Top-Down Stochastic Block Partitioning: Turning Graph Clustering Upside Down

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

Vitaliy Gleyzer and Edward Kao
{vgleyzer,edward.kao}@ll.mit.edu
MIT Lincoln Laboratory
Lexington, Massachusetts, USA

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

Abstract

Stochastic block partitioning (SBP) is a statistical inference-based algorithm for clustering vertices within a graph. It has been shown to be statistically robust and highly accurate even on graphs with a complex structure, but its poor scalability limits its usability to smaller-sized graphs. In this manuscript we argue that one reason for its poor scalability is the agglomerative, or bottom-up, nature of SBP's algorithmic design; the agglomerative computations cause high memory usage and create a large search space that slows down statistical inference, particularly in the algorithm's initial iterations. To address this bottleneck, we propose Top-Down SBP, a novel algorithm that replaces the agglomerative (bottom-up) block merges in SBP with a block-splitting operation. This enables the algorithm to start with all vertices in one cluster and subdivide them over time into smaller clusters. We show that Top-Down SBP is up to 7.7× faster than Bottom-Up SBP without sacrificing accuracy and can process larger graphs than Bottom-Up SBP on the same hardware due to an up to 4.1× decrease in memory usage. Additionally, we adapt existing methods for accelerating Bottom-Up SBP to the Top-Down approach, leading to up to 13.2× speedup over accelerated Bottom-Up SBP and up to 403× speedup over sequential Bottom-Up SBP on 64 compute nodes. Thus, Top-Down SBP represents substantial improvements to the scalability of SBP, enabling the analysis of larger datasets on the same hardware.

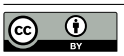
CCS Concepts

• **Mathematics of computing** → **Graph algorithms**; Metropolis-Hastings algorithm; Bayesian computation; • **Computing methodologies** → *Parallel algorithms*; *Distributed algorithms*; Cluster analysis.

Keywords

community detection, graph clustering, stochastic blockmodels, Bayesian inference, asynchronous Gibbs

ACM Reference Format:
Frank Wanye, Vitaliy Gleyzer and Edward Kao, and Wu-chun Feng. 2025. Top-Down Stochastic Block Partitioning: Turning Graph Clustering Upside Down. In *The 34th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '25), July 20–23, 2025, Notre Dame, IN, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3731545.3731589



HPDC '25, Notre Dame, IN, USA

ACM ISBN 979-8-4007-1869-4/2025/07
https://doi.org/10.1145/3731545.3731589

1 Introduction

Much of the real-world data from domains like the World Wide Web [7], computer networks [10], and bioinformatics [16] contains relational information, making graph representation the natural choice for such datasets. In such data, vertices often form groups that are more strongly connected to each other than they are to vertices in other groups. Graph clustering, also known as community detection, is a class of graph analytics algorithms that identifies such groups within a graph [3].

Because optimal graph clustering is an NP-hard problem [3], several approximate heuristics have been developed. These heuristics include modularity maximization [1], which optimizes a measure of intra-cluster connectivity, spectral clustering [26], which leverages eigenvectors to uncover clustering structure, and statistical inference over latent models of the graph structure [12]. Heuristics based on inferential methods have the advantage in terms of accuracy and statistical robustness, particularly on graphs with a complex clustering structure [14, 15].

In this manuscript, we focus on stochastic block partitioning (SBP) [8, 12, 13], an inferential graph clustering algorithm based on inference over a latent stochastic blockmodel [9]. At a high level, SBP starts with a large blockmodel where every vertex is assigned to its own cluster and searches for the optimal clustering configuration. The search is done by alternating between agglomerating (merging) entire clusters together in a model search phase and moving individual vertices between clusters via Markov chain Monte-Carlo (MCMC) inference in a model optimization phase, with both phases of the algorithm being conditioned on the description length (or entropy) of the blockmodel. While the algorithm is accurate due to the rigor of MCMC-based inference, it faces significant challenges in scalability. Both the accuracy and scalability challenges of SBP have been recognized through the formulation of the *Streaming Graph Challenge*, sponsored by IEEE, Amazon Web Services, and MIT, which seeks novel approaches for accelerating the algorithm and thus enabling accurate clustering of large graphs with millions of vertices.

Largely as a result of the *Streaming Graph Challenge*, SBP's scalability challenges have been addressed via data reduction [32], algorithmic refinement [23], and various degrees of parallelization [24, 25, 29, 31]. However, none of these methods adequately addresses the problems inherent in the bottom-up or agglomerative computational approach used in SBP. These problems include large memory requirements for storing and updating a large blockmodel and the vast search space stemming from having a substantial number of clusters, leading to slow convergence.

To address these challenges, we propose to eliminate the bottom-up/agglomerative approach altogether and replace it with an alternative methodology. Specifically, we propose Top-Down SBP, a novel algorithm that improves upon Bottom-Up SBP by starting with a small initial blockmodel, where all vertices are in one cluster, and eliminating agglomerative computation by splitting clusters over time in the model search phase. Top-Down SBP is inherently faster than Bottom-Up SBP, while retaining its accuracy and statistical rigor by continuing to rely on MCMC inference in the model optimization phase as well as having a more stable initial state.

Our main contributions are summarized as follows:

- *A novel Top-Down SBP algorithm that is up to 7.7× faster and up to 4.1× more memory efficient than Bottom-Up SBP.*
- *The acceleration of Top-Down SBP using data reduction and parallelism on shared memory and multi-node systems.* Accelerated Top-Down SBP is up to 15× faster than sequential Top-Down SBP, up to 13.2× faster than the equivalent Bottom-Up SBP implementation, and up to 403× faster than sequential Bottom-Up SBP when processing real-world graphs on 64 compute nodes, all with minimal accuracy loss.
- *An extensive experimental evaluation of the accuracy and scalability of accelerated Top-Down SBP on synthetic and real-world datasets.*

2 Background and Related Work

In this section, we provide background information on graph clustering approaches, the stochastic block partitioning (SBP) algorithm, and past research on accelerating SBP.

2.1 Graph Clustering Approaches

Over the past few decades, a plethora of graph clustering algorithms have been developed as the research community strives to push the boundaries of both accuracy and scalability. Perhaps the most prevalent class of graph clustering approaches are the bottom-up, or agglomerative, methods. Bottom-up approaches start with many clusters and iteratively reduce the number of clusters by merging them together or moving individual vertices out of non-viable clusters. Many of the most common graph clustering algorithms including Louvain [1], Leiden [22], InfoMap [18], Label Propagation [17], and even SBP [12, 13], fall under this category.

Top-down approaches, sometimes called "divisive" algorithms, start with a small number of clusters (usually one) and split them over time. A well-known example is the Girvan and Newman [5] algorithm, which splits clusters by successively removing the edge with the highest betweenness-centrality from the graph. However, such algorithms are rarely used in practice due to the computational cost of cluster-splitting methods. Computing betweenness-centrality, for example, is an $O(n^2)$ operation, while the entirety of the agglomerative Louvain and Leiden algorithms is near-linear in terms of the number of edges.

2.2 Stochastic Block Partitioning

Stochastic block partitioning (SBP) [12, 13] is an agglomerative graph clustering approach based on statistical inference. It clusters vertices by finding the optimal latent model that describes the graph's structure. The latent model of choice in SBP is the stochastic blockmodel (SBM) [9], which describes the graph in terms of the edges between its clusters. In its simplest form, a stochastic blockmodel is a matrix B, where every entry $B_{x,y}$ is the number of edges originating from vertices in cluster x and ending at vertices in cluster y. An optimal blockmodel is one that has the lowest description length [11] H — a measure of the entropy, or compression achieved by the blockmodel. It is typically of the form $H = -ln(B) - L(B|G)$, where $-ln(B)$ is the description length of the blockmodel matrix and clustering assignment, and $L(B|G)$ is the log-likelihood of the blockmodel given the graph. Different formulations of $-ln(B)$ and $L(B|G)$ exist depending on the type of blockmodel and graph being used. For an in-depth discussion on blockmodel entropy, see [11].

