variables I, B_j is the set of all state pairs (X, Y) that only differ by the value of variable j, and $\pi_i(\cdot|X_{I\{i\}})$ is the conditional distribution of variable i in π given that all other variables in state X are fixed.

However, for a complex problem like community detection, computing α becomes computationally intractable – we find that a naive implementation of the α computation, where the vertices are the variables in the model and the communities describe the states that the variables can belong to, and assuming a known initial state of the blockmodel B with C communities, is an $O(V^2C^3)$ operation. Factoring in initialization, access and updates to any underlying data structures further worsens the computational complexity. Such

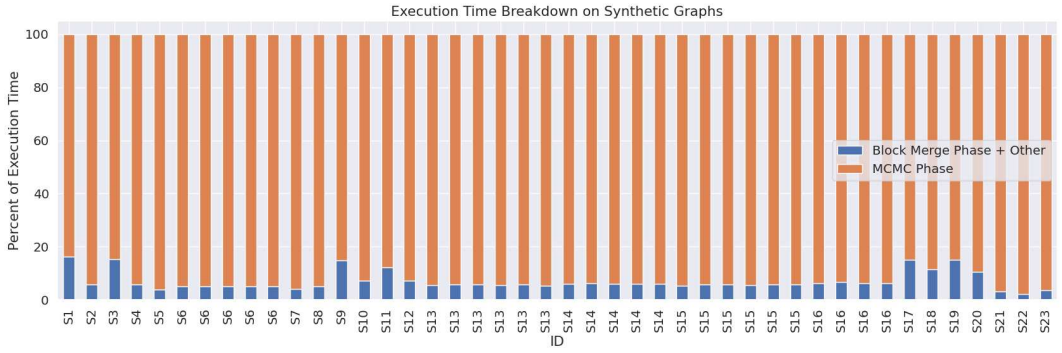


Figure 2: The breakdown of SBP execution time between the MCMC phase and the rest of the algorithm. The MCMC phase takes up to 98% of the overall SBP execution time.

a computation is infeasible on large graph datasets and real-world scenarios where graphs may have upwards of a billion vertices and hundreds of thousands, if not more, communities.

A more accurate version of asynchronous Gibbs is the *exact* asynchronous Gibbs algorithm, as proposed in [22]. This version of asynchronous Gibbs computes the Metropolis-Hastings ratio for each proposed vertex move and uses a similar acceptance criterion to the Metropolis-Hastings algorithm, at the expense of additional communication overhead. We base our asynchronous Gibbs implementation on this exact algorithm.

3 METHODOLOGY

In this section, we describe the methods we use to parallelize a single MCMC chain to speed up community detection. First, we describe our asynchronous Gibbs-based algorithm. Then, we describe a hybrid Metropolis-Hastings and asynchronous Gibbs approach, which process important vertices serially, and less important vertices using the faster asynchronous Gibbs approach, thereby providing a compromise between speed and accuracy.

3.1 Asynchronous Stochastic Block Partitioning

We introduce asynchronous stochastic block partitioning (A-SBP), our shared memory asynchronous Gibbs-based implementation of SBP. Here, we replace the Metropolis-Hastings algorithm with a version of asynchronous Gibbs that is suitable for blockmodel inference. In this algorithm, the blockmodel B and graph G are stored in shared memory and accessible to all threads without locks. Each thread is responsible for proposing a new community for a set of vertices, and computing the Metropolis-Hastings ratio for each proposal. If a move is accepted, then the thread updates the community membership of the vertex for which the move was proposed *without* modifying the blockmodel. Once a pass has been made over all vertices $v \in G$, the blockmodel B is rebuilt using the updated community membership information. This is repeated until ΔMDL falls below a threshold t or the number of iterations reaches a threshold x. The pseudocode for the MCMC phase of A-SBP is outlined in Algorithm 3.

Like in the Exact Asynchronous Gibbs method proposed in [22], A-SBP computes the Metropolis-Hastings ratio and does not accept

```
Data: Graph G, Blockmodel B, double t, int x
Result: Updated Blockmodel B
compute MDL of B;
repeat
    copy community_membership vector from B;
    for vertex v ∈ G do in parallel
        propose new community c for v;
        compute ΔMDL for proposed move;
        compute Metropolis-Hastings ratio from ΔMDL;
        if move is accepted then
            community_membership[v] = c;
        end
    end
    rebuild B from community_membership;
    compute MDL of B;
until ΔMDL < t × MDL or x times;
```

Algorithm 3: MCMC Phase of A-SBP

every proposed change to B. However, A-SBP does not communicate changes to B in an asynchronous manner. Instead, the changes are "communicated" at the end of every pass over the graph vertices, meaning the Asynchronous Gibbs sampler is always computing the Metropolis-Hastings ratio using a distribution that is at most one iteration stale. This is necessary for the community detection problem because B can be large, especially in the initial iterations, and having each thread store a copy of B that it can independently modify would lead to memory bandwidth issues. Moreover, the overhead from rebuilding B would still be present using the method in [22], since each thread would have a different version of B in memory, and these versions would have to be consolidated in order to compute an accurate ΔMDL for the updated blockmodel. This overhead can be reduced by performing the reconstruction of B in parallel.

3.2 Hybrid Stochastic Block Partitioning

Previous work [4] has linked the performance of asynchronous Gibbs sampling with the total influence α in the graph G. In terms of community detection, the value of alpha can be summarized as

the maximum change in the probability distribution of the community membership of any vertex i, when another vertex j changes its community membership. Computing α is intractable on large graphs, and thus studying influence in the real-world setting is challenging. However, there are certain intuitive assumptions that can be made about influence and vertex degree:

- High-degree vertices are generally the most influential vertices for community detection. This assumption is somewhat corroborated by a related finding in [10], which showed that the community information content of an edge is proportional to the product of the vertex degrees of the vertices that make up said edge.
- Because many, if not most real-world graph degree distributions follow the power law [1], the number of highly influential vertices in a graph is going to be relatively low.

