Parallel Median Consensus Clustering in Complex Networks

Md Taufique Hussain[1,*], Mahantesh Halappanavar[2], Samrat Chatterjee[2], Filippo Radicchi[3], Santo Fortunato[3], and Ariful Azad[1,*]

[1]Department of Intelligent Systems Engineering, Indiana University, Bloomington, IN USA
[2]Data Sciences and Machine Intelligence Group, Pacific Northwest National Laboratory, Richland, WA USA
[3]Center for Complex Networks and Systems Research (CNetS), Indiana University, Bloomington, IN USA
[*]Corresponding authors: mth@iu.edu, azad@iu.edu

ABSTRACT

We develop an algorithm that finds the consensus among many different clustering solutions of a graph. We formulate the problem as a median set partitioning problem and propose a greedy optimization technique. Unlike other approaches that find median set partitions, our algorithm takes graph structure into account and finds a comparable quality solution much faster than the other approaches. For graphs with known communities, our consensus partition captures the actual community structure more accurately than alternative approaches. To make it applicable to large graphs, we remove sequential dependencies from our algorithm and design a parallel algorithm. Our parallel algorithm achieves 35x speedup when utilizing 64 processing cores for large real-world graphs representing mass cytometry data from single-cell experiments.

1 Introduction

Complex networks often display community structure, where nodes are densely connected within communities but sparsely connected to nodes in other communities[1,2]. Identifying communities (also called clusters or modules) in a network or graph has numerous applications in biology, social science, and various other scientific fields. Over decades of research, many community detection algorithms have been developed, each optimizing different objective functions[3,4]. Due to the absence of a universally accepted definition of communities, different algorithms often identify significantly different clusters on the same graph[5,6]. Moreover, many algorithms are stochastic, leading to variations in clusters depending on input parameters and approximation techniques. These variabilities make it challenging to select one clustering over others.

Instead of choosing a single partition from multiple input partitions, we can combine many input partitions to create a new one. This consensus approach is a valuable way to cluster data, especially when no single metric can determine the best partition. Consensus clustering is a well-established technique in data analysis aimed at finding the median (or consensus) partition[7–9]. The median partition is the one most similar, on average, to all input partitions, based on between-partition similarity measures such as normalized mutual information (NMI), Rand index, adjusted Rand index, Fawlkes and Mallows measure, variation of information, split-join measure, and the Jaccard index. In its standard form, this combinatorial optimization problem is NP-complete[10], making it infeasible to solve for moderate to large datasets. Consequently, existing methods employ various greedy optimizations.

Despite being well-studied in data science, consensus clustering has received less attention in network science. We discuss the existing works on consensus clustering in detail in Section 2. Most existing graph consensus clustering methods overlook the graph's edges and treat the problem as a set partition problem. Even though overlooking the edges of the input graph can potentially produce a consensus, it significantly increases the computational requirement. As a result, most existing methods are not suitable for the large networks commonly encountered today. Moreover, to the best of our knowledge, there has been no study on parallelization strategies of the consensus clustering algorithm which is essential to apply the works on large-scale problems.

In this paper, we develop a novel graph consensus clustering algorithm that minimizes the total distance between the consensus partition and all input partitions. While there are various ways to define inter-partition distance[11], the Mirkin distance is by far the most used in the context of consensus clustering[10] and we use it here as well. For this optimization, we introduce a greedy algorithm that leverages the graph structure for iterative improvement of the consensus. We empirically demonstrate that our structure-aware algorithm produces a consensus partition that reduces the average distance between the consensus and all input partitions. For graphs with known communities, our consensus partition captures the actual community structure more accurately than alternative approaches. Additionally, unlike the graph-agnostic solution introduced by Filkov and Skiena[10], our

algorithm avoids quadratic memory requirements, making it suitable for large graphs. We eliminate sequential dependencies between successive consensus updates and develop a parallel algorithm for multi-core processors. Our parallel algorithm runs 35 times faster than the sequential algorithm when using 64 processing cores.

2 Related Work

The method of integrating various outputs into a single consensus partition is employed across multiple research domains, including machine learning[12], pattern recognition[13], bioinformatics[10], and data mining[14]. Many of these techniques are applicable to diverse data types, including graphs, as the core problem revolves around identifying the optimal partition of the vertex set.

One notable approach to consensus clustering is based on a voting mechanism. In this method, a compilation of cluster labels is created, and the most frequently occurring label is assigned to each element[14–17]. However, a significant challenge arises from the lack of inherent relationships between cluster labels across different partitions. Consequently, the primary difficulty in these approaches lies in matching the clusters from various partitions. This issue is typically addressed by constructing a weighted bipartite graph and employing a matching algorithm to find correspondences between the clusters. When all partitions contain the same number of clusters, achieving consensus through voting becomes straightforward once the clusters are matched across partitions. However, in cases where different partitions have varying numbers of clusters — which is often the norm — the challenge of merging or splitting clusters to reach a consensus emerges. This process is not trivial in voting-based methods.

In another significant approach to consensus clustering, a graph is constructed from the cluster membership of elements in the input partitions. Various methods within this framework create the graph differently and apply graph partitioning algorithms to determine the optimal consensus clustering assignment[12,13,18]. For instance, Strehl and Ghosh[12] proposed a hypergraph construction where each cluster is represented by a hyperedge, followed by the application of a hypergraph partitioning algorithm to find the optimal consensus cluster assignment. They also introduced a variant that constructs a graph representing each cluster from each partition, where vertices correspond to clusters and edges connect vertices if they share any common elements. A graph partitioning algorithm is then applied to identify similar groups of clusters, followed by a voting process to determine the final consensus cluster assignment. Fern and Brodley[13] suggested constructing a bipartite graph that represents all clusters from all partitions on one side and all elements on the other, connecting them if an element belongs to the corresponding cluster. A graph partitioning algorithm is then employed to find the consensus cluster assignment. While the goal of constructing the graph and employing graph partitioning is to reconcile the differences between partitions, this approach does not specifically optimize an objective for achieving consensus.

Another significant approach to the consensus clustering problem involves identifying a solution that represents the average of all input partitions. Methods that utilize this approach primarily focus on minimizing the total distance between the consensus partition and all input partitions[10,19,20]. The partition obtained through this method is often referred to as the median partition[7,10]. In most studies, the Mirkin distance[21] is employed as the metric to optimize, as it provides an intuitive understanding of the impact of moving an element from one cluster to another. However, it has been established that exact optimization of this problem using Mirkin distance is NP-Hard[22,23], leading to the development of heuristic methods. Filkov and Skiena[10] introduced a greedy heuristic that evaluates the feasibility of relocating elements from their current clusters to all other clusters, making decisions based on the most advantageous moves. We have selected this method, referred to as *boem*, as our baseline for experimental evaluation.

First known work of consensus clustering for graphs can be attributed to Lancichinetti and Fortunato[24]. The problem was studied from the perspective of finding a stable partition using single graph clustering algorithm as a single graph clustering algorithm may produce different outputs based on stochastic parameters. The authors proposed applying a single graph clustering algorithm multiple times to the graph and then constructing a $n \times n$ co-clustering matrix from the output where n is the number of vertices. In this matrix, each entry counts the number of partitions in which the corresponding elements are clustered together. They treated this matrix as a new graph and repeatedly applied the same algorithm until the output partitions stabilized. We have chosen this method, referred to as *lf-consensus*, as one of our baselines for experimental evaluation. While this approach effectively finds the consensus of partitions generated by a single clustering algorithm, it struggles to converge when the partitions originate from different clustering algorithms. Furthermore, this method is impractical for large graphs due to the extensive memory and runtime required to handle all possible pairs of vertices. A variant of this work was proposed by Tandon et al.[25], which aims to reduce computational costs; however, the issue of inconsistency remains when dealing with partitions produced by different clustering methods.

If the partitions differ significantly from one another, combining all clustering patterns may result in a solution of potentially lower quality than the original input partitions. This observation has inspired recent research focused on identifying homogeneous groups of input partitions, which can be viewed as a form of clustering among the partitions[26–28]. Rather than seeking a single consensus partition, these methods typically select a representative for each homogeneous group, which serves as a consensus for that group. For instance, Kirkley and Newman[27] frame the problem as one of describing the input

partitions with minimal information. Instead of describing each input partition individually, the authors suggest selecting a few representative partitions and describing each input partition in relation to the most similar representative. We have also used this work, referred to as *kirkley-newman*, for our experimental evaluation.

3 Methods

3.1 The consensus clustering problem

Let $G = (V, E)$ be a graph with a set of n vertices V and a set of m edges E. A partition or clustering of G is a collection of non-overlapping clusters, wherein each vertex is exclusively assigned to a single cluster. Let $\mathbf{P}_1, \mathbf{P}_2, ..., \mathbf{P}_k$ denote k partitions of the graph. Consequently, a vertex v is included in k clusters, one cluster from each partition. Let $d(\mathbf{P}_i, \mathbf{P}_j)$ represent the distance between the partitions $\mathbf{P}_i$ and $\mathbf{P}_j$.