The algorithm for optimizing H in SBP starts with each vertex in a separate cluster and consists of two phases; a model search phase, which changes the number of clusters/blocks in the model, and a model optimization phase, which moves vertices between clusters. In the model search phase, entire blocks are merged together based on the resulting change in H. Several merge proposals are generated per block, after which the best proposals across all blocks are sequentially applied to the blockmodel until a user-defined threshold of clusters is reached. The model optimization phase uses an iterative Markov chain Monte-Carlo (MCMC) algorithm called Metropolis-Hastings [6] to optimize the vertex-to-block assignment for a given number of clusters. Each global iteration of SBP runs both phases one after the other, simultaneously performing a golden ratio search for the optimal number of clusters and optimizing the blockmodel for that number of clusters. A high-level overview of the algorithm is given in Algorithm 1.

Algorithm 1: STOCHASTIC_BLOCK_PARTITIONING(G)

```
   Data: Graph G
   Result: Updated Blockmodel B
1  initial_clusters = range(0, G.num_vertices);
2  Blockmodel B(G, initial_clusters);
3  H = compute_H(B);
   /* T is a triplet of blockmodels used to perform
      the golden ratio search.                        */
4  T = [B, B(), B()];
5  do
6      B' = get_next_blockmodel(T);
       /* Performs the model search phase             */
7      B'.merge_blocks();
       /* Performs the model optimization phase       */
8      B'.run_metropolis_hastings();
9      ΔH = compute_ΔH(B');
10     update(T, B', ΔH);
11 while optimal_blockmodel_not_found(B);
12 return get_best_blockmodel(T);
```

SBP is of particular interest because of the high-quality results it provides on graphs with a complex structure [4, 15, 27]. It also provides the statistical benefits of using an inferential graph clustering

method over a descriptive one like modularity optimization, including the ability to identify graphs that lack an intrinsic clustering structure [14].

2.3 Challenges in Accelerating SBP

The benefits of SBP are often outweighed by scalability problems that limit its applicability to large graphs containing millions to billions of edges. This is in large part due to the difficulty in parallelizing MCMC inference in the model optimization phase, which takes up the vast majority of the algorithm's runtime [29]. In recent years, several heuristics have been developed to parallelize this phase with a minimal loss in accuracy [24, 31]. These advances have improved the algorithm's runtime on parallel and distributed systems, but we argue that they do *not* address a core computational limitation of the SBP algorithm; its agglomerative nature.

As has been shown in a parallel *Label Propagation* implementation [19], the initial stages of agglomerative algorithms can consume the majority of the algorithm's runtime. This front-loading of the computation is due to a combination of the large initial search space (V vertices that can be placed in any one of V clusters results in an approximately V^V search space, not accounting for duplicates) and the data access implications of storing and modifying a large clustering model in an algorithm with highly irregular data access patterns. SBP, as shown in [23] and in our own experience with the algorithm, is strongly affected by this phenomenon. While prior work like sampling [32] and aggressive initial merging [23] has partially addressed this problem, these methods provide limited speedup and can lead to accuracy loss with the wrong choice of hyperparameters.

3 Methods

In this section, we describe the sequential Top-Down SBP algorithm and how methods used to accelerate Bottom-Up SBP can be adapted to this novel approach. We also the discuss potential limitations of Top-Down SBP.

3.1 Top-Down SBP

As previously discussed, Bottom-Up SBP has computational weaknesses stemming from the large blockmodel size in the initial iterations. First, the large blockmodel size leads to high memory usage, which in turn causes inefficient cache usage and longer iteration/access times for the blockmodel data structure. This problem is compounded when the blockmodel transpose is stored to speed up column-wise iterations — a technique independently developed in [29] and [23] to accelerate Bottom-Up SBP for directed graphs.

Second, the large blockmodel leads to a large search space for the Markov chain Monte-Carlo (MCMC) phase of the algorithm. Because a given vertex can potentially move to any block/cluster, having many blocks leads to longer mixing times for the MCMC algorithm. It also leads to more vertex moves being performed, which in turn causes more inefficient accesses and modifications to the blockmodel data structure.

Our Top-Down algorithm addresses these issues while retaining the high accuracy and statistical guarantees provided by Bottom-Up SBP via minimizing the blockmodel description length using MCMC-based optimization. By initializing the algorithm with every vertex being placed in one block *and* replacing the block merge phase of Bottom-Up SBP with a block-splitting algorithm, Top-Down SBP retains the MCMC-based optimization while using much smaller blockmodels in the initial iterations. The conceptual differences between the two algorithms are outlined in Figure 1.

3.2 Splitting Clusters

The main challenge in developing a Top-Down SBP approach is developing an algorithm for splitting a cluster in two that is *fast and memory efficient* and that leads to *high-quality splits*. A slow-splitting algorithm could negate any benefits from the reduced MCMC phase runtime. An algorithm that is *not* memory efficient could negate the memory advantages of having a small blockmodel. If the splits are *not* of high quality, then either the MCMC phase would be prolonged in order to "fix" a poor split, or the algorithm would be unable to recover and lead to poor clustering results.

What follows is a high-level overview of our splitting approach (due to space constraints, we refer the reader to Algorithm 3 for an outline of the parallel version of this approach). Similar to the block merge phase, we make several block-split proposals for each block and store the proposal that results in the biggest decrease in the description length H. For each proposal, we extract a subgraph consisting only of vertices and edges contained *within* the given block. The subgraph is then split in two using one of the algorithms described in §3.2.1, and the ΔH between the resulting two-cluster blockmodel and the blockmodel of the subgraph with just one cluster is computed. If this is the most negative ΔH computed for this block so far, then this proposal is stored in the aforementioned vector. Once all proposals have been completed, the vector is sorted and the best splits are applied to the blockmodel until the number of blocks reaches a specified threshold.

3.2.1 Splitting Algorithms. We develop and test several ways to initialize and perform the splitting of a block into two (`propose_split` in Algorithm 3). First, we describe the different splitting methods:

- Random: This serves as the "control" algorithm. All the vertices are randomly assigned to either cluster 0 or 1.
- Snowball: This method aims to localize the two resulting blocks to two topological regions in the graph. First, two vertices are selected to initialize the two blocks. A "frontier" set of neighboring vertices for each block is kept updated throughout the algorithm's runtime. The algorithm iterates between the two blocks and adds one random neighboring vertex from the frontier set to the corresponding block until all vertices are assigned.
- Single snowball: Due to the greedy and random nature of the snowball method, both blocks may lose their locality. This method aims to correct that by focusing on the quality of a single snowball sample. Here, one vertex is selected to initialize block 0. A "frontier" set of neighboring vertices for block 0 is kept updated throughout the splitting runtime. The algorithm selects a random neighboring vertex from the frontier set until either a) the frontier set becomes empty, or b) the frontier set reaches a randomly generated threshold size. All remaining vertices are then assigned to cluster 1.
- Connectivity snowball: This approach counteracts the randomness in the snowball method by assigning vertices to

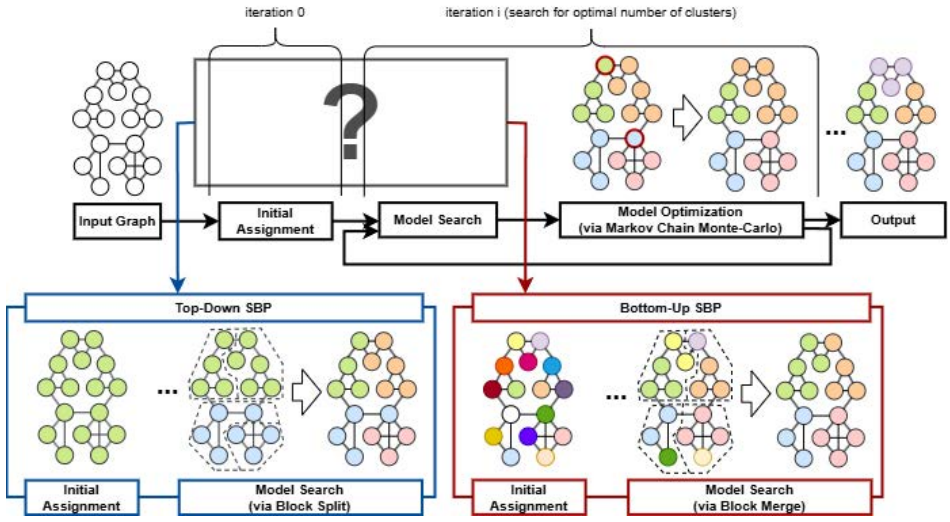


Figure 1: An illustration of the high-level differences between the Bottom-Up SBP algorithm (red, right) and the Top-Down SBP algorithm (blue, left). The steps in black (top) are the same in both algorithms.

clusters based on connectivity. This further biases clusters towards their topological locale. First, two vertices are selected to initialize each cluster. A set of "frontier" vertices that are neighbors of one or both blocks is kept updated throughout the splitting runtime. The algorithm iterates through vertices in the frontier, calculates the number of edges the vertex has adjacent to each cluster, and assigns the vertex to the cluster it is most strongly connected to.

