Bayan algorithm: Detecting communities in networks through exact and approximate optimization of modularity

Samin Aref [1,*] Mahdi Mostajabdaveh [2] and Hriday Chheda [1]
[1]*Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Canada M5S 3G8*
[2]*Department of Mathematical and Industrial Engineering, Polytechnique Montréal, Montreal, Canada H3T 1J4*

(Received 26 May 2024; accepted 24 September 2024; published 24 October 2024)

Community detection is a classic network problem with extensive applications in various fields. Its most common method is using modularity maximization heuristics which rarely return an optimal partition or anything similar. Partitions with globally optimal modularity are difficult to compute, and therefore have been underexplored. Using structurally diverse networks, we compare 30 community detection methods including our proposed algorithm that offers optimality and approximation guarantees: the Bayan algorithm. Unlike existing methods, Bayan globally maximizes modularity or approximates it within a factor. Our results show the distinctive accuracy and stability of maximum-modularity partitions in retrieving planted partitions at rates higher than most alternatives for a wide range of parameter settings in two standard benchmarks. Compared to the partitions from 29 other algorithms, maximum-modularity partitions have the best medians for description length, coverage, performance, average conductance, and well clusteredness. These advantages come at the cost of additional computations which Bayan makes possible for small networks (networks that have up to 3000 edges in their largest connected component). Bayan is several times faster than using open-source and commercial solvers for modularity maximization, making it capable of finding optimal partitions for instances that cannot be optimized by any other existing method. Our results point to a few well-performing algorithms, among which Bayan stands out as the most reliable method for small networks. A PYTHON implementation of the Bayan algorithm (*bayanpy*) is publicly available through the package installer for PYTHON.

DOI: 10.1103/PhysRevE.110.044315

I. INTRODUCTION

Community detection (CD), the data-driven process of partitioning nodes within a network [1], is a core problem in several fields including physics, mathematics, computer science, biology, and other computational sciences [2]. Among the common approaches for CD are the algorithms which are designed to maximize a utility function, *modularity* [3], across all possible ways that the nodes of the input network can be partitioned into an unspecified number of communities. Modularity measures the fraction of edges within communities minus the expected fraction under a random degree-preserving distribution of the edges. Despite their name and design philosophy, current modularity maximization algorithms, which are used by no less than tens of thousands of peer-reviewed studies [4], generally fail to maximize modularity or guarantee proximity to a globally maximum-modularity partition (an optimal partition) [5]. They have a high risk of failing to obtain relevant communities [6] and may result in degenerate partitions [7] that are provably far from the underlying community structure [8,9].

Modularity is among the first objective functions proposed for optimization-based CD [3,10]. Several limitations of modularity [11], including its resolution limit [12], have led researchers to develop alternative methods for detecting communities using information theory [13,14], stochastic block modeling (inferential methods) [15–19], and alternative objective functions [20–24]. In spite of its shortcomings, modularity is the most commonly used method for CD [2,25], while its mathematically rigorous maximization has remained relatively underexplored [5,26–28].

We propose the Bayan algorithm: an exact and approximation algorithm for the global maximization of modularity in networks with up to 3000 edges. Bayan resolves a fundamental limitation [5] of previous modularity-based community detection algorithms which rely on heuristic rules rather than rigorous mathematical optimization [e.g., integer programming (IP)]. We deploy standard techniques from integer programming (and operations research) for solving a network science (and physics) problem. In this article we present the fundamentals of our method using accessible language for network scientists and other practitioners who may deal with optimization for network clustering. We provide detailed references to the Supplemental Material document for additional technical optimization details [29].

II. EXACT AND APPROXIMATE VS HEURISTIC MODULARITY MAXIMIZATION

Maximizing modularity is an NP-hard problem [30], which explains the abundance of heuristic algorithms for modularity-based CD. These heuristic algorithms are widely used not only because of their scalability to large networks [31], but also because their high risk of failing to obtain relevant communities is not well understood [6]. The

*Contact author: aref@mie.utoronto.ca

scalability of these heuristics comes at a cost: their partitions have no guarantee of proximity to an optimal partition [7]. In an earlier study [5], we compared eight modularity-based methods based on the extent to which they maximize modularity. This included the heuristics known as greedy [32], Louvain [33], Leicht-Newman (LN) (aka Reichardt-Bornholdt) [34], Combo [25], Belief [35], Paris [36], Leiden [37], and EdMot [38]. The results showed that these heuristics rarely return an optimal partition [5]. Moreover, their suboptimal partitions tend to be disproportionately dissimilar to any optimal partition [5,8] according to results based on an adjusted mutual information notion of partition similarity [39].

There are hundreds of heuristic CD algorithms [2], including tens of CD algorithms in publicly available libraries [40–43]. However, to the best of our knowledge, there is no study comparing as many as 30 algorithms on standard benchmarks. This gap often leads empirical studies to rely on algorithms that are conveniently accessible or widely adopted [4], rather than selecting the most suitable algorithm for the specific CD task at hand.