Let $\mathbf{C}$ represent a partition of the graph that aggregates k input partitions by capturing their shared patterns. This type of partition is commonly referred to as a consensus partition, which can be determined by optimizing various objectives. In this paper, we aim to identify a median consensus partition, $\mathbf{C}^*$, that minimizes the sum of the distances between the consensus partition and all input partitions:

$$\mathbf{C}^* = \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{k} d(\mathbf{C}, \mathbf{P}_i), \tag{1}$$

In the literature, several measures of $d(\mathbf{P}_i, \mathbf{P}_j)$ have been discussed including the Rand index, Adjusted Rand index, Fawlkes and Mallows measure, Mirkin distance, variation of information, split-join measure, and the Jaccard index [11]. In our algorithm, we aim to minimize the Mirkin distance that measures disagreements between two partitions. The Mirkin distance between two partitions $\mathbf{P}_i$ and $\mathbf{P}_j$ counts (twice) the number of vertex pairs (u, v) that are grouped together in one partition but separated into different clusters in the other partition.

An alternative formulation of the optimization problem. Let $\sigma_v(\mathbf{c})$ be an indicator variable that denotes the cluster $\mathbf{c}$ to which vertex v belongs.

$$\sigma_v(\mathbf{c}) = \begin{cases} 1 & \text{iff } v \text{ is in cluster } \mathbf{c} \\ 0 & \text{otherwise} \end{cases}$$

Next, we define $\delta_{uv}(\mathbf{P})$ to denote the co-clustering distance between two vertices u and v in partition $\mathbf{P}$ as follows.

$$\delta_{uv}(\mathbf{P}) = \prod_{\mathbf{c} \in \mathbf{P}} [1 - \sigma_u(\mathbf{c})\sigma_v(\mathbf{c})] = \begin{cases} 0 & \text{iff } u \text{ and } v \text{ are co-clustered in partition } \mathbf{P} \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

Here, we defined $\delta_{uv}(\mathbf{P})$ as a co-clustering distance instead of a similarity measure used by Filkov and Skienna[10]. Then, the total co-clustering distance δ_{uv} between u and v is

$$\delta_{uv} = \sum_{i=1}^{k} \delta_{uv}(\mathbf{P_k}). \tag{3}$$

According to this definition, δ_{uv} counts the number of partitions in which (u, v) has not been clustered together. Fig. 1 shows an example graph along with co-clustering distances for different vertices. We call it the inter-cluster disagreements. As the Mirkin distance counts the total number of disagreeing pairs of vertices, for this distance metric, minimizing equation (1) is equivalent to minimizing intra-cluster disagreements and maximizing inter-cluster disagreements. Hence we represent our optimization problem of equation (1) as follows:

$$\mathbf{C}^* = \underset{\mathbf{C}}{\operatorname{argmin}} \sum_{uv} [k - 2\delta_{uv}]\ \delta_{uv}(\mathbf{C})\ . \tag{4}$$

The complete derivation of equation (4) from equation (1) is provided in the appendix.

3.2 Naïve optimization and limitations

Solving equation (4) exactly poses an NP-complete problem for $k > 2$ due to the exponential number of potential consensus partitions[10]. Filkov and Skiena[10] developed a greedy algorithm to find a consensus partition from a collection of partitions of points in Euclidean space. Their algorithm pre-computes cluster agreements between every pair of points and stores them in memory to be used in an iterative greedy algorithm. Similarly, the algorithm by Lancichinetti and Fortunato[24] computes

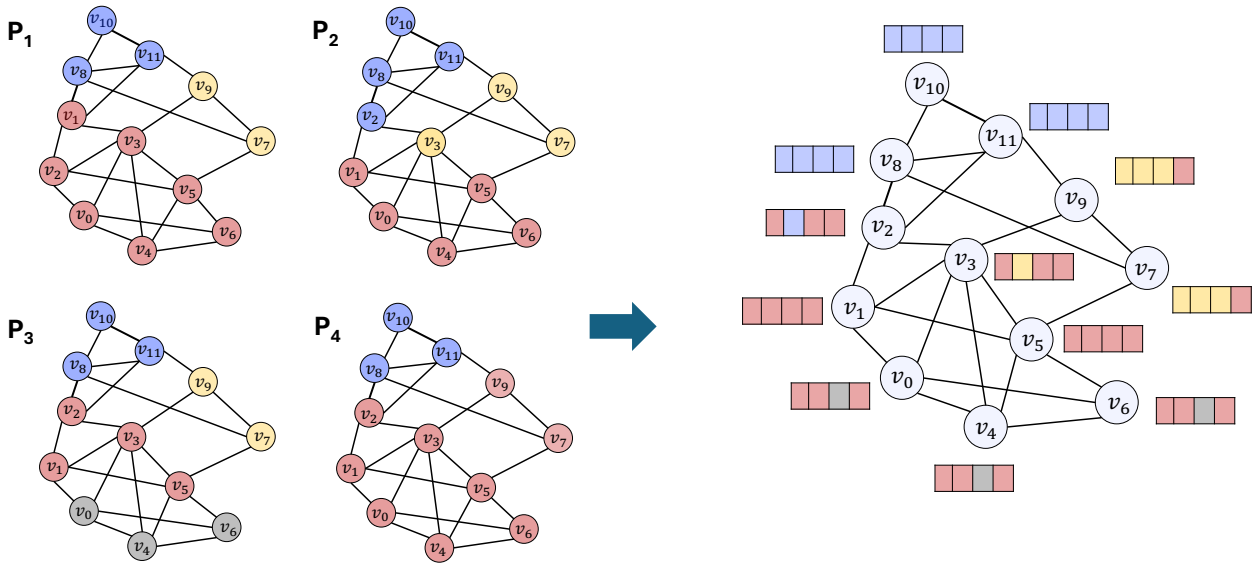


Figure 1. Four partitions $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$, and $\mathbf{P}_4$ of a graph with 12 vertices. Within each partition, vertices belonging to different clusters are shown in different colors. Different cluster assignments for a vertex across various partitions can be represented as a cluster assignment vector. Each element of this vector corresponds to the cluster assignment of the respective vertex in each of the four input partitions. The cluster membership vector for the graph depicted in the left figure is illustrated on the right, where each vertex has a vector size of four, reflecting the four input partitions. The distance between any pair of vertices is the Hamming distance between their vectors. For example, $\delta_{v_0 v_1} = 1$, $\delta_{v_0 v_4} = 0$, and $\delta_{v_3 v_9} = 2$.

agreements between every pair of vertices and uses them in a subsequent community detection algorithm. However, these methods exhibit three significant limitations in the context of graph consensus clustering. **(a) High time and memory requirements:** Computing δ_{uv} for all vertex pairs requires $O(n^2)$ time and memory, where n is the number of vertices. This is too costly and often impractical for large-scale graphs. Additionally, no existing work provides parallel implementations capable of leveraging multi-core processors for quicker consensus generation. **(b) Overlooking graph structure:** Many existing greedy algorithms for graph consensus clustering are structure-agnostic, as they search for a consensus by considering all pairs of vertices. This approach can lead to unnecessary exchanges of vertices among clusters, delaying convergence and compromising the quality of the solution. **(c) Mixing heterogeneous partitions:** When generating different partitions of a single graph, it is often observed that many of them are very different[26–28]. We call the partitions that are similar to each other to be homogeneous and the partitions that are very different from each other to be heterogeneous. Most prior methods construct a single consensus from all input partitions, potentially mixing heterogeneous partitions - which can undermine the quality of the consensus partition. Therefore, it is crucial to identify groups of homogeneous partitions and then create a consensus for each homogeneous group.

In this paper, we aim to overcome these limitations by developing a new graph consensus clustering algorithm that rapidly produces a clustering close to the median of all input partitions.

3.3 Graph-aware consensus clustering

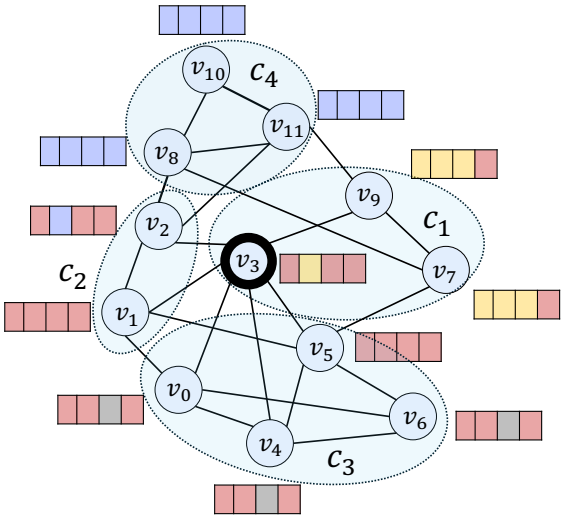
Given any consensus partition $\mathbf{C}$, we design a greedy algorithm that moves vertices across partitions to minimize the overall disagreement among clusters. The algorithm runs in several iterations. In each iteration, a vertex is moved from its current cluster to another cluster only if this movement reduces the overall disagreement. To define this precisely, let the vertex v be in cluster $\mathbf{c}$ in the current consensus, with $\mathbf{c}'$ being the cluster it might move to. Then, the change in the objective function is:

$$\Delta D_{v:\mathbf{c}\to\mathbf{c}'} = \sum_u (k - 2\delta_{uv})\sigma_u(\mathbf{c}) - (k - 2\delta_{uv})\sigma_u(\mathbf{c}') = \sum_u [\sigma_u(\mathbf{c}') - \sigma_u(\mathbf{c})](2\delta_{uv} - k). \tag{5}$$

When $\Delta D_{v:\mathbf{c}\to\mathbf{c}'}$ is negative, moving v to the new cluster $\mathbf{c}'$ reduces the overall disagreements. Hence, v is moved from its current cluster $\mathbf{c}$ to another cluster $\mathbf{c}'$ only when $\Delta D_{v:\mathbf{c}\to\mathbf{c}'} < 0$. Our algorithm does not evaluate all possible clusters $\mathbf{c}'$ to which the vertex v could move. Instead, we only consider those clusters $\mathbf{c}'$ where v has at least one neighbor in the graph.