The latter three methods all require one or two vertices to be selected to initialize the splitting algorithm. The following methods for selecting these initial vertices were tested:

- Random: The initialization vertices are selected uniformly at random. This allows the algorithm to explore different regions of the graph.
- High-degree: The initialization vertices are selected to be the two highest-degree vertices in the graph. This ensures that the snowball samples start from "hub" vertices.
- Degree-weighted: This method falls in between the random and high-degree approaches. The initialization vertices are selected at random, with the vertex degrees serving as weights for the random distribution.

As seen in Figure 2, preliminary results on the Graph Challenge graphs (see §4.1) demonstrate that the best accuracy is almost universally achieved using connectivity-snowball splits with random initialization. This is because the connectivity-snowball splitting algorithm greedily maximizes connectivity within each cluster, and random initialization allows the approach to explore a larger variety of possible splits. Henceforth, all results shown are obtained with this combination.

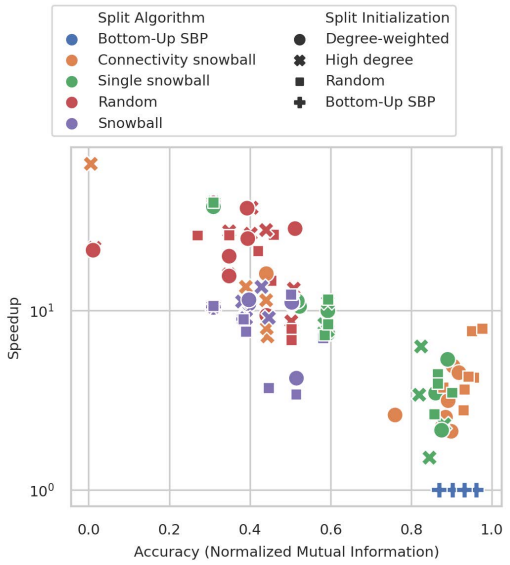


Figure 2: Speedup and accuracy results on Graph Challenge datasets with between 5,000 and 200,000 vertices using all combinations of split algorithms and split initializations. The optimal combination would be in the top-right corner.

3.3 Accelerating Top-Down SBP

In [30], it was shown that parallel computing, distributed computing, and sampling can be used together to accelerate Bottom-Up SBP with a minimal loss in accuracy. As we will describe next, since both methods rely on MCMC, these acceleration techniques can also be adapted for the Top-Down algorithm.

3.3.1 Parallel Computing. As shown in Figure 1, there are two phases to both the Top-Down and Bottom-Up algorithms — a vertex-level phase where vertices move between clusters, and a block-level phase where entire blocks are either merged or split.

The vertex-level phase in [29, 30] is parallelized using a hybrid approach, where a subset of vertices is reserved for sequential processing while the majority of vertices are processed fully asynchronously using a variant of asynchronous Gibbs [21]. More recent work [24] has shown that a batched asynchronous approach can parallelize the vertex-level MCMC computations to greater effect. We therefore use this batched asynchronous approach to parallelize the MCMC phase in our Top-Down (and Bottom-Up) implementations. Pseudocode for this is shown in Algorithm 2.

Algorithm 2: BATCHED_PARALLEL_MCMC(G, B, t, n, x)

```
Data: Graph G, Blockmodel B, double t, int n, int x
Result: Updated Blockmodel B
set i = 0;
repeat
    H = compute_H(B);
    V = shuffle(range(0, G.num_vertices));
    foreach batch b ∈ 0..n do
        start = b * G.num_vertices/n;
        end = start + G.num_vertices/n;
        list of moves M = [[-1] * G.num_vertices];
        #pragma omp parallel for
        foreach index i ∈ start..end do
            vertex v = V[i];
            /* c' = −1 if the move is rejected.    */
            cluster c' = propose_move(v);
            M[v] = c';
        end
        foreach v, c' ∈ enumerate(M) do
            if c' > −1 then
                B.move_vertex(v, c');
            end
        end
    end
    ΔH = compute_H(B) − H;
    i++;
until ΔH < t × H or x ≥ i;
return B;
```

The computations in the block-level phase in Bottom-Up SBP have no inter-block dependencies and can be executed entirely asynchronously, with each thread being responsible for proposing block merges for a distinct set of blocks. Since Bottom-Up SBP starts with an abundance of blocks, there is typically enough work per thread to maximize core utilization. Once all proposals are completed, the merges are sorted, selected, and applied to the blockmodel sequentially.

In Top-Down SBP, the block-level phase also does not have inter-block dependencies and so the blocks can be processed entirely asynchronously. However, the first few iterations have very few blocks, leading to CPU under-utilization in many-core systems. We therefore leverage the OpenMP `collapse` clause to parallelize the block-level phase at the proposal level rather than at the block level, as shown in Algorithm 3. Since there are several proposals per block, this improves core utilization and potentially improves load-balancing due to more fine-grained thread workloads.

Additionally, extracting subgraphs in parallel could lead to inefficient memory usage, particularly in the first few iterations when the number of threads is higher than the number of blocks. In the worst-case scenario, during the first iteration, this would lead to x copies of the entire graph being stored in memory, where x is the number of proposals to be made. Therefore, we pre-extract the subgraphs for each block, allowing threads working on the same block to share the same subgraph data structure memory.

Algorithm 3: PARALLEL_BLOCK_SPLIT(G, B, x, C')

```
Data: Graph G, Blockmodel B, int x, int C'
Result: Updated Blockmodel B
H = compute_H(B);
best_split = [[B()] * B.num_blocks];
best_split_ΔH = [[inf] * B.num_blocks];
subgraphs = [G() * B.num_blocks];
#pragma omp parallel for
for cluster c ∈ B do
    subgraphs[c] = extract_subgraph(G, B, c);
end
#pragma omp parallel for collapse(2)
for cluster c ∈ B do
    repeat x times
        G_c = subgraphs[c];
        B' = propose_split(G_c); // an SBM with 2 blocks
        HB' = compute_H(B');
        ΔH = HB' − compute_null_H(G_c);
        if ΔH < best_split_ΔH[c] then
            best_split_ΔH[c] = ΔH;
            best_split[c] = B';
        end
    end
end
sorted_splits = argsort(best_split_ΔH);
M = B.block_memberships.copy();
N = B.num_blocks;
for index i ∈ sorted_splits do
    if N ≥ C' then
        break;
    end
    M = apply_split(best_split[i]);
    N += 1;
end
return Blockmodel(G, M, N); // build new blockmodel
```

3.3.2 Distributed Computing. EDiSt, a distributed Bottom-Up SBP variant with vertex degree-based load balancing using MPI, was developed in [31]. EDiSt duplicates the graph and blockmodel across each MPI rank and uses all-to-all communication primitives to synchronize blockmodels during the vertex-level and block-level

phases to prevent loss of accuracy due to broken dependencies. We develop a similar implementation for Top-Down SBP.

Once again, the distributed MCMC code is identical for the two algorithmic approaches. After the MCMC vertex moves are proposed for each batch, MPI all-to-all communication synchronizes the moves across MPI ranks. The local blockmodels are then updated independently using accepted vertex moves from all ranks.

However, the communication in the distributed block-level phase had to be refactored for the Top-Down approach. In Bottom-Up SBP, a block merge could be communicated using just two block IDs and the computed ΔH. In Top-Down SBP, we also need to communicate which vertex moved to a new block. Therefore, we communicate two vectors at the end of the block-split phase: a vector of the resulting ΔH values from the best splits computed for each block and a binary vector Y indicating whether or not a vertex stayed in the same cluster (0) or moved to a new cluster (1). Once this information is synchronized across MPI ranks, the splits are applied with the following steps:

(1) Pre-computing new block IDs for every accepted block split.
(2) Iterating through the vertices marked with 1 in the synchronized binary vector Y and changing their block if they belong to a block that has an accepted split.
(3) Rebuilding the blockmodel using updated block memberships.