Given the intricacies and nuances of different CD tasks, it is recommended [44] to transition from developing general CD algorithms (one-size-fits-all methods) to specialized CD algorithms that perform well on a narrow set of tasks. For the narrow set of small networks with up to 3000 edges (in their largest connected component), advances in IP may push the limits on large networks whose optimal partitions can be obtained using regular computers. While such small networks are sometimes dismissed as having no bearing on practical applications, there are disciplines and application areas where the large majority of networks are small. For example, in psychometric network analysis where community detection is a prevalent method, most networks have fewer than 100 nodes and 89% of networks have fewer than 30 nodes [45]. For many offline social networks where the nodes are sentient beings (people, animals, or other social collectives), the data collection methods impose some practical bounds on the size and order of the network. For example, unless archival records are used, it is often not practical to survey or interview more than a few hundred people, especially when the tolerance for missing data is very low. Therefore, we argue that small networks are not just toy examples for fields where networks tend to be large scale, but they are common examples in several other fields, in which network computations can be made more accurate.

The community detection literature offers much fewer exact and approximation methods[1] than heuristics. Each previous exact method has been restricted to a specific network size: Aloise *et al*. have used column generation to find optimal partitions in networks with up to 512 nodes and 819 edges [26] (within a 27-h time limit). Dinh and Thai have used linear programming rounding to approximate optimal partitions for networks with 78–2742 edges while reporting that the exact modularity maximization for such networks is cost prohibitive [27]. Sobolevsky *et al*. have reported guaranteed optimal partitions for networks with up to 50 nodes [28]. In 2023, Brusco *et al*. have reported results on optimal partitions for networks with up to 613 edges [46].

[1]We make a distinction between heuristic and approximation methods for optimization in that approximation methods have guarantees of proximity to optimal solutions.

III. OUR CONTRIBUTIONS

Given the recommended directions [5,44], we propose the Bayan algorithm, a specialized CD method capable of providing a globally optimal partition for small networks.

Bayan can alternatively be used to approximate maximum modularity within a user-specified factor at a reduced computation time. This algorithm is theoretically grounded by an IP formulation of the modularity maximization problem [27] and relies on an exact branch-and-cut scheme for solving the NP-hard optimization problem to global optimality.

We compare Bayan with 29 other CD algorithms. This also allows us to (1) find other accurate and suitable methods regardless of whether they use modularity or not and (2) assess the practical relevance of modularity as an objective function in comparison to a wide range of other proposed approaches for CD.

Our proposed algorithm pushes the largest instances whose maximum-modularity partitions can be obtained exactly on an ordinary computer. Bayan handles previously challenging instances of modularity maximization more efficiently. It also solves new instances that cannot be optimized by any other existing methods (see Tables S3– S5 in the Supplemental Material for the detailed results).

IV. MATHEMATICAL PRELIMINARIES

A. Notations

We represent the simple undirected and unweighted[2] graph G with node set V and edge set E as $G = (V, E)$. Graph G has $|V| = n$ nodes and $|E| = m$ edges. n and m are called the order and size of the graph, respectively. The symmetric adjacency matrix of graph G is represented by $\mathbf{A} = [a_{ij}]$ whose entry at row i and column j is a_{ij}. Entry a_{ij} indicates whether node i is connected to node j ($a_{ij} = 1$) or not ($a_{ij} = 0$). The degree of node i is represented by $d_i = \sum_j a_{ij}$. The modularity matrix of graph G is represented by $\mathbf{B} = [b_{ij}]$ whose entries are $b_{ij} = a_{ij} - \gamma d_i d_j / 2m$. γ is the resolution parameter[3] [47].

Node set V of the input graph G can be partitioned into an unspecified number of k nonoverlapping communities[4] based on the partition $P = \{V_1, V_2, \ldots, V_k\}$ such that $\bigcup_1^k V_i = V$ and $V_i \cap V_j = \emptyset$. Under partition P, the relative community assignment of a pair of arbitrary nodes (i, j), is either the same [represented by $1 - x_{ij} = \delta(i, j) = 1$] or different [represented by $1 - x_{ij} = \delta(i, j) = 0$]. The partition

[2]For the sake of simplicity, we focus on unweighted graphs in this article.

[3]Without loss of generality, we set $\gamma = 1$ for all the analyses in this article. The Bayan algorithm also supports other user-specified γ values for multiresolution modularity maximization.

[4]We focus on the more general CD problem where k is not specified by the user. The desired number of communities can be indirectly controlled by changing γ in Bayan.

P can therefore be represented as a symmetric partition matrix $\mathbf{X} = [x_{ij}]$ with binary entries x_{ij} each indicating the relative community assignment of nodes i and j.

B. The optimization problem statement

Given undirected and unweighted graph G and partition $\mathbf{X}$ as input, the modularity function $Q_{(G,\mathbf{X})}$ maps its input to a real value in the range of $[-0.5, 1]$ according to Eq. (1).

$$Q_{(G,\mathbf{X})} = \frac{1}{2m} \sum_{(i,j)\in V^2} \left(a_{ij} - \gamma \frac{d_i d_j}{2m} \right) \delta(i, j). \tag{1}$$

In the modularity maximization problem for graph G, we look for an *optimal* partition: a partition $\mathbf{X}^*_{(G)}$ whose modularity is maximum over all possible partitions: $\mathbf{X}^*_{(G)} = \arg\max_{\mathbf{X}} Q_{(G,\mathbf{X})}$. Any partition of G that is not an optimal partition is a *suboptimal* partition.