Based on these assumptions, we introduce hybrid stochastic block partitioning (H-SBP). In H-SBP, vertices are initially sorted by their degrees, and a set of high-degree vertices, V^+, is selected. Then, in each MCMC iteration, first one pass of the Metropolis-Hastings algorithm is run on the vertices in V^+, giving them a chance to switch communities first. A second pass is then performed over the remaining low-degree vertices V^- using A-SBP. In this manner, H-SBP trades off some of the parallelizability and runtime improvement of A-SBP for improved convergence, which translates into higher result quality. The pseudocode for H-SBP is shown in Algorithm 4.

```
Data: Graph G, Blockmodel B, double t, int x
Result: Updated Blockmodel B
compute MDL of B;
repeat
    copy community_membership vector from B;
    foreach vertex v ∈ V+ do
        propose new community c for v;
        compute ΔMDL for proposed move;
        compute Metropolis-Hastings ratio from ΔMDL;
        if move is accepted then
            move v to c and update B;
        end
    end
    for vertex v ∈ V− do in parallel
        propose new community c for v;
        compute ΔMDL for proposed move;
        compute Metropolis-Hastings ratio from ΔMDL;
        if move is accepted then
            community_membership[v] = c;
        end
    end
    rebuild B from community_membership;
    compute MDL of B;
until ΔMDL < t × MDL or x times;
```

Algorithm 4: MCMC Phase of H-SBP

4 EXPERIMENTAL SETUP

In this section, we outline how we conduct our experiments.

4.1 Datasets

Table 1: Synthetically Generated Graphs

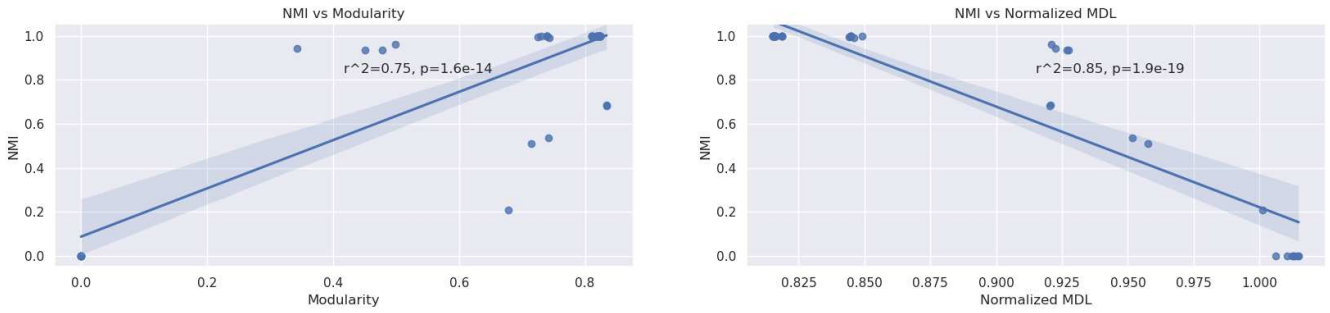
ID	V	E	r
S1	198101	321071	3
S2	199643	425466	3
S3	197894	322196	3
S4	199219	436203	3
S5	225999	4463267	3
S6	225999	5864094	3
S7	225999	4536499	3
S8	225999	6327321	3
S9	197552	321509	5
S10	199564	425382	5
S11	196287	323076	5
S12	199564	426813	5
S13	225999	4502604	5
S14	225999	5891353	5
S15	225999	4495263	5
S16	225999	6277133	5
S17	199285	322338	1
S18	201169	427949	1
S19	198875	322236	1
S20	201506	447244	1
S21	225999	4481133	1
S22	225999	5896200	1
S23	225999	4523706	1
S24	225999	6247681	1

We use a DCSBM-based graph generator, implemented via the 'graph-tool' library [20], to generate several synthetic graphs with known community memberships. We vary the degree distribution and strength of community structure in said graphs by modifying several input parameters to the generator, including minimum and maximum vertex degree, power law exponent of the degree distribution, and the ratio of within-to-between community edges r in the DCSBM. The parameters of the resulting graphs are summarized in Table 1. Note that because the generator function is stochastic and tries to fit both the DCSBM and the provided degree distribution, the final graphs are close to, but do not correspond exactly to the input parameters provided.

We also select 14 unweighted, directed real-world graph datasets from the SuiteSparse Matrix Collection [3] from different domains (see Table 2). Their community information is unknown, and as a result so are their r values.

4.2 Experiments

For each synthetic and real-world graph, we run SBP, A-SBP, and H-SBP 5 times, and select the result that leads to the lowest MDL. In all the runs, the block-merge phase is run in parallel, so any differences in runtime can be attributed solely to the change in the algorithm used in the MCMC phase. For H-SBP, we reserve



(a) (b)

Figure 3: Comparison of the correlation between NMI and Modularity (left), and NMI and Normalized MDL (right). Normalized MDL is more strongly correlated with NMI than Modularity.

Table 2: Real-World Graphs

ID	V	E
rajat01	6847	43262
wiki-Vote	7115	103689
barth5	15622	61498
cit-HepTh	27770	352807
p2p-Gnutella31	62586	147892
soc-Epinions1	75879	508837
soc-Slashdot0902	82168	948464
cnr-2000	325557	3216152
amazon0505	410236	3356824
higgs-twitter	456626	14855842
Stanford-Berkeley	683446	7583376
web-BerkStan	685230	7600595
amazon-2008	735323	5158388
flickr	820878	9837214

15% of the highest-degree vertices to be processed in sequential order. To compute MCMC phase speedup, we measure the total time taken by the MCMC phase across all MCMC iterations on all runs. Additionally, we measure the total time taken up by the community detection algorithm, including the block merge phase, to compute the overall speedup.