3.3.3 Sampling. Data reduction through sampling has been shown to accelerate Bottom-Up SBP with a minimal loss in accuracy [28, 30, 32] by partially alleviating some of the same memory usage problems that are addressed using the proposed Top-Down approach. However, Top-Down SBP can also benefit from data reduction, particularly on low-memory systems (storing much of the metadata needed to compute ΔE requires $O(V)$ or $O(E)$ storage space) and distributed systems (data duplication limits the scalability of both Bottom-Up and Top-Down distributed implementations). We therefore implement sampling for Top-Down SBP using the SamBaS framework described in [32].

3.4 Limitations

There are a few limitations of this proposed approach that merit discussion. First, Top-Down SBP will be significantly less effective as the number of identified blocks approaches the number of vertices in the graph. This is because the higher the number of blocks there are for a given input graph, the fewer iterations Bottom-Up SBP has to perform, and the more iterations Top-Down SBP has to perform. This is further compounded by the fact that block merge proposals are faster to compute, requiring only indexing into the blockmodel, while a good block-split proposal requires running a non-trivial sampling algorithm in addition to iterating over the entire graph to extract a relevant subgraph.

Furthermore, while Top-Down SBP can be expected to benefit from parallelization, distributed computing, and sampling, it is likely to benefit less from them than Bottom-Up SBP. This is because a big reason for Bottom-Up SBP's scalability is the large amount of work done per thread in the vertex-level MCMC phase. Top-Down SBP reduces the overall amount of work needed in the MCMC phase, which could highlight the inefficiencies presented by (a) Amdahl's Law in the parallel implementation, (b) all-to-all MPI communication in the distributed communication, and (c) the overhead of performing sampling and fine-tuning the result using the SamBaS framework.

4 Comparative Results

In this section, we describe our experimental setup, present and explain runtime and accuracy results obtained with Top-Down SBP, and compare them with results obtained with Bottom-Up SBP.

4.1 Experimental Setup

We conduct experiments on two sets of graphs; a set of synthetic graphs from the official Graph Challenge [8] collection of datasets and a set of real-world datasets curated from the SuiteSparse Matrix Collection [2]. To stress-test the SBP algorithm, we select only complex synthetic graphs as characterized by high degrees of inter-block connectivity and block size variation. Tables 1 and 2 summarize the selected synthetic and real-world datasets' characteristics, respectively.

Most of the experiments are conducted on the synthetic graphs because they have the ground truth, making accuracy evaluations more straightforward. The real-world datasets are used to measure scalability and ascertain that results on synthetic datasets are generalizable to real-world applications. Hence, experiments on synthetic graphs include sequential and single-node parallel runs but are restricted to at most four compute nodes. On the other hand, experiments on real-world graphs are run on up to 64 compute nodes, but are restricted to parallel and distributed Top-Down and Bottom-Up SBP implementations.

Table 1: Summary of Synthetic Graphs

Num. Vertices	Num. Edges	Num. Clusters
1,000	8,032	11
5,000	51,157	19
20,000	473,329	32
50,000	1,187,682	44
200,000	4,754,406	71
1,000,000	23,772,977	125

Table 2: Summary of Real-World Graphs

Dataset Name	Num. Vertices	Num. Edges
cit-HepPh	34,546	421,534
soc-Slashdot0902	82,168	870,161
web-BerkStan	685,230	7,600,595
amazon0601	403,394	10,162,164
citPatents	3,774,768	16,518,947
eu-2005	862,664	18,733,713
wiki-topcats	1,791,489	28,508,141
wikipedia-20070206	3,515,067	45,013,315

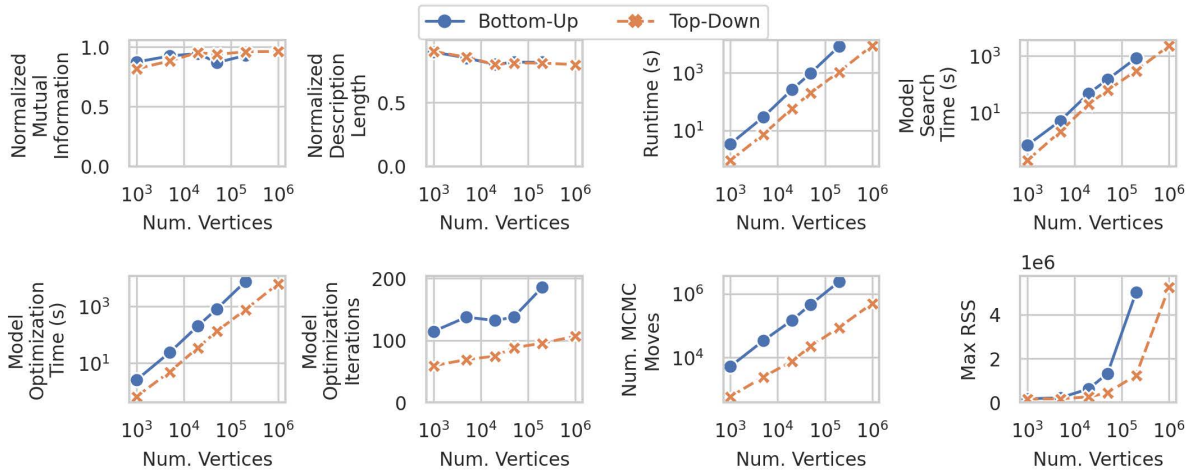


Figure 3: Sequential results comparing Top-Down SBP and Bottom-Up SBP on synthetic graphs.

We evaluate accuracy using two metrics depending on the availability of ground truth clustering labels. The first metric is normalized description length $H^{norm} = \frac{H}{H^{null}}$, where H is the description length of the algorithm's clustering result and H^{null} is the description length of a null blockmodel where every vertex is assigned to the same cluster [29]. It can be used on both synthetic and real-world graphs because it does not rely on ground truth. On synthetic graphs, where ground truth is available, we also measure accuracy using normalized mutual information (NMI) [20] between the clustering result and the ground truth. NMI is defined as $\texttt{NMI} = \frac{I(T;O)}{\sqrt{I(T)}\times\sqrt{I(O)}}$, where $I(T)$ and $I(O)$ are the Shannon entropies of the ground truth and algorithm clusterings, respectively, and $I(T;O)$ is the mutual information between the two clusterings.

To ensure fairness when comparing results across the two algorithms, we integrate the Top-Down SBP code into the common framework shown in Figure 1, using our publicly available Integrated SBP [30] code. This ensures that the only differences in code and optimizations are present in the initial assignment and model search phase portions of the algorithms.

To account for the nondeterministic nature of SBP, we perform each run multiple times and report the average result. Finally, all experiments are performed on the Stony Brook University's Ookami cluster, which consists of 176 A64FX nodes, each with 48 cores spread across four processor groups and 32GB of high-bandwidth memory.

4.2 Single-Threaded Runs

In our first set of experiments, we run Top-Down and Bottom-Up SBP on a single core using the synthetic graphs. As seen in Figure 3, the difference in accuracy between the new Top-Down approach and the old Bottom-Up approach is negligible, empirically demonstrating that the connectivity snowball splitting algorithm leads to high-quality splits. However, Top-Down SBP has significant computational benefits; it is up to 7.7× faster than Bottom-Up SBP (on the 200,000-vertex graph) and uses up to 4.1× less memory. The lower memory usage is evidenced by the lower maximum resident set size (Max. RSS) values for Top-Down SBP and by Top-Down SBP succeeding at processing the 1-million vertex graph while Bottom-Up SBP runs out of memory. Both speedup and reduction in memory usage increase with the size of the graph.

The difference in memory usage is a direct result of the differences between top-down and bottom-up computation: a smaller blockmodel is stored in a smaller matrix and requires smaller intermediate data structures for computing ΔH. The differences in runtime can be attributed to the decrease in model optimization time (6.1×, on average), which correlates with the number of MCMC moves performed in the model optimization phase. In our results, the average difference in number of MCMC moves is 13.8×. This difference can be explained by the fact that Top-Down SBP avoids certain vertex moves by virtue of working with a smaller number of clusters; a vertex move from block a to b in Bottom-Up SBP cannot happen in Top-Down SBP if a and b have not been split yet. A naive upper bound on this difference can be estimated using the expected number of "ignored" vertex move proposals because the proposed cluster was the vertex's current cluster, $p(C_{\texttt{current}})$. As shown in Algorithm 4, proposals for a vertex v are made based on the connectivity of the blocks that are adjacent to v. Thus, the probability of any given block proposal is weighted by inter-block connectivity.