C. Sparse IP formulation of modularity maximization

The modularity maximization problem can be formulated as an IP model as in Eq. (2) which is proposed in [27].

$$\max_{x_{ij}} Q = \frac{1}{2m} \left(\sum_{(i,j)\in V^2, i<j} 2b_{ij}(1 - x_{ij}) + \sum_{(i,i)\in V^2} b_{ii} \right)$$

$$\text{such that} \quad x_{ik} + x_{jk} \geqslant x_{ij} \quad \forall (i, j) \in V^2, \quad i < j, \; k \in K(i, j)$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in V^2, \quad i < j \tag{2}$$

In the IP model in Eq. (2) for the input graph G, the optimal objective function value $Q^*_{(G)}$ equals the maximum modularity of graph G. An optimal partition $\mathbf{X}^*_{(G)}$ is represented by the optimal values of the x_{ij} variables.

In Eq. (2), $K(i, j)$ indicates a minimum-cardinality separating set [27] for the nodes i, j and its usage in the IP model of this problem leads to a more efficient formulation with $O(n^2)$ constraints [27] instead of $O(n^3)$ constraints [30,48]. For every three nodes that satisfy $(i, j) \in V^2$, $i < j$, $k \in K(i, j)$, there are three constraints in the IP model in Eq. (2). These so-called *triangular constraints* collectively ensure that being in the same community is a transitive relation [30].

Solving this optimization problem is NP-hard [30], but Bayan uses a branch-and-cut scheme for pushing the limit on the largest instances whose maximum-modularity partitions can be obtained exactly (or approximated within a factor) on an ordinary computer within a reasonable time. A narrative description on how the Bayan algorithm solves the IP model in Eq. (2) is provided in Sec. V while more technical details are provided in the Supplemental Material (SM) document. A PYTHON implementation of the Bayan algorithm (*bayanpy*) is publicly available through the package installer for PYTHON (pip).

V. THE BAYAN ALGORITHM

In this section, we provide a narrative description on the main methodological details of the Bayan algorithm.

A. Triangular constraints as a disjunction

Given that the decision variables in the IP model in Eq. (2) are all binary, the triangular constraints for the triple (i, j, k) can be written as the logical disjunction of Eqs. (3) and (4).

$$x_{ij} + x_{ik} + x_{jk} = 0, \tag{3}$$

$$x_{ij} + x_{ik} + x_{jk} \geqslant 2. \tag{4}$$

B. Branch-and-cut scheme

To maximize modularity using a branch-and-cut scheme, we start by obtaining lower and upper bounds for the IP model in Eq. (2). Solving the linear programming (LP) relaxation resulted from dropping the integrality constraint from the IP model in Eq. (2) provides an upper bound. Bayan uses Gurobi [49] to solve all the LP models involved within the branch-and-cut process. The modularity value of any feasible partition (e.g., obtained using the Combo algorithm, given the relative closeness of its partition to optimal partitions [50]) provides a lower bound. These two values form the bounds for the root node of an IP search tree.

C. Branching on node triples

We then select a triple (i, j, k) for which the LP optimal solution violates both Eq. (3) and Eq. (4) (see Sec. 1.F in the SM for more details on triple selection). On the selected triple, we partition the feasible space into two subspaces by adding either Eq. (3) or Eq. (4) as a cut to the root node problem which leads to a left and a right subproblem. For each subproblem, an upper bound can be obtained by solving the corresponding LP relaxation. Similarly, for each subproblem, a lower bound can be obtained by computing the modularity of a feasible partition obtained using a heuristic (e.g., the Combo algorithm) for the graph modified based on the added cut.

D. Manipulating the graph for left subproblems

For the upper bound and lower bound of the left subproblem, we use the analytical results on equivalences in maximum-modularity partitions [51,52]. Accordingly, we apply Eq. (3) by replacing nodes i, j, k with a new supernode and connecting it to all their neighbors.

E. Manipulating the modularity matrix for right subproblems

For the lower bound of the right subproblem, we apply Eq. (4) by deducting a positive empirically tuned value Δ from the six modularity matrix entries associated with the triple (i, j, k). This change makes the nodes i, j, k less appealing to be assigned to the same community in the heuristic feasible solution. It does not guarantee that the heuristic solution satisfies Eq. (4), but there is no such requirement for Bayan's convergence to optimality (see Sec. 1.C in the SM for more details on how the Δ parameter is tuned).

F. Upper bound and lower bound for maximum modularity

The search tree can further be explored by branching based on another triple whose triangular constraints are violated by the new LP optimal solutions. Throughout the search, the

largest lower bound is continuously updated and stored as the *incumbent*, while the largest upper bound (of each level) is stored as the *best bound* only after completing the computations for each level of the tree.

G. Closing nodes in the search tree

Throughout the branching process, three conditions lead to designating a tree node as *fathomed*. First, the LP solution becoming integer. Second, the LP becoming infeasible. Third, the LP solution becoming smaller than the current incumbent. In all these cases, we do not branch on the node and *close* it.