We also run a strong scaling experiment using H-SBP on the soc-Slashdot0902 graph, where we vary the number of parallel threads from 1 to 128, and record the resulting total MCMC phase runtime.

On synthetic graphs with known ground truth, we measure the normalized mutual information (NMI) [13] between the community memberships in the ground truth and those found by the community detection algorithm. We calculate NMI as $NMI = \frac{I(X,Y)}{H(X)+H(Y)}$, where $I(X,Y)$ is the mutual information between two sets of community memberships X and Y, and $H(X)$ is the entropy of the community memberships X.

In the case of the real-world graphs, where there is no known ground truth, we measure the MDL of the blockmodel that forms the algorithm's solution. MDL has previously been used to evaluate graphs with no ground truth [23], but it varies greatly with graph size. In order to make the MDL values comparable, we expand on this work by presenting the normalized version of the MDL – MDL^{norm} – which is normalized via a structure-less null blockmodel. We choose the null blockmodel to be one where every vertex belongs to the same community. The normalized MDL is therefore given by $MDL^{norm} = \frac{MDL}{MDL^{null}}$, where MDL^{null} is the MDL of the null blockmodel for a given graph. We find that MDL^{norm} not only correlates strongly with NMI on synthetic graphs, but is also comparable across graphs. Furthermore, it is a direct measure of how well the SBP algorithm performs at minimizing MDL. For the sake of completeness, we also present the Newman's Modularity [16] scores for the same runs. However, Modularity has been previously shown to a) not always be able to separate graphs with very different structure [2], and b) not always correspond to the most semantically appropriate community labels [6]. In our experiments, we find that Modularity does correlate with NMI on synthetic graphs, but not as strongly as MDL^{norm} (see Fig. 3).

We run these experiments on the Virginia Tech Tinkercliffs cluster. Each node on the cluster contains an AMD EPYC 7702 CPU with 128 cores and 256 GB of memory. We use OpenMP to parallelize the parallel loops in Algorithms 1, 3 and 4 with 128 threads in all experiments unless stated otherwise.

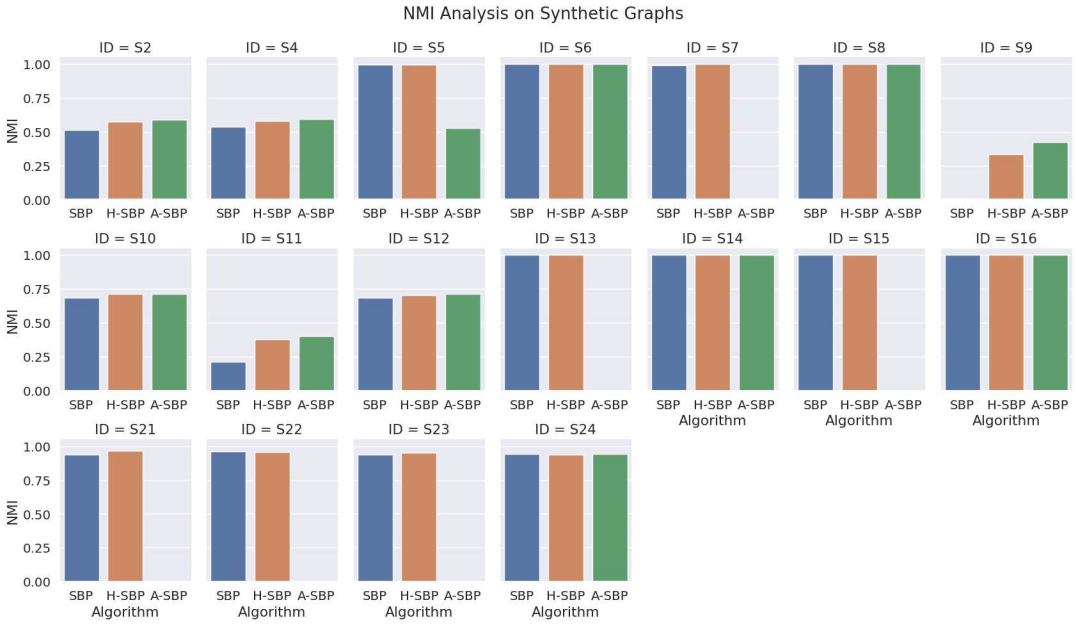
5 RESULTS

In this section, we discuss the results obtained from the previously described experiments. For the sake of brevity, we redact results of 6 synthetic graphs on which all 3 algorithms failed to converge due to a combination of low r values and low density, leading to too little community structure being present in the graph.