Thus, the cluster $\mathbf{c}^*(v)$ where the vertex v might move to is computed as follows:

$$\mathbf{c}^*(v) = \underset{\mathbf{c}' \in \mathbf{C}_v}{\operatorname{argmin}} \ \Delta D_{v:\mathbf{c}\to\mathbf{c}'}, \tag{6}$$



$$\sum_{u \in c_1} [k - 2\delta_{uv_3}] = 8 - 2 * (\delta_{v_9 v_3} + \delta_{v_7 v_3})$$
$$= 8 - 2 * (2 + 2) = 0$$
$$\sum_{u \in c_2} [k - 2\delta_{uv_3}] = 8 - 2 * (\delta_{v_2 v_3} + \delta_{v_1 v_3})$$
$$= 8 - 2 * (1 + 1) = 4$$
$$\sum_{u \in c_3} [k - 2\delta_{uv_3}] = 16 - 2 * (\delta_{v_0 v_3} + \delta_{v_4 v_3} + \delta_{v_5 v_3} + \delta_{v_6 v_3})$$
$$= 16 - 2 * (2 + 2 + 1 + 2) = 2$$
$$\Delta D_{v_3 : c_1 \to c_2} = 0 - 4 = -4$$
$$\Delta D_{v_3 : c_1 \to c_3} = 0 - 2 = -2$$
$$\Delta D_{v_3 : c_1 \to c_4} = \text{not considered}$$

Figure 2. A consensus clustering $\mathbf{C} = \{c_1, c_2, c_3, c_4\}$ of the graph shown in Fig. 1. We show the computation to identify the best move for v_3 using equation (5). Since c_4 does not contain a neighbor of v_3, it is not considered in this computation. Based on this calculations, v_3 moves to the cluster c_2 since it reduces the total distance by 4.

where $\mathbf{C}_v$ is the set of all clusters in the current consensus where vertex v has at least one neighbor. The destination cluster $\mathbf{c}^*(v)$ represents the optimal choice for v with respect to the current consensus. However, the order in which vertices are moved influences the set of potential moves for v in the current iteration. Consequently, this greedy moving policy, like any greedy algorithm, does not guarantee an optimal solution.

Fig. 2 shows the calculations for moving a vertex from its current cluster to other clusters. Notice that equation (6) only considers moving vertices along the edges of the graph, similar to the approach employed in greedy community detection algorithms like the Louvain algorithm[29]. Considering the graph structure, at most $O(m)$ moves are considered as opposed to $O(n^2)$ moves in each iteration. Methods oblivious to the graph structure often try to improve this computational complexity but they still have high space complexity ($O(n^2)$). Our method does not have such a high memory requirement as well. We discuss the computation and memory complexity of our algorithm in more detail in Sec. 3.4.

Grouping homogeneous partitions. We take a simple pre-processing step to group the homogeneous partitions. At first, we create a complete graph, where each input partition is treated as a vertex, and the distance between them serves as the edge weight. We used split-join distance[30] as edge weights for its intuitiveness, but other distance metrics can be used as well. Next, we sparsify this complete graph by removing edges whose weights are greater than a threshold λ. This edge removal step creates multiple connected components where each component is composed of relatively homogeneous partitions with inter-partition distances less than λ. We empirically select λ using the elbow method, which involves plotting the number of groups of partitions as a function of λ and identifying the "elbow" point on the curve as the optimal choice. The elbow method is a widely used heuristic for determining the optimal number of clusters in a dataset[31,32]. Once homogeneous groups of partitions are identified, we apply the consensus clustering algorithm independently to each group. This pre-processing step is equivalent to clustering of clustering[27] or meta clustering[33]. We stress that our emphasis here is on the consensus clustering technique, not on the procedure to identify homogeneous sets of partitions. In fact, any method that involves clustering of clustering, as discussed in previous studies[26–28] can be employed before utilizing our consensus clustering approach.

3.4 The algorithm

Given the optimization objectives mentioned in the previous sections, we describe the steps of our algorithm below.

- **Input:** A graph $G(V,E)$ with n vertices and m edges. A set of k partitions of the graph $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_k$.
- **Find homogeneous groups:** Find a homogeneous group by using the method discussed in Section 3.3.
- **Initialization:** We begin with a consensus partition comprising n clusters, where each vertex constitutes a separate cluster: $\mathbf{C} = \{\{v_1\}, \{v_2\}, \ldots, \{v_n\}\}$.
- **Greedy moves:** Find the best move for each vertex v in the graph using equation (6). v is moved from its current cluster $\mathbf{c}$ to $\mathbf{c}'$ only if $\Delta D_{v:\mathbf{c} \to \mathbf{c}'} < 0$.

- **Stopping condition:** The algorithm stops when no vertex can be moved.

Computation and memory complexity. To identify the homogeneous groups of partitions, we create a complete graph with k vertices. This involves comparing $k(k-1)/2$ pairs of partitions, where calculating the split-joint distance between two partitions of n elements requires $O(n \log n)$ time[34]. After pruning for a certain value of λ, this graph contains at most $O(k^2)$ edges, hence detecting connected components on such a graph using any graph traversal technique takes $O(k+k^2)$ time. Overall, finding homogeneous groups of partition takes $O(k^2 n \log n)$ time for a certain value of λ. To find the optimal value of λ using the elbow method, one may need to repeat the process multiple times for multiple values of λ, which usually is a constant.

After detecting the homogeneous groups, we make a consensus of each group. The computational complexity of each iteration of the consensus algorithm depends on the cluster sizes and the number of inter-cluster edges of the consensus partition before the iteration. Let b represent the number of clusters in the current consensus partition. On average, each cluster then contains $\frac{n}{b}$ vertices. To compute the feasibility of each potential move for each vertex, the algorithm computes two terms on the right-hand side of equation (5). . If there are k_h input partitions in the homogeneous group, the computation required for each term is $O(\frac{nk_h}{b})$. If there are $\hat{m}$ inter-cluster edges, the number of potential moves is at most $2\hat{m}$. Hence, based on the assumption of all clusters being of equal size, the total computation required on each iteration is $O(\frac{\hat{m}nk_h}{b})$. One can argue that this complexity can be as high as $O(n^2)$ if $\hat{m}$ is very high and b is very low. For the simple case of same-size clusters, it is easy to see that both of those conditions cannot occur simultaneously. As we start from singleton clusters, during the initial iterations $\hat{m}$ would be high but b would be high as well. As one of the implicit goals of graph clustering is to reduce $\hat{m}$, it decreases in later iterations but b also decreases as small clusters merge with bigger clusters. In practical cases, i.e. graphs with heterogeneous degree and cluster size distributions, this analysis is not so simple. Large clusters having many inter-cluster edges may enhance the computational cost. It might be possible to solve this problem without penalizing the quality too much by appropriate sampling and graph sketching techniques. We leave that for future work. However, we present an empirical evaluation of the runtime in Sec. 4.4.

The median clustering algorithm requires no additional memory beyond what is needed to store the input partitions and the final output. We have two inputs to our algorithm - the graph and the input partitions. Storing the graph takes $O(m)$ space if we store the edges. Storing the input partitions takes $O(nk)$ space if represented as a $n \times k$ matrix, where each column represents a partition and each row represents cluster memberships of a vertex v across different input partitions(as discussed in Sec. 3.1). Hence the memory complexity is $O(m+nk)$. It is possible to reduce time complexity by increasing the space complexity, as discussed in Sec. 3.3. For this work, we choose to make the algorithm applicable to large realistic graphs through parallel computation keeping the memory requirement the same. We discuss the parallel algorithm in Sec. 3.5 below.

The low memory requirement of our median consensus algorithm is particularly useful when dealing with large practical graphs. For instance, in Section 4.4, we present experimental evidence for a large practical graph containing 514k vertices and 12 million edges coming from single-cell experiments. We generate a consensus of 16 different partitions of the vertices. Given that all algorithms utilize 8-byte numbers for each piece of information, our algorithm required 296 megabytes of memory while some other baseline methods required nearly 268 gigabytes of memory hence could not run as the requirement was too high even on a powerful computing resource.

3.5 Parallel algorithm

The algorithm discussed in the previous section identifies the best possible move for every vertex considering the consensus partition at the beginning of the iteration. After some validation checks of the identified moves, all moves are performed together at the end of the iteration. Because one move may change the partition in such a way that some other move may not be optimal anymore, one can argue to identify one best move in an iteration, perform it, and then move to the next iteration. Some other methods, i.e. Filkov and Skienna[10], work in this way. Empirically we observe no quality degradation in our experiment by adopting the approach mentioned in the previous section. In this way, the algorithm takes a few more iterations to converge but they converge to the same quality solution. The solution obtained through this method may differ from the one derived by considering the single best move at a time; however, we observed that the distance from the consensus partition to all other partitions remains the same in both approaches once the algorithm converges. Crucially, this approach eliminates the need for performing moves sequentially one after another, enabling effortless parallelization of the most computationally demanding components.