H. Termination of exact and approximate Bayan

The convergence of incumbent and best bound guarantees that a globally optimal solution has been found. This branch-and-cut scheme gives rise to the *Bayan exact algorithm* if this convergence is used as the termination criterion. Alternatively, users can set a different termination criterion (specifying a desired run time or an optimality gap tolerance) which leads to the *Bayan approximate algorithm* for approximating maximum modularity within an optimality gap (the relative gap between the incumbent and best bound at the termination of the algorithm).

Section 1 of the SM document provides six detailed subsections on specific technical details about the more nuanced design aspects of the Bayan algorithm. Having provided an accessible explanation of the Bayan algorithm, we move on to describing how standard benchmarks and partition quality measures are used in our study for comparing 30 algorithms including Bayan on an equal footing.

VI. TECHNICAL BACKGROUND FOR THE COMPARISONS

A. Partition similarity measure

We quantify the similarity of a partition obtained by an algorithm to a desirable partition (e.g., a planted ground-truth partition) using adjusted mutual information [39] and normalize it symmetrically [53]. The normalized adjusted mutual information (AMI) is a measure of similarity of partitions which (unlike normalized mutual information [53,54]) adjusts the measurement based on the similarity that two partitions may have by pure chance. The AMI for a pair of identical partitions (or permutations of the same partition) equals 1. For two partitions which have no similarity beyond the similarity caused by random chance, AMI takes a small (positive or negative) value close to 0 [39].

We use AMI because it is shown to be a reliable measure of partition similarity [39,55], and avoid using normalized mutual information (NMI) because, despite its common use [56,57], several studies indicate that using it leads to incorrect assessments [39,55,58,59] and incorrect evaluation of competing algorithms [53]. Similar to the NMI, other popularly used measures of partition similarity (the Jaccard index, the Fowlkes-Mallows index, the adjusted Rand index, and the F measure) suffer from at least one form of undesirable bias [55] and therefore, we avoid using them.

B. Generating structurally diverse benchmark networks

To evaluate the performance of different community detection algorithms, we use Lancichinetti-Fortunato-Radicchi (LFR) benchmark graphs [60] as well as artificial benchmarks for community detection (ABCD) graphs [61]. We generate 1000 synthetic graphs with randomized parameters described below to ensure that the differences observed in the performance of the algorithms are not restricted for specific graph structures.

We generate LFR benchmarks based on the following randomized parameters: number of nodes (n) randomly chosen from the range $[20, 300]$, maximum degree $\lfloor 0.3n \rfloor$, maximum community size $\lfloor 0.5n \rfloor$, power-law exponent for the degree distribution $\tau_1 = 3$, power-law exponent for the community size distribution $\tau_2 = 1.5$, and average degree of 4. The parameter μ (mixing parameter) is chosen from the set $\{0.01, 0.1, 0.3, 0.5, 0.7\}$ for five experiment settings (each with 100 LFR graphs).

ABCD benchmarks [61] are the more recent alternative to the LFR model for generating benchmarks [60]. Being directly comparable to LFR, ABCD offers additional benefits including higher scalability and better control for adjusting an analogous mixing parameter [61]. We generate ABCD benchmarks based on the following randomized parameters: number of nodes (n) randomly chosen from the range $[10, 1000)$; minimum degree $d_{\min}$ and minimum community size $k_{\min}$ randomly chosen from the range $[1, n/4)$; maximum community size chosen randomly from $[k_{\min} + 1, n)$; maximum degree chosen randomly from $[d_{\min} + 1, n)$; and power-law exponents for the degree distribution and community size distribution randomly from $(1, 8)$ and then rounded off to two decimal places. The parameter ξ (mixing parameter) is chosen from the set $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ for five experiment settings (each with 100 ABCD graphs). The mixing parameters μ and ξ can be considered as indicators of the noise in the LFR and ABCD data generation process, respectively.

C. Partition quality measures

Retrieval tests based on LFR and ABCD benchmarks provide a level playing field for the community detection algorithms to be compared against each other in a method-agnostic way [60,61]. We also provide six partition quality measures for additional comparisons of the 30 CD algorithms without relying on the retrieval of planted partitions.

Comparing CD algorithms based on a single quality function that is used in some of them (e.g., description length or modularity) is suggested to be unfair and not rigorous [61]. Such an approach favors algorithms that were designed based on that specific measure and disfavors other CD algorithms. Given the disagreements on the suitability of partition quality measures [62], we refrain from relying on a single measure and instead report all the following measures: description length [17], modularity, average conductance [63], coverage [64], performance [64], and well clusteredness [62].

All six of these partition quality measures are functions that take input graph G and input partition P and return a value as the output. For description length and average conductance, a lower value indicates a better partition. For the other four measures, a higher value indicates a better partition. Except

for description length and modularity, the other four partition quality measures return values in the unit interval. Well clusteredness returns a binary value. We briefly describe these six partition quality measures.

Description length. The description length of a dataset refers to the amount of information required to describe the data. According to the minimum description length principle, a model that can compress the data more effectively has captured more of its regularities and is therefore considered to have shown a better performance [65].