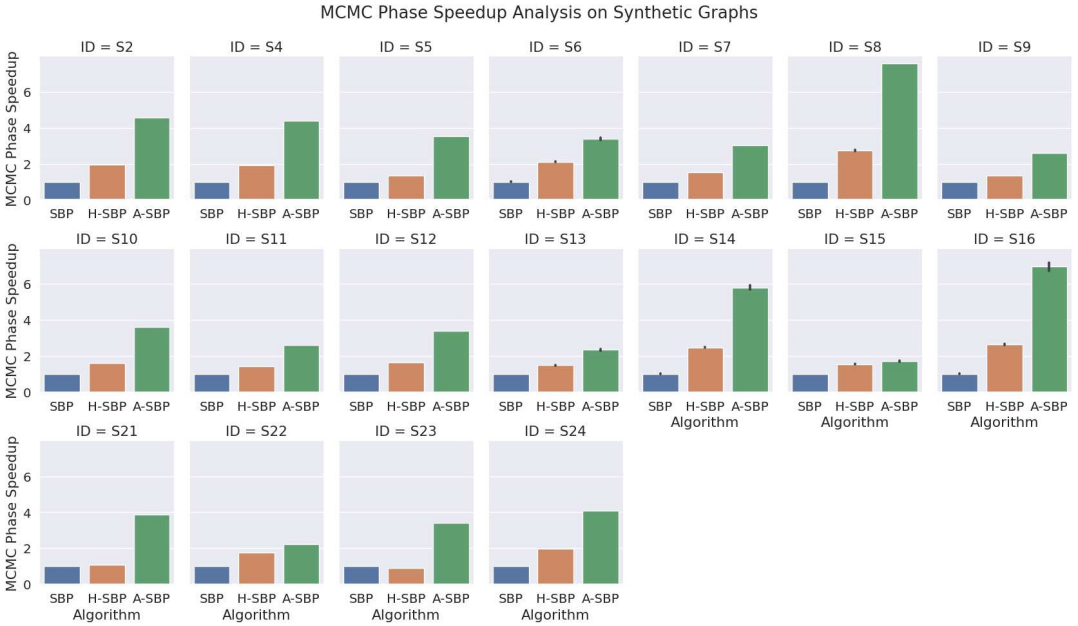
5.1 Accuracy Results on Synthetic Graphs

Our results show that A-SBP matches the accuracy, in terms of NMI, of SBP on 10 out of the 18 graphs, but fails to properly converge otherwise. H-SBP, on the other hand, matches the accuracy of SBP on all graphs on which SBP converges, making it the more versatile approach. A summary of these results is shown in Fig. 4a.

On two of the graphs, S9 and S11, A-SBP and H-SBP achieve higher NMI than SBP. On S9, SBP fails to converge entirely, while



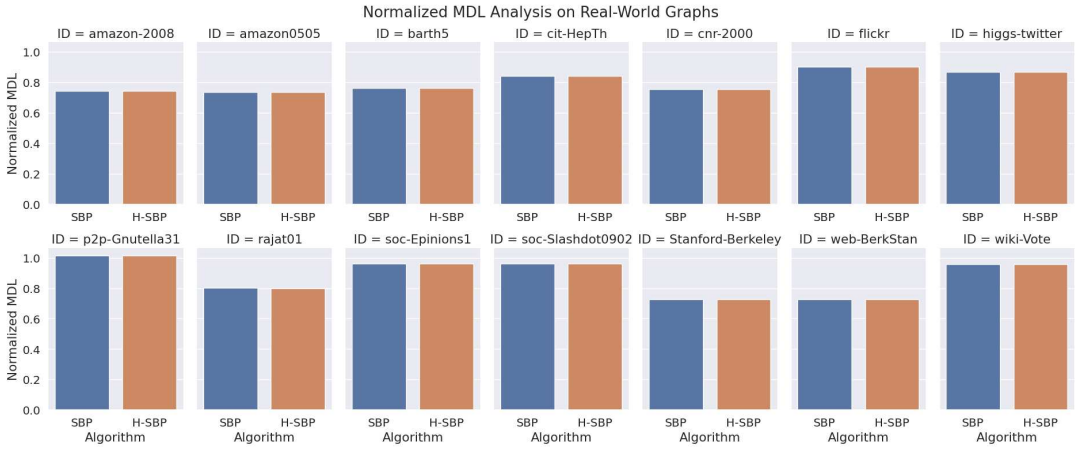
(a) NMI results with SBP, H-SBP, and A-SBP on the synthetic graphs. A-SBP matches the scores obtained with SBP on approximately half of the synthetic graphs, while H-SBP performs as well as SBP on all 18 graphs.



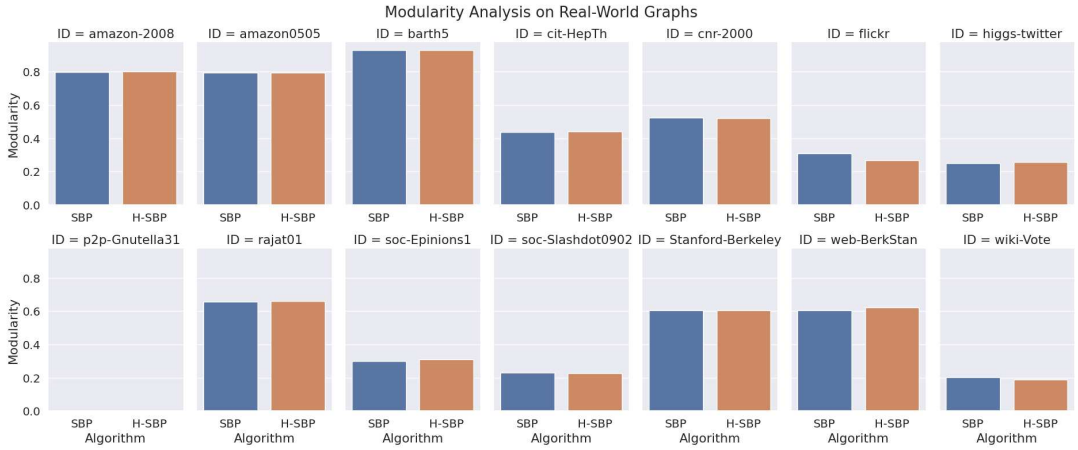
(b) MCMC phase speedup results with SBP, H-SBP, and A-SBP on the synthetic graphs. A-SBP provides significant speedups over SBP on all graphs, up to just over 6X. H-SBP also provides a significant speedup on most graphs, up to approximately 2.3X.