Algorithm 4: MCMC_PROPOSAL(G, B, v)

Data: Graph G, Blockmodel B, vertex v
Result: Cluster c'

```
list n = G.get_neighbors_of(v);
vertex v' = choose_neighbor_random(n);
cluster c = B.get_cluster_of(v');
list nb = B.get_neighbors_with_weights(c);
return choose_neighbor_weighted(nb);
```

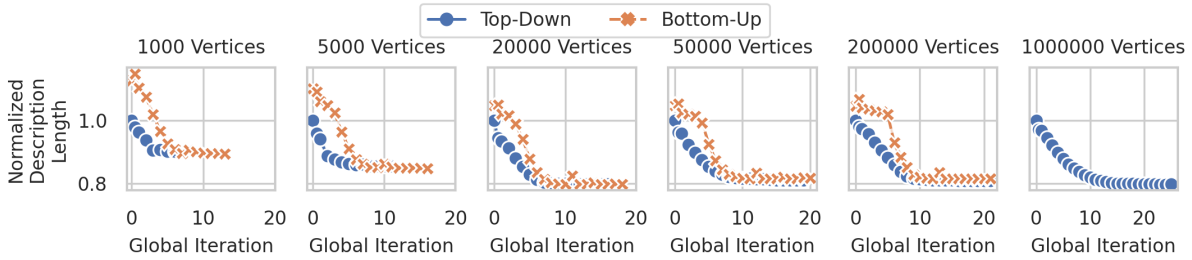


Figure 4: Convergence behavior of sequential Bottom-Up SBP and Top-Down SBP on synthetic graphs.

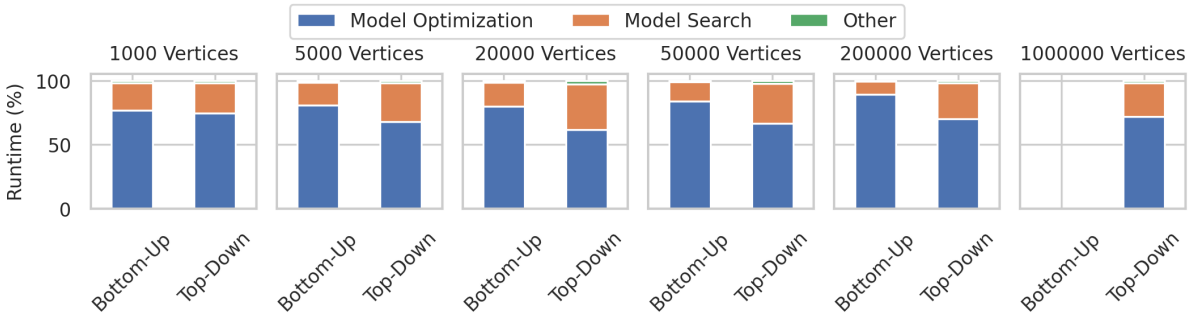


Figure 5: Profiling of sequential Bottom-Up SBP and Top-Down SBP runtime into the algorithm's major phases on synthetic graphs

Following this logic, assuming a sparse graph, the minimum $p(C_{\texttt{current}})$ for Bottom-Up SBP is $1/d^2$ in the first iteration, where d is the average degree of the graph, while the minimum $p(C_{\texttt{current}})$ for Top-Down SBP is $1/2$, also in the first iteration, when the number of blocks is 2. This leads to an upper-bound estimate of the difference in vertex moves of 282.5× on the 200,000-vertex graph. This value is much higher than the observed value of 27.4× because (a) $p(C_{\texttt{current}})$ will grow as both algorithms progress towards the final solution, and (b) the $p(C_{\texttt{current}})$ values for both algorithms will be similar in the latter iterations so long as they lead to similar results. A more accurate estimate, which we leave to future work, could potentially be computed with a more accurate estimate of the number of candidates for an MCMC proposal over time.

Another possible contributor to the decrease in model optimization time in Top-Down SBP can be seen in its convergence behavior (Figure 4). Although both algorithms generally take a similar number of iterations to converge, the starting point for the Top-Down algorithm is significantly better than the starting point for the Bottom-Up algorithm, as evidenced by a lower initial H^{norm}. In fact, the initial H^{norm} for Top-Down SBP is always 1.0, since its initialization is equivalent to a null blockmodel. The starting point for Bottom-Up SBP is therefore worse than a random vertex partition according to the SBM description length, which contributes to a high number of vertex moves.

Finally, we profile the runtime of the two approaches to understand how much time, percentage-wise, is spent in the model optimization, model search, and all other phases (Figure 5). In both algorithms, the model optimization phase takes up the majority of the algorithm runtime. However, Top-Down SBP spends comparatively more time in the model search phase. All other computations put together amount to a negligible portion of SBP runtime.

4.3 Parallel Runs

Next, we run Top-Down and Bottom-Up SBP on all 48 cores of a single processor using the synthetic graphs. Note that the 48 cores are spread over separate processor groups, potentially negatively affecting the reported parallel runtimes. As in the sequential runs, Figure 6 shows little difference in accuracy between Top-Down and Bottom-Up SBP, with the Top-Down approach being significantly (up to 4.9×) faster[1] and being able to process larger graphs on the same hardware.

That said, the parallel speedup recorded for Top-Down SBP over Bottom-Up SBP is lower than its sequential counterpart. Although both algorithms have a similar theoretical parallel speedup as defined by Amdahl's Law, the parallel model search phase with block splits is less efficient than its block merge counterpart. This happens because in earlier iterations of the Top-Down algorithm, there aren't enough blocks to be divided across all 48 threads, leading to low CPU core utilization and load balancing issues until the number of blocks grows sufficiently high.

[1]Notably, the parallel Top-Down SBP runtime on the 1-million vertex graph (20 minutes) is only 1.67× slower than the fastest-recorded Bottom-Up SBP runtime on the same graph (12 minutes) [30], while using 170× fewer cores than the Bottom-Up SBP run. (Caveat: The CPU cores in this paper, Fujitsu A64FX, differ from those in [30], which were AMD EPYC 7702.

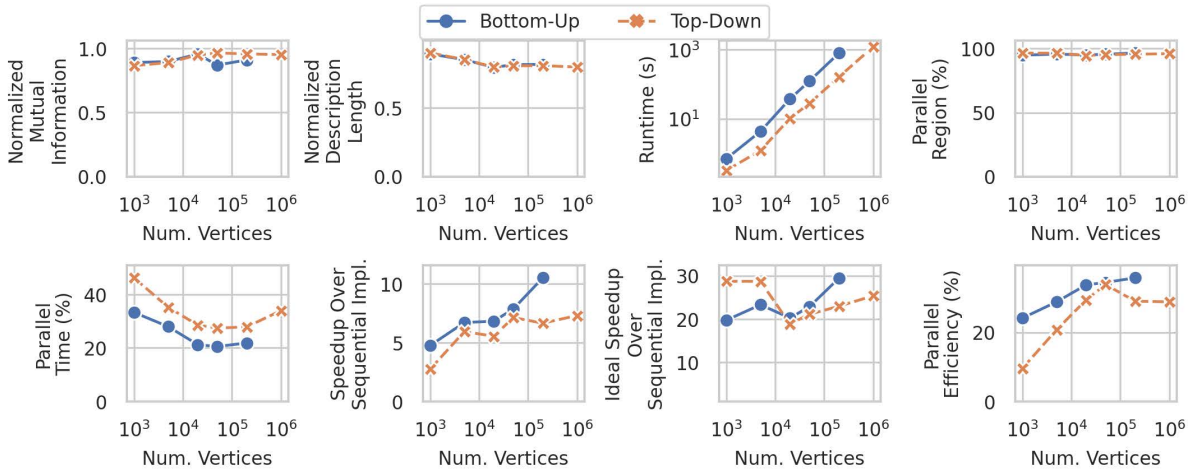


Figure 6: Parallel results on 48 cores comparing Top-Down SBP and Bottom-Up SBP on synthetic graphs.