To evaluate a partition P of a graph $G = (V, E)$ using a stochastic block model (SBM) [16] with parameters θ, the description length is calculated as

$$-\ln p(G|\theta, P) - \ln p(\theta, P),$$

where $-\ln p(G|\theta, P)$ is the negative log-likelihood of the graph G given the partition P and the SBM parameters θ, and $-\ln p(\theta, P)$ is the prior probability of the partition P and the SBM parameters θ. In our evaluations, we calculate the description length using the planted partition model introduced in [17] as the underlying SBM.

Modularity. The modularity value $Q_{(G,\mathbf{X})}$ for graph G and partition P (that corresponds to the partition matrix $\mathbf{X}$) is calculated based on Eq. (1).

Average conductance. The average conductance of partition P and graph G is the mean of conductance values over communities of partition P. A value for conductance can be calculated for each community, i, of partition $P = \{V_1, V_2, \ldots, V_k\}$ and graph G based on

$$\frac{|E_i|}{\min\left(d(V_i), d(V \setminus V_i)\right)},$$

where $|E_i|$ denotes the number of edges with one end point in community V_i, $d(V_i)$ is the sum of degrees of nodes assigned to community V_i, and $d(V \setminus V_i)$ is the sum of degrees of nodes not assigned to community V_i.

Coverage. The coverage of partition P is the ratio of the number of its intracommunity edges to the total number of edges in graph G.

Performance. The performance of partition P is the total count of intracommunity edges plus intercommunity nonedges divided by $n(n-1)/2$. This denominator is the size of the complete graph that has n nodes just like G.

Well clusteredness. The well clusteredness is a binary measure based on the three measures $\bar{K}_{\text{inter}}$, K, and $\bar{K}_{\text{intra}}$ proposed in [62] and defined as follows:

$$\bar{K}_{\text{inter}}(G, P) = \frac{2}{k(k-1)} \sum_{i=1}^{k} \sum_{j=i+1}^{k} \frac{|E_{ij}|}{|V_i||V_j|},$$

where k is the number of communities in partition P, V_i denotes the ith community in partition P, and $|E_{ij}|$ denotes the number of edges with one end point in community i and one end point in community j;

$$K(G) = 2m/n(n-1),$$

where $K(G)$ is the density of graph G; and

$$\bar{K}_{\text{intra}}(G, P) = \frac{\sum_{i=1}^{k} K(V_i)}{k},$$

where $K(V_i)$ denotes the density of community V_i.

The necessary condition for graph G to be well clustered by partition P is for the two inequalities $\bar{K}_{\text{inter}} < K < \bar{K}_{\text{intra}}$ to hold [62]. For any graph G and partition P, the binary measure, well clusteredness, takes 1 if graph G is well clustered by partition P, and it takes 0 otherwise.

D. Comparing CD algorithms

There are different views on how CD algorithms can be assessed. The two major, and often opposing views, rely on theoretical argumentation vs practical argumentation.

Some experts use a theoretical argumentation and suggest that when we generate networks using some generative process, like the LFR or ABCD benchmark, then provably the Bayes-optimal algorithm for inferring the planted communities is the one that simply inverts the generative process for inference. Therefore, without performing any comparisons, we know that maximum *a posteriori* estimation with a degree-corrected SBM with appropriate priors and mixing structure will perform the best for the LFR benchmark if an infinite number of trials are conducted and we use a suitable error measure for the performance (an ideal partition similarity measure). They argue that similar guarantees hold for other estimators and loss functions. This argument concludes that any distinction among algorithm performance is thus a finite sample size issue or because the error measure used is different (e.g., AMI or NMI).

Other experts use a practical argumentation and suggest that standard LFR and ABCD benchmarks are informative for comparing the capabilities of CD algorithms [60,61]. LFR and ABCD are standard and widely adopted benchmarks developed to serve this specific purpose of comparing the practical capabilities of CD algorithms in a controlled environment. In practical situations, appropriate priors and mixing structure cannot be simply assumed and an infinite number of trials cannot be performed. Therefore, the LFR and ABCD benchmarks provide valuable insights on the performance of CD algorithms that may violate theoretical expectations. Moreover, if we generate benchmarks and use the information on how they are generated (LFR/ABCD) to reach analytical results for supporting a specific algorithm, we would defeat the purpose of the benchmark models.

Our problem definition in Sec. I is aligned with the practical perspective rather than the theoretical perspective. Therefore, after acknowledging the merits and limitations of the theoretical perspective that suggests comparing CD algorithms is fruitless, we continue to compare 30 CD algorithms using standard LFR and ABCD benchmarks as well as six partition quality measures to assess their practical capabilities on structurally diverse benchmarks across different mixing parameter values. We briefly describe the motivations behind the development of several benchmark models for community detection to prepare the context for the interpretation of our results in Sec. VII.

The LFR benchmarks are developed with a timely observation on the part of its developers arguing that "many algorithms have been proposed [...] the question of how good an algorithm is, with respect to others, is still open" ([60], pp 046110-1). The LFR benchmark model attempts to address a methodological problem of crucial importance

[66,67]: comparing the accuracy of CD methods. The LFR benchmark improved upon earlier benchmarks [68] which were preferential attachment networks of 128 nodes with fixed degrees and fixed community sizes.