Figure 4: Experimental results on synthetic graphs, showing NMI results (top) and MCMC phase speedup results (bottom).

A-SBP achieves an NMI of approximately 0.4. On S11, both algorithms converge, but A-SBP achieves a higher NMI score. However, both of these graphs are very sparse and as a result have little community structure; both of their true MDL^{norm} values are very close to 1.0, meaning there is not much more structural information in the ground truth labels than in placing all vertices in a single community. On these two graphs, H-SBP behaves similarly to A-SBP, showing NMI values in between those obtained with SBP and A-SBP.



(a)



(b)

Figure 5: Normalized MDL (top) and Modularity (bottom) results with SBP and H-SBP on the real-world graphs. H-SBP matches the result quality of SBP on all graphs, in terms of both Normalized MDL and Modularity.

5.2 Speedup results on synthetic graphs

Our results on synthetic graphs show that A-SBP significantly speeds up the MCMC phase speedups over SBP on all graphs, ranging from approximately 1.7 × on S15 to 7.6 × on S8. The MCMC phase runtime of H-SBP is generally in between that of SBP and A-SBP, with the highest speedup being just 2.7 × on S8. This is despite the fact that both A-SBP and H-SBP require significantly more MCMC iterations to converge on the synthetic graphs (see Fig. 8a in Appendix A). The speedup results are summarized in Fig. 4b. Due to Amdahl's Law, the overall speedup over SBP, including the block merge phase, ranges from 1.5 × to 5.7 × for A-SBP, and 0.9 × to 2.6 × for H-SBP.

H-SBP does lead to slight slowdown over SBP on S23 and matches the runtime of S21. However, it provides speedups over SBP on all other graphs, and we do not observe a consistent trend between graph size or density and speedup with either A-SBP or H-SBP. It is also interesting to note that the speedups incurred with A-SBP hold true whether or not A-SBP converges, and we see the same speedup behavior even in the 6 redacted graphs where all 3 algorithms fail to converge.

5.3 Accuracy results on real-world graphs

On real-world graphs, our results show that H-SBP matches the accuracy obtained by SBP on all 12 graphs, both in terms of Modularity and Normalized MDL, as seen in Fig. 5. Given that these graphs cover a large number of domains, this strongly suggests that the parallel H-SBP algorithm is a viable option for speeding up community detection in real-world scenarios.

On the p2p-Gnutella31 graph, all 3 algorithms fail to converge. The algorithms result in a $MDL^{norm} > 1$, suggesting that it has little-to-no inherent community structure. There could be a number of reasons for this, including too many between-community edges for a graph of its density, or all vertices in the graph being sampled from the same structural community.

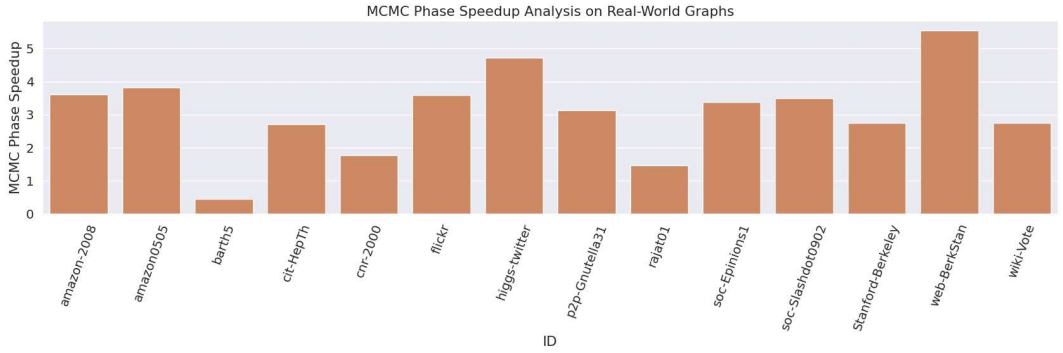


Figure 6: MCMC phase speedup results with SBP and H-SBP on the real-world graphs. H-SBP provides a speedup over SBP on all but one of the graphs tested, with the highest speedup being 5.6 × on the web-BerkStan graph.

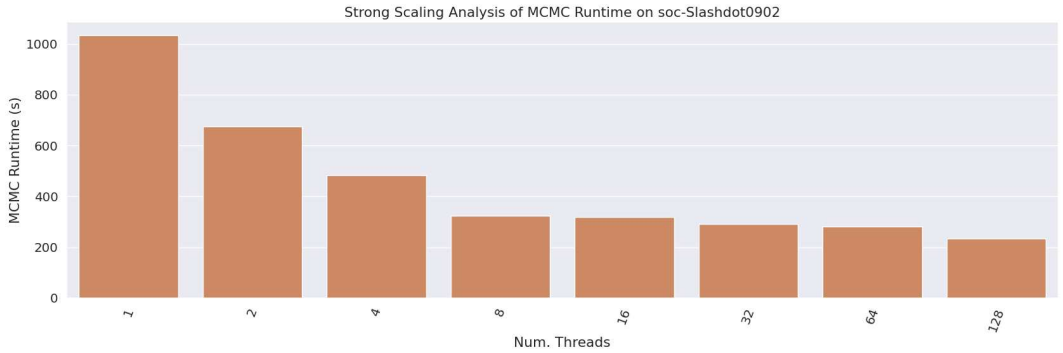


Figure 7: Strong scaling results of the MCMC phase runtime using H-SBP on the soc-Slashdot0902 graph. H-SBP benefits from more threads in all cases, although the benefits begin to taper off around the 16 thread mark.