As discussed in the previous paragraph, computation of the two terms of equation (5) is the most compute-intensive part. Because this computation for one potential move does not depend on any other move within an iteration, we can compute for all potential moves in parallel. In each iteration, we first identify the inter-cluster edges, as these edges provide a list of all potential moves. To assess the feasibility of each move, the algorithm must compute the two components of equation (5). For each vertex and its potential move, the second component depends on the vertex's current cluster, while the first component

depends on the cluster to which the vertex may move. The computational demand of each component is determined by the size of the respective clusters. Since the calculations for each of these components can be performed independently, we identify all necessary terms and assign them to parallel threads. We utilize the cluster sizes to determine the computational load of each term. We assign the terms to a parallel thread in such a way that the total load is balanced. This parallelization strategy requires no additional memory beyond maintaining a list of all potential moves during an iteration, which may be as large as another copy of the input graph. We implement our algorithm in C++ and use the OpenMP programming model to implement the parallel algorithm.

4 Results

We evaluate the performance of our algorithm using three key metrics, which demonstrate its effectiveness both in scenarios with and without community structures and its applicability to large-scale networks:

1. **Accuracy**: We demonstrate the accuracy of our algorithm by testing it against ground truth communities in both synthetic and real-world networks.
2. **Median partition**: We measure how effectively our algorithm identifies a median partition.
3. **Scalability**: We discuss the capability of our algorithm to handle increasingly large input graphs.

4.1 Experiment Settings

4.1.1 Graphs

We use synthetic and real-world graphs to evaluate our consensus clustering method empirically.

Synthetic graphs. We use LFR benchmark graphs of different clustering structures[35]. To pick the parameters of LFR graphs, we take ideas from the original paper and other works that use this benchmark graph. In all cases, we fix the following parameters - minimum degree = 5, maximum degree = 50, minimum cluster size = 5, maximum cluster size = 200, degree exponent $\gamma = 3.0$, and community size exponent $\beta = 1.1$. To generate graphs of different pronounced clustering structures, we vary the mixing parameter μ from 0.1 to 0.7 keeping other parameters fixed. A smaller value of μ represents a more pronounced clustering structure in the graph. To generate graphs of different sizes, we vary the number of vertices from 200 to 125000. We generate these graphs using the Networkx python package[36].

Real-world graphs. We use three real-world graphs from two different application scenarios. *email-Eu-core* is an email communication network containing 1k vertices and 26k edges. *Levine13* and *Samusik* comes from single cell experiments. *Levine13* contains 81k vertices and 1.8 million edges while *Samusik* contains 514k vertices and 12 million edges.

4.1.2 Input partitions

To generate different partitions of the same graph, we design the experiments in two ways: (1) when partitions are slightly different from each other, and (2) when partitions are significantly different from one another.

Slightly different partitions. To generate a set of partitions that are not very different from each other, we use 16 different random initialization parameters of the *louvain* algorithm[29] to get 16 different partitions for the same graph. To generate these partitions, we use the implementation available on the CDLib python package[37]. We choose 16 different randomization parameters equivalent to 16 different vertex orderings for the updates of the algorithm. Narrow black error bars of Fig. 4 show that these partitions are slightly different.

Widely different partitions. To generate a set of partitions that may be very different from each other, we use different parameters of 14 different clustering algorithms to get 38 different partitions of the same graph. We use the following 14 clustering algorithms available in the CDLib python package[37] - *label-propagation*[38], *leiden*[39], *significance-communities*[40], *surprise-communities*[41] , *cnm-greedy-modularity*, *paris*[42] , *louvain*[29] , *infomap*[43] , *walktrap*[44] , *markov-clustering*[45] , *expectation-maximization*[46] , *ricci-community*[47] , *stochastic-block-model*[48], and *spinglass*[49]. We use the default parameter of most of these implementations other than *louvain*, *markov-clustering*, *expectation-maximization*, *spinglass*, and *ricci-community* algorithms as it was not allowed by the implementation. We use different resolution and vertex ordering parameters for *louvain*, different pruning and inflation parameters for *markov-clustering*, randomly chosen expected number of clusters for *expectation-maximization* as well as *spinglass* and different probability distribution parameters for *ricci-community*. The wide black error bars of Fig. 5 indicate that these partitions differ significantly from one another.

4.1.3 Grouping homogeneous partitions

In Sec. 3.3, we discussed our method of grouping homogeneous sets of partitions. Fig. 3 shows the effect of the distance threshold parameter λ. We change the distance threshold value from 1.0 down to 0.05 and observe how many groups are detected and the size of the groups. We observe that as the distance threshold increases, the number of groups decreases. For our experimental dataset, we also observe that while most of these groups contain very few partitions, one group usually

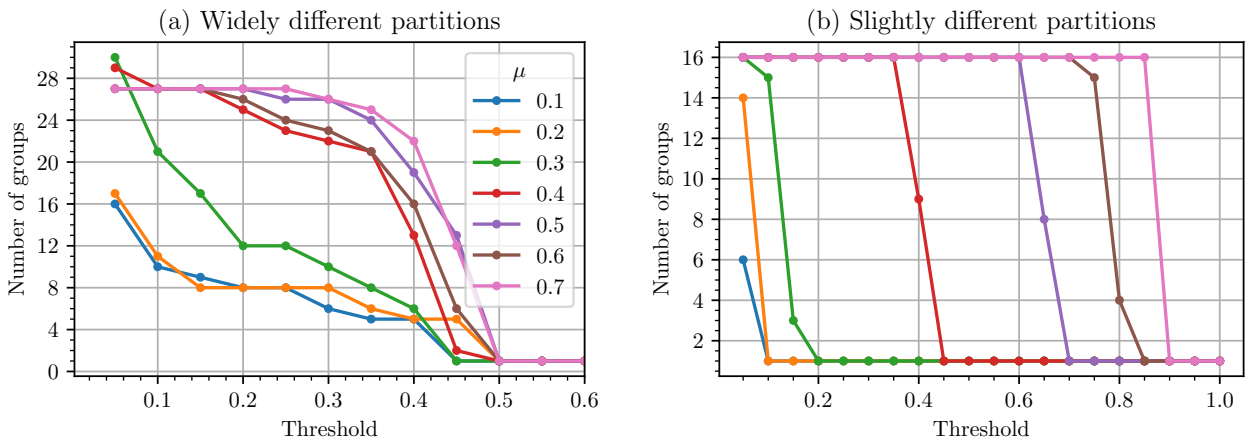


Figure 3. Effect of outlier removal parameter when creating a consensus of (a) widely different partitions and (b) slightly different partitions. LFR benchmark networks with n=5000 are used for this experiment. For (a), 38 different partitions were involved; for (b), 16 different partitions were involved.

is significantly larger than the others. Based on this study, we pick the distance threshold parameter (λ) for subsequent experiments. Since our experimental dataset primarily consists of one large group and several small groups, we select the value of λ at the point where more than one group begins to emerge (e.g., $\lambda = 0.5$ in Fig. 3(a)). We then take the partitions in the largest group and determine their consensus.

4.1.4 Baseline algorithms

We compare our algorithm, *median-consensus*, with the following baseline algorithms.

lf-consensus[24]. We use *lf-consensus* as the baseline algorithm to perform consensus clustering on graphs. In each iteration, the original *lf-consensus* work runs the same algorithm multiple times on the consensus graph generated from the previous iterations. To be consistent with that, in each iteration, we use the same parameters of the same algorithms that have been used to generate the input partitions. We observed that for experiments involving widely-different partitions, *lf-consensus* does not converge. Hence, we omit this method in such cases.

kirkley-newman[27]. We use *kirkley-newman* algorithm to pick the best candidate partition from the inputs as a consensus. *kirkley-newman* algorithm groups the partitions into homogeneous groups and picks one representative partition from each group. To be consistent with our experiments where we make consensus of the largest homogeneous group of partition, we pick the representative of the largest group found by *kirkley-newman* as the consensus partition.

boem[10]. As our method focuses on minimizing the distance from the consensus partition to all input partitions, we compare it with other methods optimizing the similar goal. We use *boem* proposed by Filkov and Skiena as a baseline algorithm from this category. Since *boem* is not designed for graphs, we apply it without utilizing the graph structure.

4.1.5 Evaluation metrics

We evaluate partitions with several different partition comparison metrics. Gates et al. compiled a summary of the performance of the most popular partition comparison metrics, categorized by their desirable properties[50]. Meila proved that it is impossible to have any partition distance metric that satisfies all of the most desirable properties of partition comparison[11]. Therefore, we select three popular partition comparison metrics, each of which satisfies a subset of the most desirable properties - (i) *variation of information distance*[51] that quantifies the amount of information lost or gained when switching from one partition to another, (ii) *split-join distance*[30] that quantifies the minimum number of split and join operations required to transform one partition into the other and (iii) *rand distance*[52] that quantifies the number of pairs of elements that are clustered differently in the two partitions. Please note that *rand distance* is a normalized form of *mirkin distance*, hence they are equivalent. However, *rand distance* can be perceived as probability and hence more intuitive. All three of these measures are true distance metrics and a combination of these measures covers most of the desirable properties of cluster comparison. Our motivation is that observing good performance in terms of all three of these metrics would strengthen the argument.