Additionally, both parallel algorithms are highly inefficient, as can be seen by comparing the speedups achieved over their sequential implementations against the ideal speedups as predicted by Amdahl's Law (see Figure 7). Some contributing factors to this include the overhead of parallel computation, the processor architecture having CPU cores spread across multiple processor groups leading to inefficient shared memory accesses, and the fact that asynchronous Gibbs, used to parallelize the model optimization phase, tends to need more compute iterations to converge.

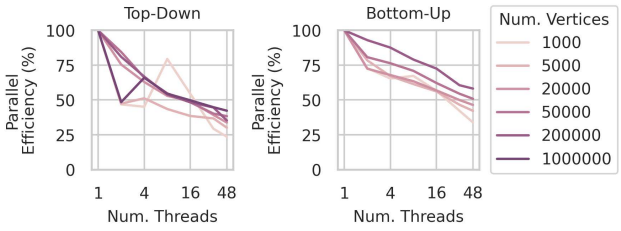


Figure 7: Strong scaling results for both parallel algorithms.

However, significant speedups are unlikely to be obtained by making the parallel implementations more efficient; at 48 threads, less than 50% of either algorithm is being spent in parallel computations. The majority of the time is now spent sequentially modifying the blockmodel with accepted vertex moves and block splits/merges.

4.4 Parallel and Distributed Runs

In our third set of experiments, we run parallel and distributed Top-Down and Bottom-Up SBP on the synthetic graphs using four nodes, or a total of 192 cores. To make better use of the processor groups, we run two MPI tasks with 24 threads each per node, for a total of eight MPI ranks. Figure 8 shows that the parallel and distributed Top-Down approach is up to 6.9× faster than its Bottom-Up counterpart, with a negligible difference in accuracy. However, the parallel and distributed Top-Down algorithm is more likely to be hampered by communication at scale, since it spends a significantly higher percentage of its time performing MPI communication and communicates more data in general.

Despite running on four nodes, speedup over the parallel implementation is less than 2× for both algorithms, leading to a distributed efficiency of under 50%. Furthermore, Figure 9 shows that as the number of compute nodes increases to 64, the distributed efficiency for both algorithms drops to single-digit percentages, with Top-Down SBP having a higher variation between the efficiency on smaller and larger graphs. Interestingly, the highest distributed efficiency using four nodes is obtained by the Top-Down approach on the 200,000-vertex graph, despite the fact that distributed block splits require more communication than distributed block merges. This is because the increase in communication occurs in the block split phase, which calls the communication primitives less often (once per global iteration) than the model optimization phase (multiple times per global iteration). On the 200,000-vertex graph, for example, the most communication-heavy collective in Top-Down SBP is an `AllReduce` from the block split phase, which is only called 208 times. In Bottom-Up SBP, it is an `AllGather` call from the model optimization phase, which is called 3,184 times. At the same time, the decrease in number of model optimization iterations means that there are fewer MPI calls in Top-Down SBP overall, leading to better performance.

One other observation is that for both algorithms, the most time-consuming collective call is *not* the one that sends the most data. On the same graph, in Bottom-Up SBP the `AllGather` call that accounts for 90.2% of the MPI runtime transfers just 0.02% of the data. In Top-Down SBP, the `AllReduce` call that consumes 79.2% of the MPI runtime transfers just 0.07% of the data. It is therefore likely that improving load imbalance in the model search phase could significantly improve scalability for both algorithms.

4.5 Parallel & Distributed Runs with Sampling

Next, we run parallel and distributed implementations of Top-Down and Bottom-Up SBP with sampling on four compute nodes. We use

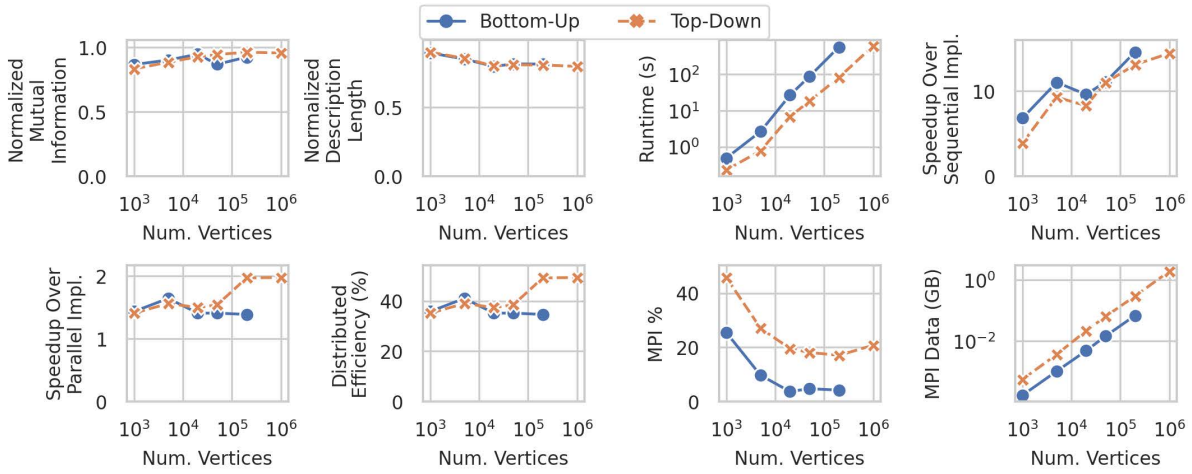


Figure 8: Parallel and distributed results on 4 nodes comparing Top-Down SBP and Bottom-Up SBP on synthetic graphs.

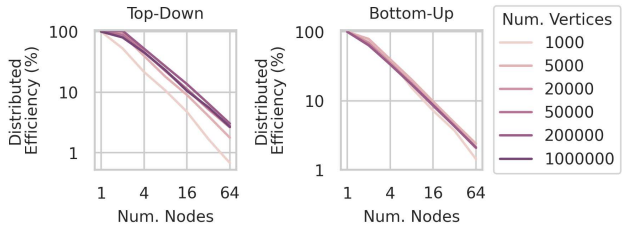


Figure 9: Strong scaling results for both parallel and distributed algorithms.

the uniform random sampling algorithm at a 50% sample size as per [30]. As before, we run two MPI ranks per node with 24 threads per node, for a total of eight MPI ranks and 192 cores.

With sampling, accuracy decreases for both Top-Down and Bottom-Up SBP, particularly on smaller graphs ($\leq$ 5,000 vertices). Distributed Top-Down SBP with sampling is up to 4× faster than the equivalent Bottom-Up algorithm. This is substantially lower than the 6.9× seen without sampling, because Top-Down SBP sees significantly less benefit from sampling, particularly on large graphs. This is evidenced in the higher total time spent performing sampling overhead computations (including sampling, extending partial clustering result to the un-sampled vertices in the graph, and finetuning the clustering result), as well as the higher overall percentage of runtime spent in those operations. Most of this overhead is spent in the finetuning phase and can be attributed to the larger number of clusters C returned by the Top-Down approach, which can be seen in Figure 10 to closely mirror the sampling overhead time.

This increase in C is not necessarily a weakness except in specific use-cases where over-estimating the number of clusters is to be avoided. Bottom-Up SBP tends to underestimate the number of clusters in the synthetic graphs, while the Top-Down approach tends to overestimate it. Since both result in similar NMI and H^{norm} values, the final clusterings are equally valid.

4.6 Real-World Graph Runs

Finally, we run 1–64 compute node scaling experiments using the parallel and distributed implementations of both algorithms, with and without sampling, on the real-world graphs in Table 2. As before, we run two MPI ranks per node with 24 threads each, for a total of up to 128 MPI ranks and 6,144 cores. The single-node runs also schedule parallel and distributed implementations of Top-Down and Bottom-Up SBP, with two MPI ranks running on a single node.

As seen in Figure 11, the overall trends do not differ much from those seen in the synthetic graphs. There is little-to-no difference in accuracy (as measured by H^{norm}) between the Top-Down and Bottom-Up approaches, and the proposed Top-Down algorithm is up to 13.2× faster than the equivalent Bottom-Up implementation. On larger graphs like wiki-topcats, Bottom-Up SBP runs out of memory, while Top-Down SBP does not, even without sampling. Both algorithms exhibit poor strong scaling with less than 50% distributed efficiency at eight nodes and higher. By 32 compute nodes, the efficiency is generally below 10%. Despite the poor efficiency, both algorithms benefit from distributed computation, with 64-node runs completing up to 4.6× faster than single-node runs. Moreover, at 64 compute nodes, Top-Down SBP with sampling is 321.7× faster than the sequential Bottom-Up SBP baseline without sampling on the citPatents graph (the largest graph in terms of the number of vertices), and 403.3× faster than the baseline on the eu-2005 graph (the largest graph in terms of number of edges than Bottom-Up SBP can process).