5.4 Speedup results on real-world graphs

H-SBP provides a speedup over SBP on all but one of the real-world graphs tested, with the highest speedup being 5.6 × on the web-BerkStan graph. However, H-SBP leads to an *increase* in runtime on barth5. On this graph, the number of iterations needed by H-SBP to converge is much higher than in the other real-world graphs (see Fig. 8b in Appendix A), although in general, the difference between the number of MCMC iterations between H-SBP and SBP in real-world graphs appears to be much smaller than in the synthetic graphs. Hence, the generally higher speedup numbers in the real-world graphs when compared to the synthetic graphs. The speedup results are summarized in Fig. 6. The overall speedup over SBP, including the block merge phase, ranges from 0.5 × on barth5 to 4.2 × on higgs-twitter. The overall speedup on soc-Slashdot0902 was higher than on web-BerkStan because the MCMC phase took up more of the algorithm runtime on soc-Slashdot0902 than on web-BerkStan.

5.5 Strong Scaling Results

Our strong scaling experiment shows that, at least until 128 threads, the runtime of H-SBP improves with additional threads, showing that the algorithm is highly parallelizable. The benefit does taper off between the 8 and 16 thread mark. This can be improved with better load balancing, which is a non-trivial endeavor and out of the scope of this paper.

5.6 Discussion

Our results show that in the real-world scenario, H-SBP is a very viable approach to parallelizing a single MCMC chain, and consequently, parallelizing blockmodel inference for community detection. H-SBP matches the result quality of SBP, in terms of NMI, MDL^{norm} and Modularity on all graphs tested, even when including real-world graphs from a number of different domains, including a co-purchasing and a book similarity graph, and a number of graphs from the social and web domains. H-SBP also speeds up the MCMC phase by up to 5.6 × on a majority of the real-world graphs, with the only real-world graph on which it does not provide a speedup being very sparse and having an exceptional increase in the number of MCMC iterations resulting from asynchronous processing.

These speedups occur despite an increase in the number of MCMC iterations resulting from asynchronous processing. Further optimizations, which are out of the scope of this paper, could

improve A-SBP and H-SBP speedup by decreasing the number of MCMC iterations (with a relaxed threshold) or casting the graph to be undirected, enabling data access and storage optimizations for the blockmodel.

6 CONCLUSION

In this work, we parallelize MCMC within the chain in the context of community detection without sacrificing accuracy. We implement and study two variants of stochastic block partitioning (SBP), an agglomerative community detection algorithm based on stochastic blockmodel inference, which uses MCMC techniques to move vertices between communities. The first of these variants is asynchronous SBP (A-SBP), a variant of SBP based on the parallel asynchronous Gibbs algorithm. To the best of our knowledge, this is the first work that studies asynchronous Gibbs in the context of community detection. We show that A-SBP can match the result quality of SBP on many synthetic graphs, with the added benefit of significant MCMC phase speedups of up to 7.6 × due to its ability to be parallelized. However, identifying when A-SBP will fail to converge is challenging due to the computational complexity of computing influence α on large graphs.

Based on assumptions on the link between vertex degree and α we develop hybrid SBP (H-SBP), the second SBP variant. H-SBP processes the majority of the graph vertices using parallel asynchronous Gibbs, but reserves a user-defined percentage of influential, high degree vertices to be processed first, and in a synchronous fashion. We then show that H-SBP is a more generally applicable algorithm, in that it is able to match the accuracy of SBP on all tested graphs, even when A-SBP fails to converge, at the expense of some speedup.

Put together, these findings provide a novel direction for scaling accurate community detection. The asynchronous Gibbs algorithm is embarrassingly parallel, needing no locks and virtually no communication between threads. Therefore, a more optimized A-SBP implementation is likely possible, with better load balancing and utilizing data structures that are more suited to repeated reconstruction, as well as possibly a more robust early stopping mechanism to reduce the impact of additional MCMC iterations incurred with A-SBP. Speeding up the graph reconstruction phase would also make batched A-SBP possible, which could potentially provide similar benefits to H-SBP without the need for synchronous processing.

In future work, we plan to study alternative, easy-to-compute heuristic metrics for predicting whether or not A-SBP will converge on large graphs. We also intend to test our approach on larger real-world graphs, as well as on weighted and undirected graphs. Additionally, we plan to study optimizations specific to the A-SBP and H-SBP algorithms, and we plan to study how best to distribute A-SBP and H-SBP in order to further speed up the algorithms and enable processing of graphs that are too large to fit in memory on a single computational node.

REFERENCES

[1] William Aiello, Fan Chung, and Linyuan Lu. 2001. A random graph model for power law graphs. *Experimental Mathematics* 10, 1 (2001), 53–66. https://projecteuclid.org/euclid.em/999188420

[2] Ginestra Bianconi, Paolo Pin, and Matteo Marsili. 2009. Assessing the relevance of node features for network structure. *Proceedings of the National Academy of Sciences* 106, 28 (7 2009), 11433–11438. https://doi.org/10.1073/PNAS.0811511106

[3] Timothy A. Davis and Yifan Hu. 2011. The university of Florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)* 38, 1 (12 2011), 1–25. https://doi.org/10.1145/2049662.2049663

[4] Christopher De Sa, Kunle Olukotun, and Christopher Ré. 2016. Ensuring Rapid Mixing and Low Bias for Asynchronous Gibbs Sampling. *IJCAI International Joint Conference on Artificial Intelligence* 0 (2 2016), 4811–4815. https://doi.org/10.48550/arxiv.1602.07415