4.2 Accuracy

When the ground-truth communities are known, we evaluate the results by measuring how closely the consensus partition matches the ground truth. We evaluate this by computing the distance from the ground-truth partition to the other partitions.

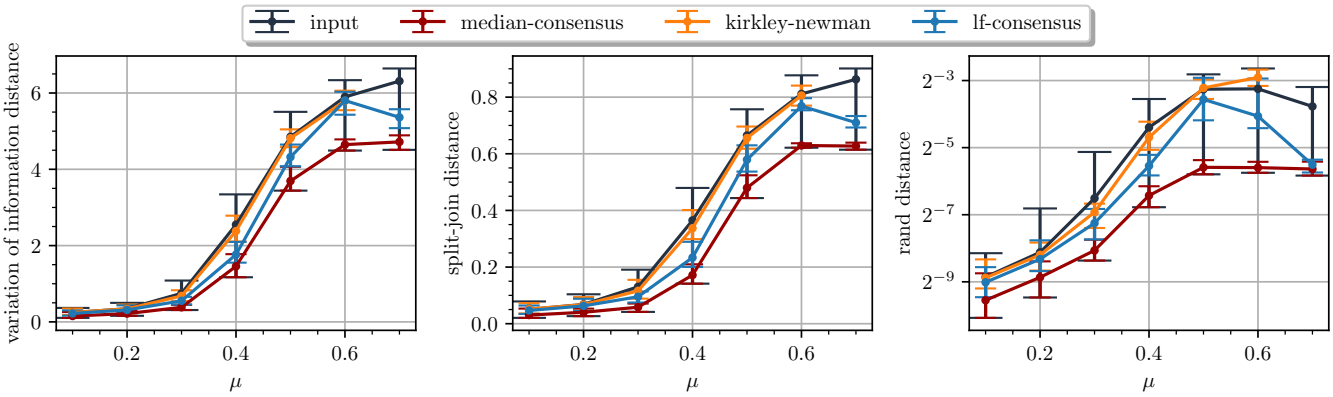


Figure 4. Accuracy of different consensus partitions for LFR benchmark graphs with n=5000, when slightly different input partitions are considered. Here input partitions are obtained by different runs of the Louvain algorithm. Since the ground truth communities are known for LFR benchmark graphs, partition accuracy is measured by the distance from the ground truth community (where lower distance values indicate better accuracy). The black line represents the accuracy of the input partitions used to build the consensus, while the colored lines represent various consensus methods. Points on each line represent the average from 10 sets of experiments, with the corresponding error bars indicating the maximum and minimum values observed across all 10 sets.

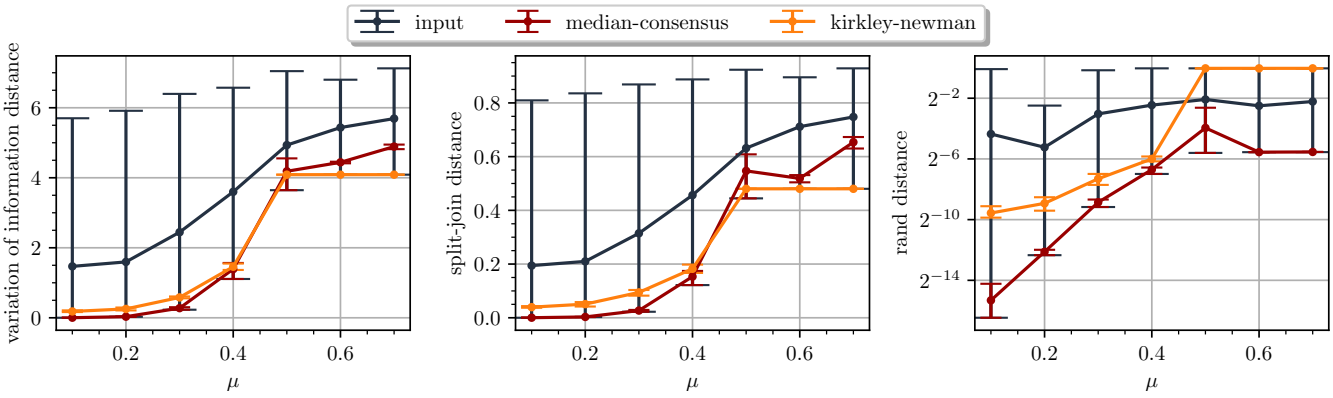
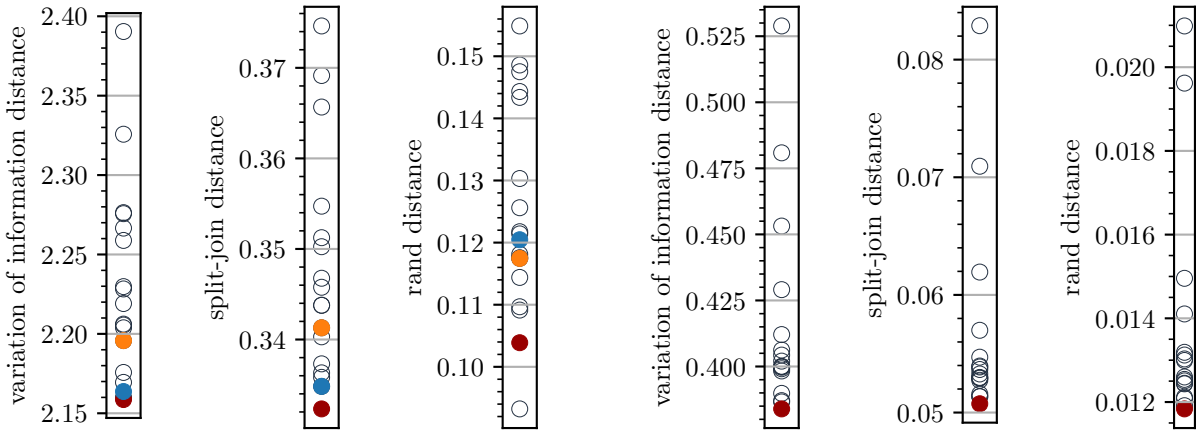


Figure 5. Accuracy of different partitions of LFR benchmark graphs with n=5000 when widely different partitions are considered. Here input partitions are obtained by different parameters of 14 different clustering algorithms. The black line represents the accuracy of the input partitions used to build the consensus, while the colored lines represent different consensus methods. Points on each line represent the average from 10 sets of experiments, with the corresponding error bars indicating the maximum and minimum values observed across all 10 sets. In all cases, a lower value in the Y-axis is better as it signifies the distance to ground truth is smaller.

The partition that is closest to the ground truth is considered to be best in terms of accuracy. The ground-truth partitions of the LFR benchmark graphs are known by design. Two of our real-world datasets also have ground-truth information.

Slightly different partitions of synthetic graphs. We present our evaluation in Fig. 4. Across all evaluation metrics, our *median-consensus* consistently identifies a more accurate partition compared to all baseline consensus methods. Additionally, we observe that in this setting, the consensus produced by *kirkley-newman* closely follows the input partitions. We attribute this to the fact that the input partitions are only slightly different from one another, and since *kirkley-newman* selects a solution directly from the input, it tends to follow the overall trend of the input partitions.

Widely different partitions of synthetic graphs. We present our evaluation in Fig. 5. *lf-consensus* did not converge for this case, so we have excluded it from our analysis. We observe that when the graph exhibits a clear clustering structure ($\mu < 0.5$), across all evaluation metrics, *median-consensus* consistently generates more accurate partitions than other baseline methods across all evaluation metrics. When the graph lacks a significant clustering structure ($\mu > 0.5$), it is difficult to determine which method produced more accurate partitions, as the three evaluation metrics yielded conflicting results. However, since this evaluation is based on synthetic graphs and real-world graphs with appreciable community structure, we can safely focus on the



(a) Accuracy of clustering the *email-Eu-core*[53] communication graph. Grey circles represent the clusters found by different runs of the Louvain algorithm. Red, blue, and orange dots represent the clustering found by *median-consensus*, *lf-consensus* and *kirkley-newman* respectively. In all cases, lower is better.

(b) Accuracy of clustering kNN graph of *Levine13*[54] single-cell data from FastPG[55] pipeline. Grey circles represent the clusters found by different runs of the Louvain algorithm. Red dots represent the partition found by *median-consensus*. In all cases, lower is better. Other baseline methods failed to generate a consensus for this large graph.

Figure 6. Accuracy of consensus clustering in two real-world networks.

cases for $\mu < 0.5$. As the input partitions in this setting differ widely, *kirkley-newman* chooses from the best partitions of the input. Hence, unlike the case of slightly different partitions, it does not stay close to the average input partition trend. By comparing Fig. 4 and Fig. 5, we observe that *median-consensus*, which aggregates partitions from various clustering algorithms, yields more accurate partitions across all three evaluation metrics than the consensus of partitions produced by Louvain alone. Hence, we argue that aggregating partitions from multiple algorithms could be advantageous following outlier removal.