The development of the ABCD benchmarks was motivated by the necessity to compare CD algorithms on synthetic graphs that resemble some key features of typical real-world networks like community structure, heterogeneous degrees, and heterogeneous community sizes [61]. Algorithms should be compared based on networks with a varying strength of community structure which is operationalized through the ξ mixing parameter. Emphasizing on the usefulness of the LFR benchmark model, ABCD builds upon the same foundation, but resolves three limitations of the LFR model [61].

E. CD algorithms included in our evaluations

Using LFR and ABCD as two different families of benchmarks, six partition quality measures, and four real networks from different contexts, we compare the following 30 CD algorithms: Bayan, eight modularity-based heuristics, other optimization-based algorithms, and CD methods which do not rely on modularity or optimization. The 29 algorithms compared to Bayan (in the chronological order of 1970–2023) are the CD methods known as Kernighan-Lin bisection [69], greedy [32], Chinese whispers [70], Reichardt Bornholdt with Erdős-Rényi as the null model (RB) [71], Walktrap [72], kcut [73], asynchronous label propagation [74], Louvain [33], Infomap [14], Reichardt Bornholdt with the configuration model as the null model (LN) [34,71], Genetic Algorithm (GA) [75], semisynchronous label propagation [76], CPM [77], significant scales [78], Weighted Community clustering (WCC) [79], Combo [25], Belief [35], Stochastic Block Model (SBM) (minimum description length) [16], SBM with Markov Chain Monte Carlo (MCMC) [16], Surprise [20], Diffusion Entropy Reducer (DER) [80], Paris [36], Leiden [37], EdMot [38], GemSec [81], Bayesian Planted Partition (BPP) [17], BPP with MCMC [17], Markov stability (PyGenStability[5]) [43] with random partition selection (MR), and Markov stability [43] with minimum normalized variance of information (MV).

We use the public implementations of these 29 algorithms from the following PYTHON libraries. For the two Markov stability algorithms (MR and MV), we use the *PyGenStability* library (version 0.2.2) [43]. For the four inferential methods (SBM, SM, BPP, and BM), we use the *graph_tool* library (version 2.57) [41]. For the two algorithms, Kernighan-Lin and asynchronous label propagation, we use the NetworkX library (version 3.1) [40]. For the remaining 21 algorithms, we use the Community Discovery library (CDlib; version 0.2.6) [42].

[5]The Markov stability algorithm in the PyGenStability library produces multiple partitions at different scales for one input network [43]. We use two simple adaptations of it (MR and MV) to obtain one partition for each input network. These two adaptations involve running PyGenStability with its automated optimal scale selection [43] which returns multiple candidate partitions corresponding to different scales. In MV, we select the partition with the minimum normalized variance of information. In MR, we select a partition randomly.

Three comparisons are provided on these 30 algorithms in Secs. VII–IX.

VII. PARTITION RETRIEVAL COMPARISONS

This section provides the first of the three comparisons which assesses the 30 algorithms on the extent to which they retrieve planted partitions of LFR and ABCD networks.

A. Comparison of 30 algorithms based on retrieving planted communities in LFR graphs

In our first retrieval comparison, we use the LFR benchmarks [60] introduced and parametrized earlier, and measure the performance of each method using the AMI of the partition with the planted partition from the LFR graph generation process.

In this evaluation, we use 500 LFR benchmark graphs in five experiment settings; each having 100 LFR graphs generated based on a specific μ value ($\mu \in \{0.01, 0.1, 0.3, 0.5, 0.7\}$). The mixing parameter μ in the LFR model determines the fraction of intercommunity edges. Larger values of μ complicate the accurate discovery of communities by decreasing the association between the structure and the planted communities.

The results on the average AMI of each algorithm in each experiment setting are provided in the Appendix (Table I). Figure 1 illustrates the ranking of the 30 algorithms, including Bayan, according to the AMI averaged over 100 LFR graphs in each of the five experiment settings (five values of μ). Among the 30 algorithms, Bayan is always among the three algorithms with the highest average AMIs. Bayan also has the most stable performance across the five values of μ. Walktrap and RB are two other algorithms demonstrating high performance; each of them achieves an average AMI higher than Bayan's on two out of five experiment settings. However, their performance ranking across the five values of μ is less stable than Bayan's.

In Figure 1, the two algorithms Paris and Combo seem to perform comparatively well for small values of μ, but their performance rankings decrease substantially when the μ mixing parameter passes the threshold value of 0.1. Contrary to this pattern, there are algorithms like EdMot whose comparative performance improves when μ increases. Infomap and three modularity-based heuristics, Combo, LN, and Leiden, show a comparatively decent performance, having higher AMIs than most other methods. Our observations on the high performance of modularity-based methods on LFR graphs are aligned with the results on LFR graphs reported in [53], where modularity-based methods return AMIs higher than asynchronous label propagation, walktrap, Bayesian planted partition, and sometimes higher than Infomap.