[5] Travis Desell, Lee A. Newberg, Malik Magdon-Ismail, Boleslaw K. Szymanski, and William Thompson. 2012. Finding Protein Binding Sites Using Volunteer Computing Grids. *Advances in Intelligent and Soft Computing* 144 AISC, VOL. 1 (2012), 385–393. https://doi.org/10.1007/978-3-642-28314-7_52

[6] Santo Fortunato. 2010. Community detection in graphs. *Physics Reports* 486, 3-5 (2 2010), 75–174. https://doi.org/10.1016/J.PHYSREP.2009.11.002

[7] Sayan Ghosh, Mahantesh Halappanavar, Antonino Tumeo, Ananth Kalyanaraman, and Assefaw H. Gebremedhin. 2018. Scalable Distributed Memory Community Detection Using Vite. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, Waltham, MA, 1–7. https://doi.org/10.1109/HPEC.2018.8547534

[8] W. K. Hastings. 1970. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika* 57, 1 (4 1970), 97–109. https://doi.org/10.1093/biomet/57.1.97

[9] Edward Kao, Vijay Gadepally, Michael Hurley, Michael Jones, Jeremy Kepner, Sanjeev Mohindra, Paul Monticciolo, Albert Reuther, Siddharth Samsi, William Song, Diane Staheli, and Steven Smith. 2017. Streaming graph challenge: Stochastic block partition. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Waltham, MA, 1–12. https://doi.org/10.1109/HPEC.2017.8091040

[10] Edward K. Kao, Steven Thomas Smith, and Edoardo M. Airoldi. 2019. Hybrid Mixed-Membership Blockmodel for Inference on Realistic Network Interactions. *IEEE Transactions on Network Science and Engineering* 6, 3 (7 2019), 336–350. https://doi.org/10.1109/TNSE.2018.2823324

[11] Brian Karrer and M. E. J. Newman. 2011. Stochastic blockmodels and community structure in networks. *Physical Review E* 83, 1 (1 2011), 016107. https://doi.org/10.1103/PhysRevE.83.016107

[12] Glenn G. Ko, Yuji Chai, Rob A. Rutenbar, David Brooks, and Gu-Yeon Wei. 2019. FlexGibbs: Reconfigurable Parallel Gibbs Sampling Accelerator for Structured Graphs. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, San Diego, CA, 334–334. https://doi.org/10.1109/FCCM.2019.00075

[13] Andrea Lancichinetti, Santo Fortunato, and János Kertész. 2009. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics* 11, 3 (3 2009), 033015. https://doi.org/10.1088/1367-2630/11/3/033015

[14] Xu Liu, Jesun Sahariar Firoz, Marcin Zalewski, Mahantesh Halappanavar, Kevin J. Barker, Andrew Lumsdaine, and Assefaw H. Gebremedhin. 2019. Distributed Direction-Optimizing Label Propagation for Community Detection. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Waltham, MA, 1–6. https://doi.org/10.1109/HPEC.2019.8916215

[15] Willie Neiswanger, Chong Wang, and Eric Xing. 2014. Asymptotically Exact, Embarrassingly Parallel MCMC. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, Quebec City, Quebec, Canada, 623–632. https://doi.org/10.5555/3020751.3020816

[16] M. E. J. Newman. 2004. Analysis of weighted networks. *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics* 70, 5 (7 2004), 9. https://doi.org/10.1103/PhysRevE.70.056131

[17] Symeon Papadopoulos, Yiannis Kompatsiaris, Athena Vakali, and Ploutarchos Spyridonos. 2012. Community detection in Social Media. *Data Mining and Knowledge Discovery 2011 24:3* 24, 3 (6 2012), 515–554. https://doi.org/10.1007/S10618-011-0224-Z

[18] Tiago P. Peixoto. 2013. Parsimonious Module Inference in Large Networks. *Physical Review Letters* 110, 14 (4 2013), 148701. https://doi.org/10.1103/PhysRevLett.110.148701

[19] Tiago P. Peixoto. 2014. Efficient Monte Carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E* 89, 1 (1 2014), 012804. https://doi.org/10.1103/PhysRevE.89.012804

[20] Tiago P. Peixoto. 2014. The graph-tool python library. *figshare* (2014). https://doi.org/10.6084/m9.figshare.1164194

[21] Jose B. Pereira-Leal, Anton J. Enright, and Christos A. Ouzounis. 2003. Detection of functional modules from protein interaction networks. *Proteins: Structure, Function, and Bioinformatics* 54, 1 (9 2003), 49–57. https://doi.org/10.1002/prot.10505

[22] Alexander Terenin, Dan Simpson, and David Draper. 2020. Asynchronous Gibbs Sampling. In *International Conference on Artificial Intelligence and Statistics*. PMLR, Palermo, 144–154. https://arxiv.org/pdf/1509.08999.pdf