Clustering a real-world communication network. We first demonstrate the applicability of our consensus clustering algorithm on the *email-Eu-core* graph - which is an email communication network of a large European research institute[53]. Each vertex represents an individual belonging to some department of the research institute. A group of individuals belonging to the same department is considered to be a ground truth cluster in the graph. We generate different partitions of the network by applying the Louvain algorithm several times with different random vertex orderings and then apply consensus. We compare the different partitions with the known ground truth information from the metadata. Fig. 6a shows that *median-consensus* produces a more accurate partition than the other baseline methods.

Clustering single-cell experiment data. We also demonstrate the applicability of our consensus clustering algorithm in real-world single-cell experiments. FastPG is a pipeline of clustering millions of cells from single-cell experiments[55]. It has been demonstrated to produce accurate results a magnitude faster than other comparative methods. The superior performance comes from the construction of a k-nearest-neighbor (kNN) graph and applying a variant of the Louvain algorithm to exploit multi-core parallelism. Because Louvain and its parallel variants suffer from producing inconsistent clustering output for different runs, we applied our consensus clustering algorithm to get a consistent result. We take the kNN graph generated by the FastPG pipeline for a dataset coming from human bone marrow cells of healthy donors (*Levine13*)[54]. We run the Louvain algorithm with 16 random vertex orderings to get 16 different partitions. We make a consensus of these 16 different partitions with our *median-consensus* method. As this dataset contains ground truth cell types, we consider each cell type as a cluster and compare it with partitions. Fig. 6b shows that in this way we could recover more accurate clusters from the kNN graph.

4.3 Median partition

In the absence of known ground-truth clusters, we evaluate the consensus partition from a median perspective, as it provides the most impartial assessment. In the following experiments, we disregard the ground-truth information of the LFR benchmark graphs and instead evaluate the consensus based on its proximity to all input partitions. For each consensus partition, we calculate its distance to all input partitions and identify the best consensus partition as the one that minimizes the mean distance to the inputs. For this evaluation, we include *boem* into our consideration as it actively optimizes the goal of finding a median partition. *lf-consensus* did not converge for widely varying partitions, so it is excluded from consideration in that scenario. We present our evaluation in Figs. 7 and 8. Across all distance metrics, we observe that both *median-consensus* and *lf-consensus*

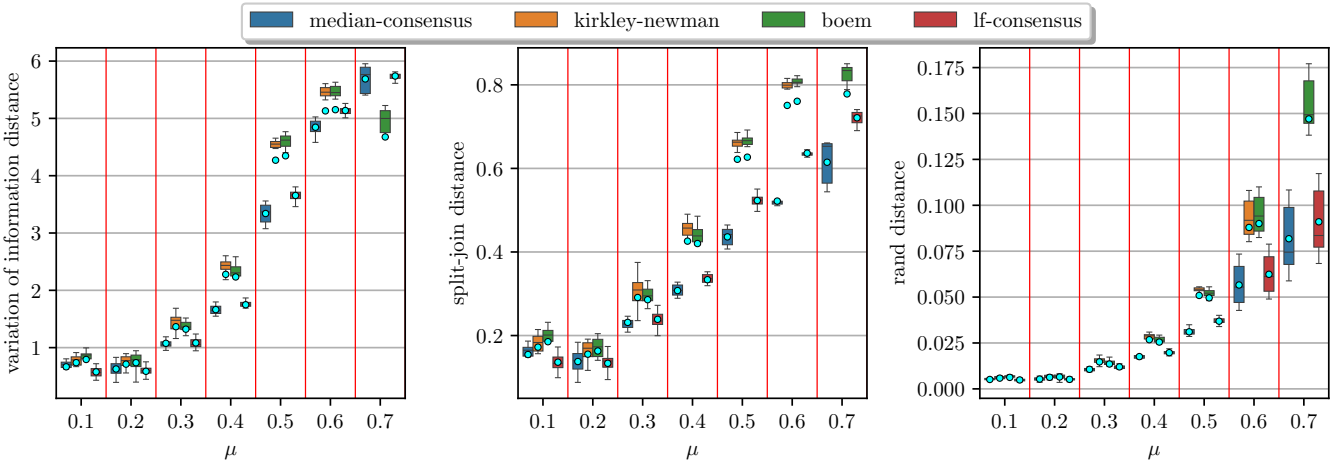


Figure 7. Distribution of distance from consensus partition to all input partitions for slightly different partitions. LFR benchmark graphs with n=5000 are used. Cyan dots represent the mean of each distribution.

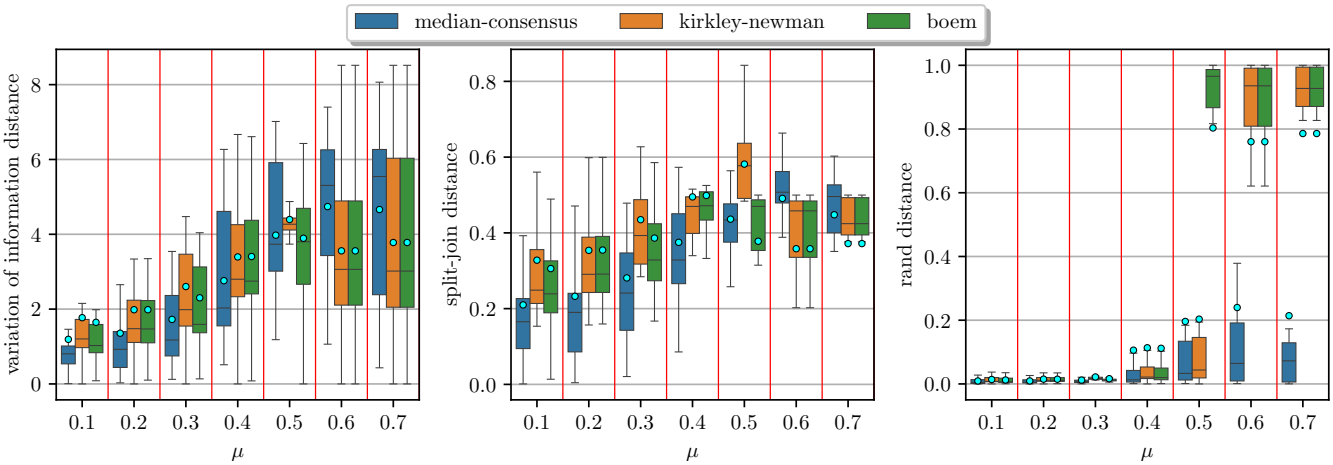
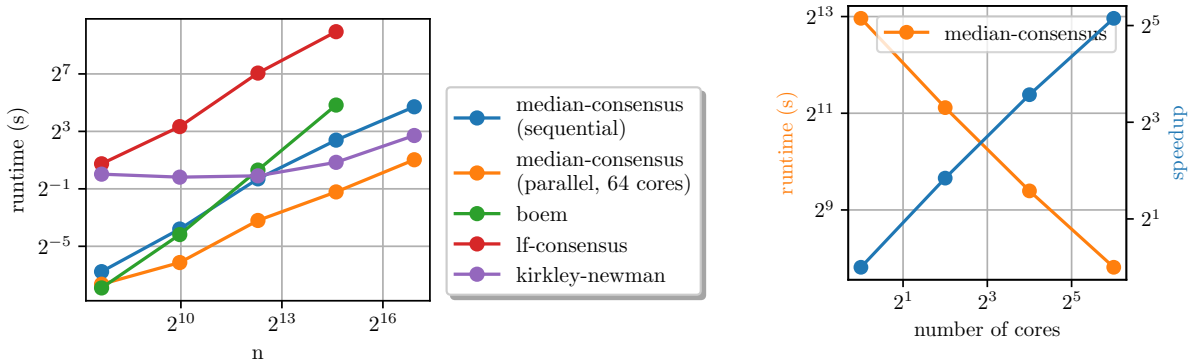


Figure 8. Distribution of distance from consensus partition to all input partitions for widely different partitions. LFR benchmark graphs with n=5000 are used. Cyan dots represent the mean of each distribution.

performed well in finding a good median consensus partition. For the graphs that have a very pronounced clustering structure (very small values of μ), *lf-consensus* performed slightly better than *median-clustering* but the latter method worked well across graphs of different clustering structures as well as widely-different partitions.

4.4 Runtime

We evaluate the runtime of consensus methods in two different ways - (i) runtime on varying-size graphs and (ii) scalability on large practical graphs. For the first case, we generate LFR benchmark graphs of 200 to 125000 vertices with mixing parameter $\mu = 0.4$ and generate different partitions by running the Louvain algorithm multiple times with different vertex orderings. All our experiments were performed on a computing node equipped with AMD EPYC 7783 processor having 64 cores in a single socket and 256 GB of memory. In Fig. 9a we show the runtime of each consensus method as the size of the graph increases. We observe that both *boem* and *lf-consensus* failed to run on the 125000 node graph due to going out of memory. We also observe that even though *kirkley-newman* ran faster than the sequential version of *median-consensus*, by utilizing high-performance computing resources efficiently the parallel version of *median-consensus* ran several times faster than *kirkley-newman* for larger graphs. On a larger real-world graph (Fig. 9b) we observe that *median-consensus* is the better performing algorithm than *kirkley-newman*. We take a kNN graph of 514k vertices and 12 million edges obtained from single-cell experiments[55] and generate different partitions in the same way as above. We observe that while *median-consensus* could successfully find consensus in little more than 2 hours using a single computing core, *kirkley-newman* could not finish in 10 hours. We also observe that the *median-consensus* method shows a linear speedup as we go from 1 computing core to 64



(a) Runtime of different consensus methods on LFR benchmark graphs of varying sizes. *boem* and *lf-consensus* experiments crashed by going out of memory for n=125000.