The descriptive AMI ranking results of the 30 algorithms in each experiment setting are validated using a Friedman test of multiple groups [82] followed by a *post hoc* Li test of multiple hypotheses [83]. More information about these statistical procedures is provided in Sec. 2 of the SM. The AMI results on the 100 LFR graphs with $\mu = 0.01$ with Bayan as the control method lead to the rejection of the null hypotheses in 19 out of 29 *post hoc* Li tests. This suggests

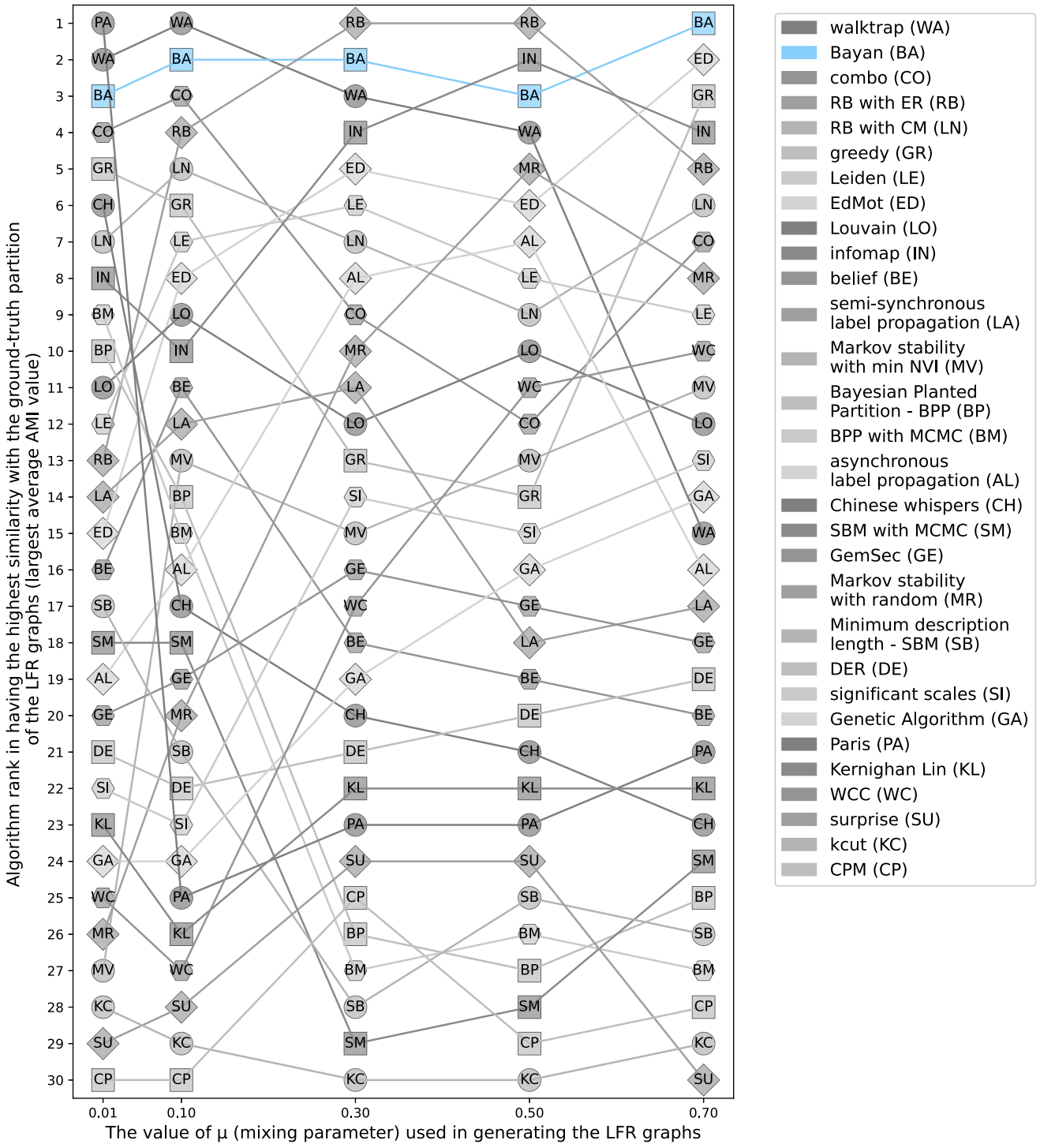

FIG. 1. Performance ranking of 30 CD algorithms based on the AMI averaged over 100 LFR graphs for each data point. (Magnify the high-resolution figure on screen for details.)

that for LFR graphs with $\mu = 0.01$, Bayan returns partitions with statistically significant higher similarity to the planted partition compared to each of those 19 other algorithms. For $\mu = 0.1$, the null hypotheses are rejected in 28 out of 29 *post hoc* Li tests. The only exception is the Walktrap algorithm, for which there is insufficient evidence of difference from Bayan in AMI. The limited rejections are partly due to the limitations in the statistical power of conducting a large number of tests among 30 algorithms while controlling the familywise error rate to 0.05 [83]. Overall, the statistical results (in Table S1 in the SM) show that Bayan has a significantly higher AMI means in comparison to most of the 29 other algorithms considered confirming the descriptive results in Fig. 1 (and Table I in the Appendix).

B. Comparison of 30 algorithms based on retrieving planted communities in ABCD graphs

We conduct additional retrieval tests to ensure that the results are not artifacts of using LFR benchmarks. In our second retrieval comparison, we use the ABCD benchmarks [61] introduced and parametrized earlier, and compare Bayan to the same set of 29 other CD algorithms and rank them based on their average AMI with the planted partitions on ABCD benchmark graphs.