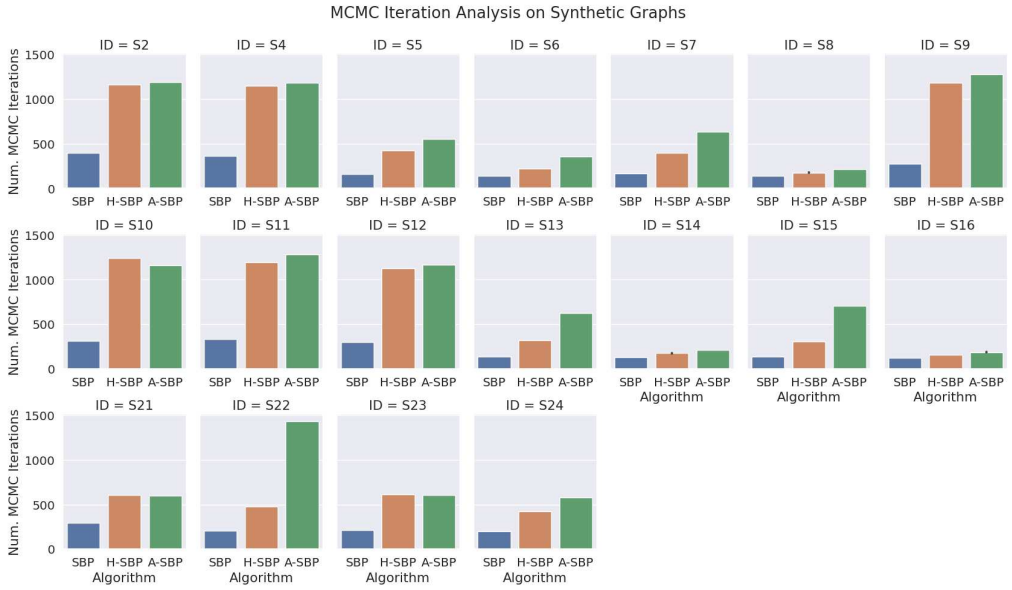
[23] Frank Wanye, Vitaliy Gleyzer, Edward Kao, and Wu-chun Feng. 2021. Topology-Guided Sampling for Fast and Accurate Community Detection. *arXiv* (8 2021). https://doi.org/10.48550/arxiv.2108.06651

[24] Darren J Wilkinson. 2005. Parallel Bayesian Computation. In *Handbook of Parallel Computing and Statistics* (1st ed.), Erricos John Kontoghiorghes (Ed.).

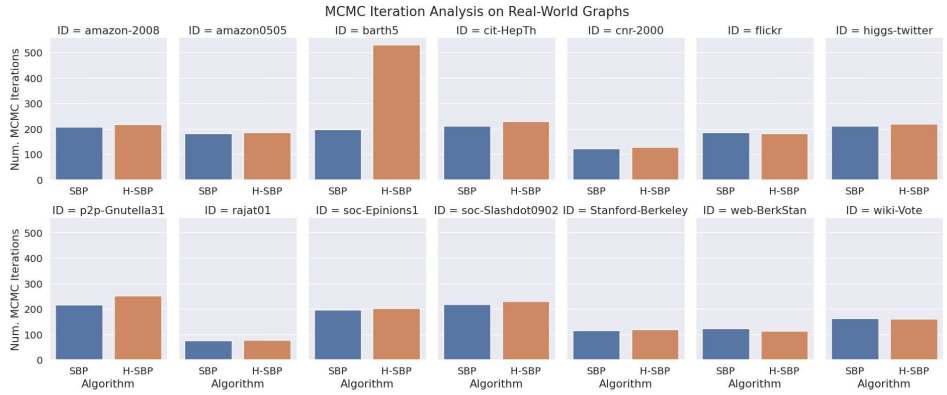
Marcel Dekker/CRC Press, Boca Raton, Florida; London, UK, Chapter 18, 481–512.
http://www.mas.ncl.ac.uk/~ndjw1/docs/pbc.pdf

A NUMBER OF MCMC ITERATIONS PERFORMED BY EACH ALGORITHM

In this section, we present evidence on the impact of asynchronous execution on the number of MCMC iterations needed for convergence. On the synthetic graphs (see Fig. 8a), A-SBP and H-SBP require a significantly higher number of MCMC iterations to converge to a final answer. This occurs regardless of whether or not the A-SBP algorithm ends up matching the accuracy of SBP. Surprisingly, on real-world graphs (see Fig. 8b), the difference in MCMC iterations between H-SBP and SBP is less pronounced, except in the barth5 graph.



(a)



(b)

Figure 8: The number of MCMC iterations required by SBP, H-SBP, and A-SBP to converge to a final answer on the synthetic (top) and real-world (bottom) graphs. H-SBP and A-SBP require significantly more MCMC iterations to converge than SBP does on synthetic graphs. In real-world graphs, the number of MCMC iterations is relatively equal, except in the barth5 graph.

B PAPER ARTIFACT DESCRIPTION APPENDIX

Computational Artifacts: Yes

B.1 Artifact Description Details

The manuscript involves 2 sets of experiments; one where A-SBP, H-SBP and SBP were run on a set of synthetic graphs, and one where SBP and H-SBP were run on a set of real-world graphs. Each run is performed 5 times, and the best result is taken. Artifact 1 contains the code for the 3 SBP variants, as well as the code for generating the synthetic graphs. It also contains sample scripts for generating the graphs and running the experiments.

B.2 Software Artifact Availability

The code for running SBP, A-SBP and H-SBP is available via a public GitHub repository.

B.3 Data Artifact Availability

The exact synthetic datasets that we used in this work are not currently available for download; however the parameters used to generate the datasets as well as the generation script is available via a public GitHub repository. The real-world datasets used are available for download via the SuiteSparse Matrix Collection.

B.4 Artifact 1

GitHub URL:
https://github.com/vtsynergy/Hybrid-Stochastic-Block-Partitioning
Artifact Name: SBP, A-SBP, and H-SBP algorithm code repository.
Relevant hardware details: Experiments were run on the Virginia Tech TinkerCliffs cluster. Sample scripts are included in the repository under the ‘scripts‘ directory. Each experiment run was done on 1 node running 128 threads, each on 1 CPU core.
Operating systems and versions: Red Hat Enterprise Linux Server release 7.9.
Compilers and versions: GCC v8.2.0.
Applications and versions: CMake v3.18.4, OpenMPI v4.1.1.
Libraries and versions: graph-tool v2.29.
Dataset URL: The datasets are available through the SuiteSparse Matrix Collection, at https://sparse.tamu.edu/.