The wikipedia-20070206 graph is an outlier; the accuracy (H^{norm}) is much better *with* sampling and without sampling the runtime increases with the number of threads. This mirrors the findings in [32], where sampling improved the accuracy of Bottom-Up SBP on some sparse graphs. Top-Down SBP appears to exhibit the same behavior. Without sampling, the runtime increases because the low accuracy is due to Top-Down SBP finding a very low number of clusters (2–17); there is not enough work to effectively distribute across all processes, particularly in the model search phase.

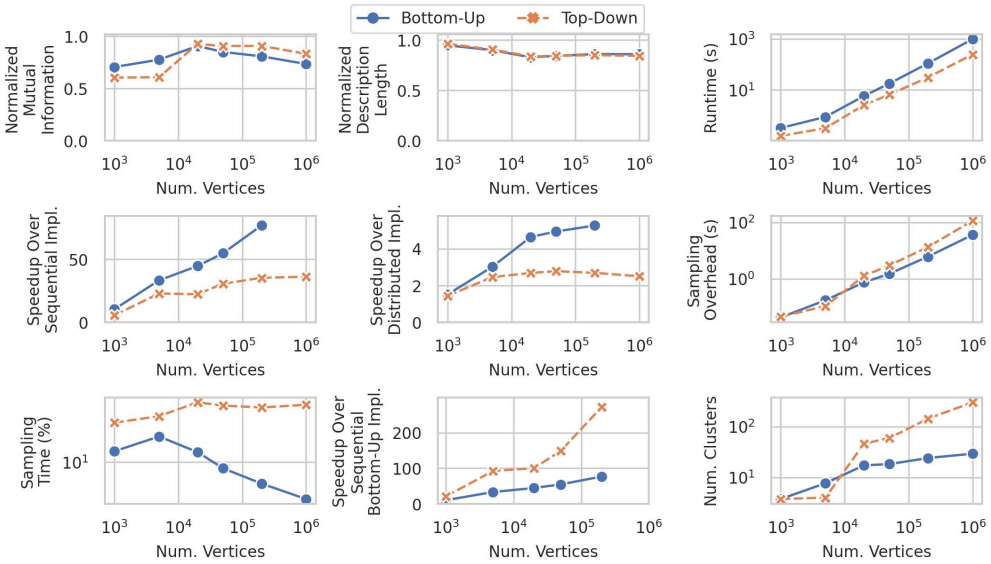


Figure 10: Parallel and distributed results with sampling on four nodes comparing Top-Down SBP and Bottom-Up SBP on synthetic graphs.

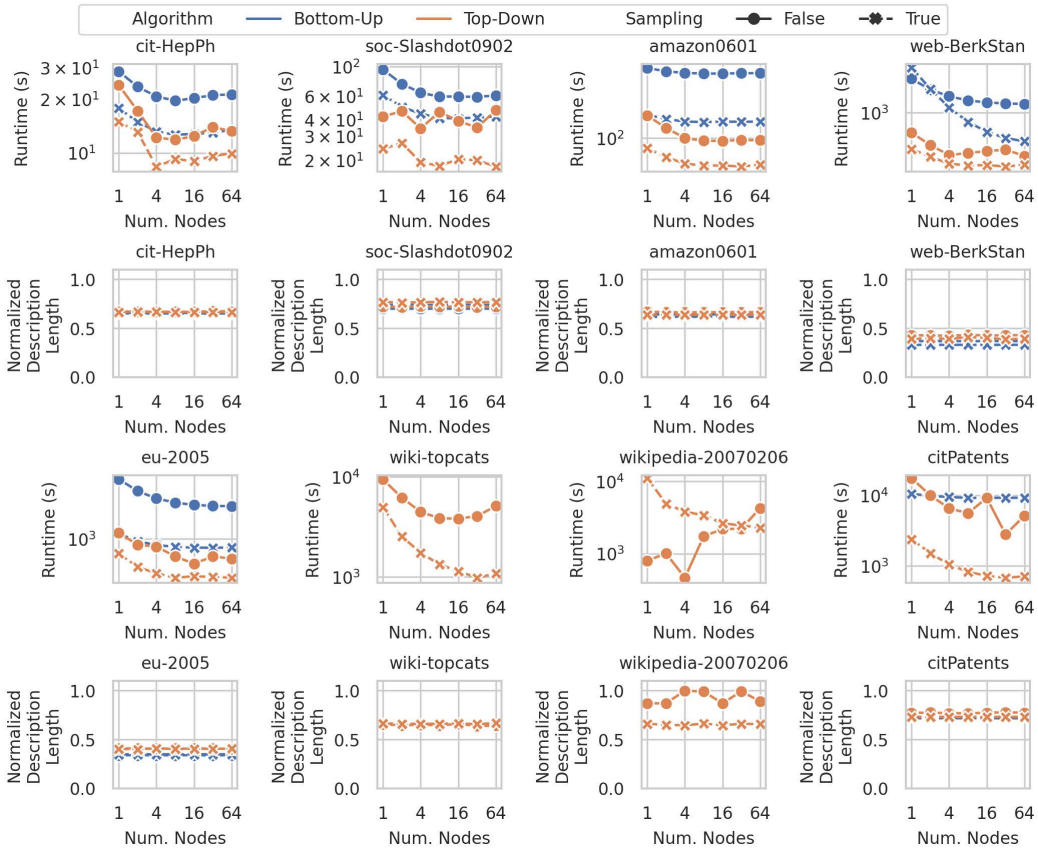


Figure 11: Strong scaling and normalized description length (lower is better) results comparing Top-Down SBP and Bottom-Up SBP on real-world graphs.

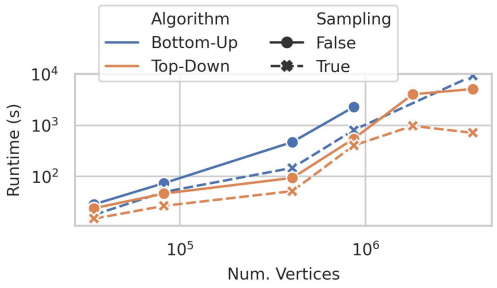


Figure 12: Weak scaling results comparing Top-Down and Bottom-Up SBP on real-world graphs.

In Figure 12, we plot an estimate of the weak scaling properties of both algorithms on the real-world graphs. We start with the citPatents graph with $V = 3,774,768$ vertices running on 64 nodes, and select datasets whose vertex numbers are closest to factors of two of V. Since our datasets do not contain a graph with close to 235,923 vertices, we skip the entry with four compute nodes. The results show predictably poor weak scaling efficiency across all four algorithm implementations. On 16 nodes without sampling, Bottom-Up SBP achieves a weak scaling efficiency of 1.3% while Top-Down SBP achieves 3.7%. With sampling, those values are 2.2% and 3.6%, respectively. This is partially expected because the runtime complexity of SBP and Top-Down SBP is superlinear at $O(E \log^2 E)$, but is also exacerbated by the high overhead of MPI communication and in the case of Top-Down SBP, sampling overhead.

5 Future Work

The major limitation of our Top-Down approach is that it is only faster than Bottom-Up SBP when the number of clusters is small compared to the size of the graph. On graphs with a relatively large number of clusters, Bottom-Up SBP is likely to converge faster than Top-Down SBP. Additionally, our experimental results highlight two limitations of the accelerated variants of both algorithms. The first is that both approaches may lose some accuracy on certain graphs as a result of sampling-based data reduction. The second is that both algorithms exhibit poor parallel and distributed scaling efficiency (under 50% on four compute nodes). Furthermore, since the distributed implementations of both algorithms do not partition the graph or the blockmodel across MPI ranks, the size of graph that they can process is limited by the memory availability of a single compute node, rather than the entire compute system.