(b) Consensus clustering runtime and speedup of a kNN graph of *Samusik-all*[54] single-cell data from FastPG[55] pipeline (514k vertices and 12 million edges) as we increase the number of computing resources. Sequential implementation of *kirkley-newman* did not finish in 10 hours.

Figure 9. Runtime and scalability of consensus clustering algorithms.

cores. Utilizing 64 computing cores *median-consensus* produced the same result in less than 4 minutes. It underscores the fact that *median-consensus* method is capable of producing consensus of even larger datasets utilizing the high-performance computing resources of modern days.

5 Conclusions

This paper introduces a new algorithm for finding a consensus among different partitions of a complex network. Unlike previous graph consensus clustering methods, our algorithm aims to find a median of all input partitions. We identified several key advantages of the median partitions generated by our algorithm. (1) A median partition is typically closer to ground truth communities compared to methods that do not search for median partitions. (2) Our median partition provides a consensus with fewer disagreements among input partitions. (3) By using a graph-aware greedy algorithm, we reduce the computational complexity of the process. (4) We eliminate sequential dependencies among vertex movements and develop a parallel algorithm for multi-core processors, achieving linear speedup as the number of processing cores increases. (5) Due to faster computations and reduced memory requirements, our median consensus algorithm can handle large graphs where other methods fail to generate any consensus. (6) We also developed a simple step to group homogeneous partitions before independently finding the consensus of each group. Overall, our work enables finding consensus partitions on large graphs quickly and accurately.

An extension of our algorithm would be to construct consensus clustering for dynamic networks that evolve over time. In such networks, clustering structures change as the network evolves, making consensus clustering an effective approach for identifying core and stable clusters[56–58]. Since our algorithm identifies a median partition that, on average, minimizes distance with all input partitions, it can be readily adapted to dynamic networks if a suitable distance metric is defined for dynamic graphs. We plan to explore this promising research direction in the future. We also plan to explore how the median consensus can be found quickly by using correspondence information between the partitions[59] instead of optimizing a single value.

Data availability

Our implementation of *median-consensus* is located at https://github.com/taufique71/pamcon. We used synthetic and publicly available networks for our experiments. The sources of the networks are mentioned in their respective results sections.

Acknowledgements

MTH and AA are supported by the Applied Mathematics Program of the DOE Office of Advanced Scientific Computing Research under contracts numbered DE- SC0022098 and DE-SC0023349 and by the NSF OAC-2339607 grant. MH, SC, and MTH (internship at Pacific Northwest National Laboratory (PNNL)) were supported by the DOD SERDP RC20-1183 grant titled “Networked Infrastructures under Compound Extremes.” PNNL is operated by Battelle Memorial Institute for the U.S. Department of Energy under Contract No. DE-AC05-76RL01830. SF was partially supported by the National Institutes of

Health under awards U01 AG072177 and U19 AG074879. FR was partially supported by the Air Force Office of Scientific Research under awards FA9550-21-1-0446 and FA9550-24-1-0039. The funders had no role in study design, data collection, and analysis, the decision to publish, or any opinions, findings, conclusions, or recommendations expressed in the manuscript.

Author contributions

The algorithm was developed through collaborative discussions among all authors. MTH implemented the algorithms and conducted the experiments for the paper. All authors contributed to the writing.

Competing interests

The authors declare no competing interests.

References

1. Fortunato, S. Community detection in graphs. *Physics reports* **486**, 75–174 (2010).
2. Porter, M. A., Onnela, J.-P., Mucha, P. J. *et al.* Communities in networks. *Notices Amer. Math. Soc.* **56**, 1082–1097 (2009).
3. Coscia, M., Giannotti, F. & Pedreschi, D. A classification for community discovery methods in complex networks. *Statistical Analysis Data Mining: The ASA Data Science Journal* **4**, 512–546 (2011).
4. Newman, M. E. & Girvan, M. Finding and evaluating community structure in networks. *Physical review E* **69**, 026113 (2004).
5. Dao, V. L., Bothorel, C. & Lenca, P. Community structure: A comparative evaluation of community detection methods. *Network Science* **8**, 1–41 (2020).
6. Fortunato, S. & Hric, D. Community detection in networks: A user guide. *Physics reports* **659**, 1–44 (2016).
7. Vega-Pons, S. & Ruiz-Shulcloper, J. A survey of clustering ensemble algorithms. *International Journal Pattern Recognition Artificial Intelligence* **25**, 337–372 (2011).
8. Nguyen, N. & Caruana, R. Consensus clusterings. In *Seventh IEEE international conference on data mining (ICDM 2007)*, 607–612 (IEEE, 2007).
9. Goder, A. & Filkov, V. Consensus clustering algorithms: Comparison and refinement. In *2008 Proceedings of the Tenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 109–117 (SIAM, 2008).
10. Filkov, V. & Skiena, S. Integrating microarray data by consensus clustering. *International Journal on Artificial Intelligence Tools* **13**, 863–880 (2004).
11. Meilă, M. Comparing clusterings: an axiomatic view. In *Proceedings of the 22nd international conference on Machine learning*, 577–584 (2005).
12. Strehl, A. & Ghosh, J. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal machine learning research* **3**, 583–617 (2002).
13. Fern, X. Z. & Brodley, C. E. Solving cluster ensemble problems by bipartite graph partitioning. In *Proceedings of the twenty-first international conference on Machine learning*, 36 (2004).
14. Topchy, A. P., Law, M. H., Jain, A. K. & Fred, A. L. Analysis of consensus partition in cluster ensemble. In *Fourth IEEE International Conference on Data Mining (ICDM'04)*, 225–232 (IEEE, 2004).
15. Dimitriadou, E., Weingessel, A. & Hornik, K. Voting-merging: An ensemble method for clustering. In *International conference on artificial neural networks*, 217–224 (Springer, 2001).
16. Fischer, B. & Buhmann, J. M. Bagging for path-based clustering. *IEEE Transactions on Pattern Analysis Machine Intelligence* **25**, 1411–1415 (2003).
17. Ayad, H. G. & Kamel, M. S. On voting-based consensus of cluster ensembles. *Pattern Recognition* **43**, 1943–1953 (2010).
18. Iam-On, N., Boongoen, T. & Garrett, S. Lce: a link-based cluster ensemble method for improved gene expression data analysis. *Bioinformatics* **26**, 1513–1519 (2010).
19. Bertolacci, M. & Wirth, A. Are approximation algorithms for consensus clustering worthwhile? In *Proceedings of the 2007 SIAM International Conference on Data Mining*, 437–442 (SIAM, 2007).
20. Ailon, N., Charikar, M. & Newman, A. Aggregating inconsistent information: ranking and clustering. *Journal ACM (JACM)* **55**, 1–27 (2008).

21. Mirkin, B. *Mathematical classification and clustering*, vol. 11 (Springer Science & Business Media, 2013).

22. Wakabayashi, Y. Aggregation of binary relations: algorithmic and polyhedral investigations. *Acta informatica* (1986).

23. Křivánek, M. & Morávek, J. Np-hard problems in hierarchical-tree clustering. *Acta informatica* **23**, 311–323 (1986).

24. Lancichinetti, A. & Fortunato, S. Consensus clustering in complex networks. *Scientific reports* **2**, 336 (2012).

25. Tandon, A., Albeshri, A., Thayananthan, V., Alhalabi, W. & Fortunato, S. Fast consensus clustering in complex networks. *Physical Review E* **99**, 042301 (2019).

26. Calatayud, J., Bernardo-Madrid, R., Neuman, M., Rojas, A. & Rosvall, M. Exploring the solution landscape enables more reliable network community detection. *Physical Review E* **100**, 052308 (2019).

27. Kirkley, A. & Newman, M. Representative community divisions of networks. *Communications Physics* **5**, 40 (2022).

28. Peixoto, T. P. Revealing consensus and dissensus between network partitions. *Physical Review X* **11**, 021003 (2021).

29. Blondel, V. D., Guillaume, J.-L., Lambiotte, R. & Lefebvre, E. Fast unfolding of communities in large networks. *Journal statistical mechanics: theory experiment* **2008**, P10008 (2008).

30. Van Dongen, S. M. *Performance criteria for graph clustering and Markov cluster experiments* (CWI (Centre for Mathematics and Computer Science), 2000).

31. Thorndike, R. L. Who belongs in the family? *Psychometrika* **18**, 267–276 (1953).

32. Ketchen, D. J. & Shook, C. L. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal* **17**, 441–458 (1996).

33. Caruana, R., Elhawary, M., Nguyen, N. & Smith, C. Meta clustering. In *Sixth International Conference on Data Mining (ICDM'06)*, 107–118 (IEEE, 2006).