Given these limitations, several avenues for future work present themselves. First, exploring alternative data reduction strategies could lead to better accuracy preservation or accuracy preservation at smaller sample sizes. Second, effective data and workload partitioning as well as parallelizing more code regions within the algorithm, such as blockmodel updates, could improve the parallel and distributed efficiency of both algorithms. Finally, extending Top-Down SBP to more complex clustering scenarios, such as overlapping (multi-membership) clusters and time-evolving dynamic graphs, could broaden its applicability in real-world scenarios.

6 Conclusion

In this manuscript, we present Top-Down SBP, a novel graph clustering algorithm that addresses a key limitation of the traditional agglomerative, or bottom-up SBP approach. By reversing the computational paradigm — starting with one cluster and iteratively splitting it instead of starting with many clusters and iteratively merging them — Top-Down SBP reduces both the memory footprint and search space of SBP, particularly in the first few iterations when the number of clusters is close to the number of vertices. As a result, our experimental evaluation shows that Top-Down SBP is both up to 7.7× faster than Bottom-Up SBP *and* uses up to 4.1× less memory than Bottom-Up SBP, enabling it to cluster larger graphs on the same hardware.

SBP has previously been accelerated via data reduction and various levels of parallelism. We adapt these approaches to accelerate Top-Down SBP, yielding up to 4× speedup over accelerated Bottom-Up SBP and up to 220× speedup over sequential Bottom-Up SBP on four compute nodes. We further test the scalability of accelerated Top-Down SBP on real-world graphs, showing up to 13.2× speedup over accelerated Bottom-Up SBP and up to 403× speedup over sequential Bottom-Up SBP on 64 compute nodes.

Acknowledgments

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

The authors would like to thank Stony Brook Research Computing and Cyberinfrastructure, and the Institute for Advanced Computational Science at Stony Brook University for access to the innovative high-performance Ookami computing system, which was made possible by NSF OAC-1927880.

DISTRIBUTION STATEMENT A. Approved for public release. Distribution is unlimited. This material is based upon work supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering. ©2025 Massachusetts Institute of Technology. Delivered to the U.S. Government with Unlimited Rights, as defined in DFARS Part 252.227-7013 or 7014 (Feb 2014). Notwithstanding any copyright notice, U.S. Government rights in this work are defined by DFARS 252.227-7013 or DFARS 252.227-7014 as detailed above. Use of this work other than as specifically authorized by the U.S. Government may violate any copyrights that exist in this work.

References

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* 2008, 10 (10 2008), P10008. https://doi.org/10.1088/1742-5468/2008/10/P10008

[2] 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

[3] 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

[4] Sayan Ghosh, Mahantesh Halappanavar, Antonino Tumeo, and Ananth Kalyanarainan. 2019. Scaling and quality of modularity optimization methods for graph clustering. In *2019 IEEE High Performance Extreme Computing Conference*

(HPEC). IEEE, 1–6. https://doi.org/10.1109/HPEC.2019.8916299

[5] M. Girvan and M. E.J. Newman. 2002. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences of the United States of America* 99, 12 (6 2002), 7821–7826. https://doi.org/10.1073/pnas.122653799

[6] 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

[7] Bernardo A. Huberman and Lada A. Adamic. 1999. Growth dynamics of the World-Wide Web. *Nature* 401, 6749 (9 1999), 131–131. https://doi.org/10.1038/43604

[8] 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

[9] 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

[10] Balachander Krishnamurthy, Jia Wang, Balachander Krishnamurthy, and Jia Wang. 2000. On network-aware clustering of Web clients. *ACM SIGCOMM Computer Communication Review* 30, 4 (2000), 97–110. https://doi.org/10.1145/347057.347412

[11] Tiago P. Peixoto. 2012. Entropy of stochastic blockmodel ensembles. *Physical Review E* 85, 5 (5 2012), 056122. https://doi.org/10.1103/PhysRevE.85.056122

[12] 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

[13] 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

[14] Tiago P. Peixoto. 2023. *Descriptive vs. inferential community detection in networks: pitfalls, myths, and half-truths.* Cambridge University Press, Cambridge. https://doi.org/10.1017/9781009118897

[15] Tiago P. Peixoto and Alec Kirkley. 2023. Implicit models, latent compression, intrinsic biases, and cheap lunches in community detection. *Physical Review E* 108, 2 (8 2023), 024309. https://doi.org/10.1103/PhysRevE.108.024309

[16] 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

[17] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. 2007. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E* 76, 3 (9 2007), 036106. https://doi.org/10.1103/PhysRevE.76.036106

[18] M. Rosvall, D. Axelsson, and C. T. Bergstrom. 2009. The map equation. *The European Physical Journal Special Topics* 178, 1 (11 2009), 13–23. https://doi.org/10.1140/epjst/e2010-01179-1

[19] Christian L. Staudt and Henning Meyerhenke. 2016. Engineering parallel algorithms for community detection in massive networks. *IEEE Transactions on Parallel and Distributed Systems* 27, 01 (1 2016), 171–184. https://doi.org/10.1109/TPDS.2015.2390633

[20] Alexander Strehl and Joydeep Ghosh. 2003. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research* 3, 3 (3 2003), 583–617. https://doi.org/10.1162/153244303321897735

[21] Alexander Terenin, Dan Simpson, and David Draper. 2020. Asynchronous Gibbs sampling. In *International Conference on Artificial Intelligence and Statistics*. PMLR, Palermo, 144–154. https://doi.org/10.48550/arXiv.1509.08999

[22] V. A. Traag, L. Waltman, and N. J. van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports 2019 9:1* 9, 1 (3 2019), 1–12. https://doi.org/10.1038/S41598-019-41695-Z

[23] Ahsen J. Uppal, Jaeseok Choi, Thomas B. Rolinger, and H. Howie Huang. 2021. Faster stochastic block partition using aggressive initial merging, compressed representation, and parallelism control. In *2021 IEEE High Performance Extreme Computing Conference, HPEC 2021*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/HPEC49654.2021.9622836

[24] Ahsen J. Uppal, Thomas B. Rolinger, and H. Howie Huang. 2023. Decontentioned stochastic block partition. In *2023 IEEE High Performance Extreme Computing Conference, HPEC 2023*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/HPEC58863.2023.10363480

[25] Ahsen J. Uppal, Guy Swope, and H. Howie Huang. 2017. Scalable stochastic block partition. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 1–5. https://doi.org/10.1109/HPEC.2017.8091050

[26] Ulrike Von Luxburg. 2007. A tutorial on spectral clustering. *Statistics and Computing* 17, 4 (12 2007), 395–416. https://doi.org/10.1007/s11222-007-9033-z

[27] Frank Wanye and Wu-chun Feng. 2023. On the multi-dimensional acceleration of stochastic blockmodeling for community detection. In *2023 IEEE International Conference on Cluster Computing Workshops, (CLUSTER Workshops)*. Institute of Electrical and Electronics Engineers Inc., Santa Fe, NM, 70–71. https://doi.org/10.1109/CLUSTERWORKSHOPS61457.2023.00030

[28] Frank Wanye, Vitaliy Gleyzer, and Wu-chun Feng. 2019. Fast stochastic block partitioning via sampling. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, Waltham, MA, USA, 1–7. https://doi.org/10.1109/HPEC.2019.8916542

[29] Frank Wanye, Vitaliy Gleyzer, Edward Kao, and Wu-chun Feng. 2022. On the parallelization of MCMC for community detection. In *Proceedings of the 51st International Conference on Parallel Processing*. ACM, New York, NY, USA, 1–13. https://doi.org/10.1145/3545008

[30] Frank Wanye, Vitaliy Gleyzer, Edward Kao, and Wu-chun Feng. 2023. An integrated approach for accelerating stochastic block partitioning. In *2023 IEEE High Performance Extreme Computing Conference, HPEC 2023*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/HPEC58863.2023.10363599

[31] Frank Wanye, Vitaliy Gleyzer, Edward Kao, and Wu-chun Feng. 2023. Exact distributed stochastic block partitioning. In *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 25–36. https://doi.org/10.1109/CLUSTER52292.2023.00010

[32] Frank Wanye, Vitaliy Gleyzer, Edward Kao, and Wu-chun Feng. 2024. SamBaS: sampling-based stochastic block partitioning. *IEEE Transactions on Network Science and Engineering* 11, 3 (5 2024), 3053–3065. https://doi.org/10.1109/TNSE.2024.3358301

Received 6 February 2025; accepted 4 April 2025