34. Csardi, G. & Nepusz, T. The igraph software. *Complex syst* **1695**, 1–9 (2006).

35. Lancichinetti, A., Fortunato, S. & Radicchi, F. Benchmark graphs for testing community detection algorithms. *Physical review E* **78**, 046110 (2008).

36. Hagberg, A., Swart, P. & S Chult, D. Exploring network structure, dynamics, and function using networkx. Tech. Rep., Los Alamos National Lab.(LANL), Los Alamos, NM (United States) (2008).

37. Rossetti, G., Milli, L. & Cazabet, R. Cdlib: a python library to extract, compare and evaluate communities from complex networks. *Applied Network Science* **4**, 1–26 (2019).

38. Cordasco, G. & Gargano, L. Community detection via semi-synchronous label propagation algorithms. In *2010 IEEE international workshop on: business applications of social network analysis (BASNA)*, 1–8 (IEEE, 2010).

39. Traag, V. A., Waltman, L. & Van Eck, N. J. From louvain to leiden: guaranteeing well-connected communities. *Scientific reports* **9**, 1–12 (2019).

40. Traag, V. A., Krings, G. & Van Dooren, P. Significant scales in community structure. *Scientific reports* **3**, 2930 (2013).

41. Traag, V. A., Aldecoa, R. & Delvenne, J.-C. Detecting communities using asymptotical surprise. *Physical review e* **92**, 022816 (2015).

42. Bonald, T., Charpentier, B., Galland, A. & Hollocou, A. Hierarchical graph clustering using node pair sampling. *arXiv preprint arXiv:1806.01664* (2018).

43. Rosvall, M. & Bergstrom, C. T. Maps of random walks on complex networks reveal community structure. *Proceedings national academy sciences* **105**, 1118–1123 (2008).

44. Pons, P. & Latapy, M. Computing communities in large networks using random walks. In *Computer and Information Sciences-ISCIS 2005: 20th International Symposium, Istanbul, Turkey, October 26-28, 2005. Proceedings 20*, 284–293 (Springer, 2005).

45. Enright, A. J., Van Dongen, S. & Ouzounis, C. A. An efficient algorithm for large-scale detection of protein families. *Nucleic acids research* **30**, 1575–1584 (2002).

46. Newman, M. E. & Leicht, E. A. Mixture models and exploratory analysis in networks. *Proceedings National Academy Sciences* **104**, 9564–9569 (2007).

47. Ni, C.-C., Lin, Y.-Y., Luo, F. & Gao, J. Community detection on networks with ricci flow. *Scientific reports* **9**, 9984 (2019).

48. Peixoto, T. P. Efficient monte carlo and greedy heuristic for the inference of stochastic block models. *Physical Review E* **89**, 012804 (2014).

49. Reichardt, J. & Bornholdt, S. Statistical mechanics of community detection. *Physical Review E-Statistical, Nonlinear, Soft Matter Physics* **74**, 016110 (2006).

50. Gates, A. J., Wood, I. B., Hetrick, W. P. & Ahn, Y.-Y. Element-centric clustering comparison unifies overlaps and hierarchy. *Scientific reports* **9**, 8574 (2019).

51. Meilă, M. Comparing clusterings—an information based distance. *Journal multivariate analysis* **98**, 873–895 (2007).

52. Rand, W. M. Objective criteria for the evaluation of clustering methods. *Journal American Statistical association* **66**, 846–850 (1971).

53. Leskovec, J., Kleinberg, J. & Faloutsos, C. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)* **1**, 2–es (2007).

54. Weber, L. M. & Robinson, M. D. Comparison of clustering methods for high-dimensional single-cell flow and mass cytometry data. *Cytometry Part A* **89**, 1084–1096 (2016).

55. Bodenheimer, T. *et al.* FastPG: fast clustering of millions of single cells. *BioRxiv* 2020–06 (2020).

56. Hussain, M. T. *et al.* Disruption-robust community detection using consensus clustering in complex networks. In *2022 IEEE International Symposium on Technologies for Homeland Security (HST)*, 1–6 (IEEE, 2022).

57. Yadav, N., Chatterjee, S. & Ganguly, A. R. Resilience of urban transport network-of-networks under intense flood hazards exacerbated by targeted attacks. *Scientific reports* **10**, 10350 (2020).

58. Dong, G. *et al.* Resilience of networks with community structure behaves as if under an external field. *Proceedings National Academy Sciences* **115**, 6911–6915 (2018).

59. Glantz, R. & Meyerhenke, H. Many-to-many correspondences between partitions: Introducing a cut-based approach. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, 1–9 (SIAM, 2018).

A Appendix

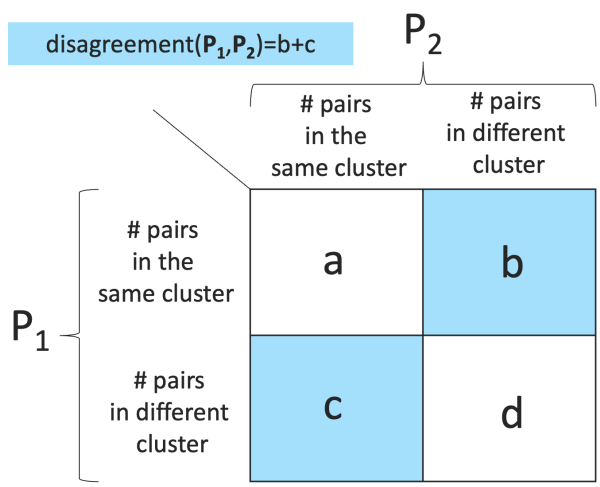


Figure 10. Contingency table of number of vertex pair count when comparing two clusterings. Pairs of vertices for a partion can be divided into two groups - pairs belonging to the same cluster and pairs belonging to different clusters. When comparing two clusterings the pairs can be divided into four groups - a, b, c and d.

Equation 1 can be optimized with any distance metric. In our algorithm, we aim to minimize the Mirkin distance that measures disagreements between two partitions. Here, we show the derivation of Equation 4 from Equation 1 using the Mirkin distance. The Mirkin distance is a measure used in clustering and partitioning contexts to quantify the dissimilarity between two partitions of a set. It is defined as the number of pairs of elements that are clustered together in one partition but not in the other. Hence, according to Fig. 10, the Mirkin distance $\mathbf{P}_1$ and $\mathbf{P}_2$ is $d(\mathbf{P}_1,\mathbf{P}_2) = b+c$. This distance is a metric and is equivalent to the symmetric difference between the sets of all co-clustered pairs in $\mathbf{P}_1$ and $\mathbf{P}_2$ (i.e. the symm. diff. of the equivalence relations induced by the partition). Generally, the Mirkin distance between $\mathbf{P}_i$ and $\mathbf{P}_j$ is defined as

$$d(\mathbf{P}_i,\mathbf{P}_j) = b+c = \sum_{uv}(\delta_{uv}(\mathbf{P}_i)+\delta_{uv}(\mathbf{P}_j)-2\delta_{uv}(\mathbf{P}_i)\delta_{uv}(\mathbf{P}_j)). \tag{7}$$

In this derivation, we used the following properties of a symmetric difference $A \triangle B$ between two sets A and B: $|A \triangle B| = |A| + |B| - 2|A \cap B|$.

We now derive Equation 4 from Equation 1.

$$
\begin{aligned}
\sum_{i=1}^{k} d(\mathbf{C}, \mathbf{P}_i) &= \sum_{i=1}^{k} \sum_{uv} (\delta_{uv}(\mathbf{C}) + \delta_{uv}(\mathbf{P}_i) - 2\delta_{uv}(\mathbf{C})\delta_{uv}(\mathbf{P}_i)) \\
&= \sum_{uv} (\sum_{i=1}^{k} \delta_{uv}(\mathbf{C}) + \sum_{i=1}^{k} \delta_{uv}(\mathbf{P}_i) - 2\sum_{i=1}^{k} \delta_{uv}(\mathbf{C})\delta_{uv}(\mathbf{P}_i)) \\
&= \sum_{uv} (k\delta_{uv}(\mathbf{C}) + \delta_{uv} - 2\delta_{uv}(\mathbf{C})\delta_{uv}) \\
&= \sum_{uv} \delta_{uv} + \sum_{uv} [k - 2\delta_{uv}]\delta_{uv}(\mathbf{C}) \\
&\approx \sum_{uv} [k - 2\delta_{uv}]\delta_{uv}(\mathbf{C})
\end{aligned}
$$

Note that $\sum_{uv} \delta_{uv}$ does not depend on $\mathbf{C}$, thus we do not consider it in our optimization problem.

Now, we consider the change in the objective function when the vertex v is moved from its current cluster $\mathbf{c}$ to another cluster $\mathbf{c}'$.

$$
\begin{aligned}
\Delta D_{v:\mathbf{c}\to\mathbf{c}'} &= \sum_{u \in \mathbf{c}} (k - 2\delta_{uv}) - \sum_{u \in \mathbf{c}'} (k - 2\delta_{uv}) \\
&= \sum_{u} (k - 2\delta_{uv})\sigma_u(\mathbf{c}) - (k - 2\delta_{uv})\sigma_u(\mathbf{c}') \\
&= \sum_{u} [\sigma_u(\mathbf{c}') - \sigma_u(\mathbf{c})](2\delta_{uv} - k)
\end{aligned}
$$