To evaluate the 30 algorithms based on the retrieval of the planted communities, we use 500 ABCD benchmark graphs in five experiment settings, each having 100 ABCD graphs generated based on a specific ξ value ($\xi \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$). The mixing parameter $\xi \in [0, 1]$ determines the independence of the edge distribution from the communities [61]. At the extreme value of $\xi = 0$, all edges are within communities. In contrast, at the other extreme value, $\xi = 1$ indicates that communities do not influence the distribution of edges [61]. Therefore, larger values of ξ complicate the discovery of communities by decreasing the association between the structure and the planted communities.

The results on the average AMI of each algorithm in each experiment setting are provided in the Appendix (Table II). Figure 2 illustrates the comparative ranking of the algorithms, including Bayan, according to the AMI values averaged over 100 ABCD graphs in each of the five experiment settings (five values of ξ). Bayan has the highest or the second highest average AMI in four out of five experiment settings. Among the 30 algorithms for $\xi \in \{0.1, 0.3, 0.5, 0.7\}$, Bayan and Combo are consistently in the top three algorithms with the highest average AMIs. Setting aside the high-noise experiment setting of $\xi = 0.9$ where AMIs substantially drop across the board, Bayan and Combo have the most stable performance in accurate retrieval of the planted partitions of these ABCD graphs.

In Fig. 2, the three algorithms walktrap, LN, and Leiden seem to perform comparatively well especially for small values of ξ. Contrary to this pattern, there are algorithms like EdMot and surprise whose comparative performance monotonically improves when ξ increases. Averaging the AMI values for all 500 ABCD graphs, Bayan has the highest total average AMI followed by Combo and Leiden, respectively. Note that these three high-performing algorithms are all based on modularity while Bayan differs from the others in having optimality guarantees.

Similar to the rankings on LFR graphs, the descriptive ranking results on ABCD graphs are validated using a Friedman test of multiple groups [82] followed by a *post hoc* Li test of multiple hypotheses [83]. The AMI results on the 100 ABCD graphs with $\xi = 0.1$ with Bayan as the control method lead to the rejection of the null hypotheses in 19 out of 29 *post hoc* Li tests. This suggests that for ABCD graphs with $\xi = 0.1$, Bayan statistically significantly differs from each of the 19 other algorithms in retrieving the ground-truth partition. The exceptions are ten algorithms for which there is insufficient evidence of statistically significant difference from Bayan in AMI. Detailed statistical test results on ABCD graphs are provided in the SM (Table S2) confirming the results in Fig. 2 (and Table II in the Appendix).

Taken together with the results on LFR graphs, maximum-modularity partitions are shown to have a distinctive accuracy in retrieving planted partitions across a wide range of parameter settings for both standard CD benchmark models. Our results in Figs. 1 and 2 (and Tables I and II) show the major differences between the extent to which these 30 algorithms retrieve planted partitions in practice where the sample size is not infinite.

Designed as a specialized algorithm for solving an NP-hard problem in small networks (with up to 3000 edges), Bayan is not suitable for and does not scale to large-scale networks. Figures 1 and 2 also provide insight on algorithms other than Bayan that have reasonably good performance on both LFR and ABCD assessments. At the low extreme of performance ranks, there are several algorithms whose capability of retrieving ground-truth communities is not better than that of the Kernighan-Lin bisection algorithm from 1970. Note that for comparing all 30 algorithms, including several algorithms that do not (and cannot possibly) scale to large instances, we could not include large-scale benchmark networks in our assessments. Given this limitation, our results are safer to be interpreted within the context of the comparative performance of 30 algorithms on small networks with up to 3000 edges. Moreover, from a theoretical standpoint, the finite sample size prevents us from making a firm determination about how these algorithms compare asymptotically.

C. Comparison of 30 algorithms based on similarity to node attributes of fairly modular real networks

We also assess the similarity between the partitions of the same 30 algorithms and a set of node metadata (discrete-valued node attributes) on five real benchmark networks which are commonly used [28,38,84–89] in the literature for demonstrating the output of CD algorithms. Retrieving node label communities is not the purpose of CD algorithms, and network formation may poorly correlate with any arbitrarily chosen node attribute [44]. Therefore, we do not consider a given set of node metadata as ground truth. However, it is useful to compare the partitions returned by different algorithms based on their similarity with node metadata [85,86] to evaluate them in a setting without synthetic benchmarks on networks where node labels are aligned with the structure. Consistent with the literature [38,84,88,90], we use the five real networks commonly referred to as *dolphins*, *football*, *karate*, *polbooks*, and *risk_game* which all have node attributes. Information on accessing all networks can be found in Sec. 5 of the SM.

Figure 3 shows the AMI of the partitions from each of the 30 algorithms and the node attributes of the networks. As the node attribute considered is at most one factor among a multitude of factors determining the formation of these networks, the AMI values confound several effects and cannot be reliably interpreted as a performance measure of the algorithms [44]. They simultaneously measure the metadata's relevance to the structure and the performance of the algorithms in retrieving communities corresponding to the structure. Bearing these limitations in mind, the results indicate that some algorithms including Bayan return partitions with reasonably high similarity to the node attributes for all five networks,

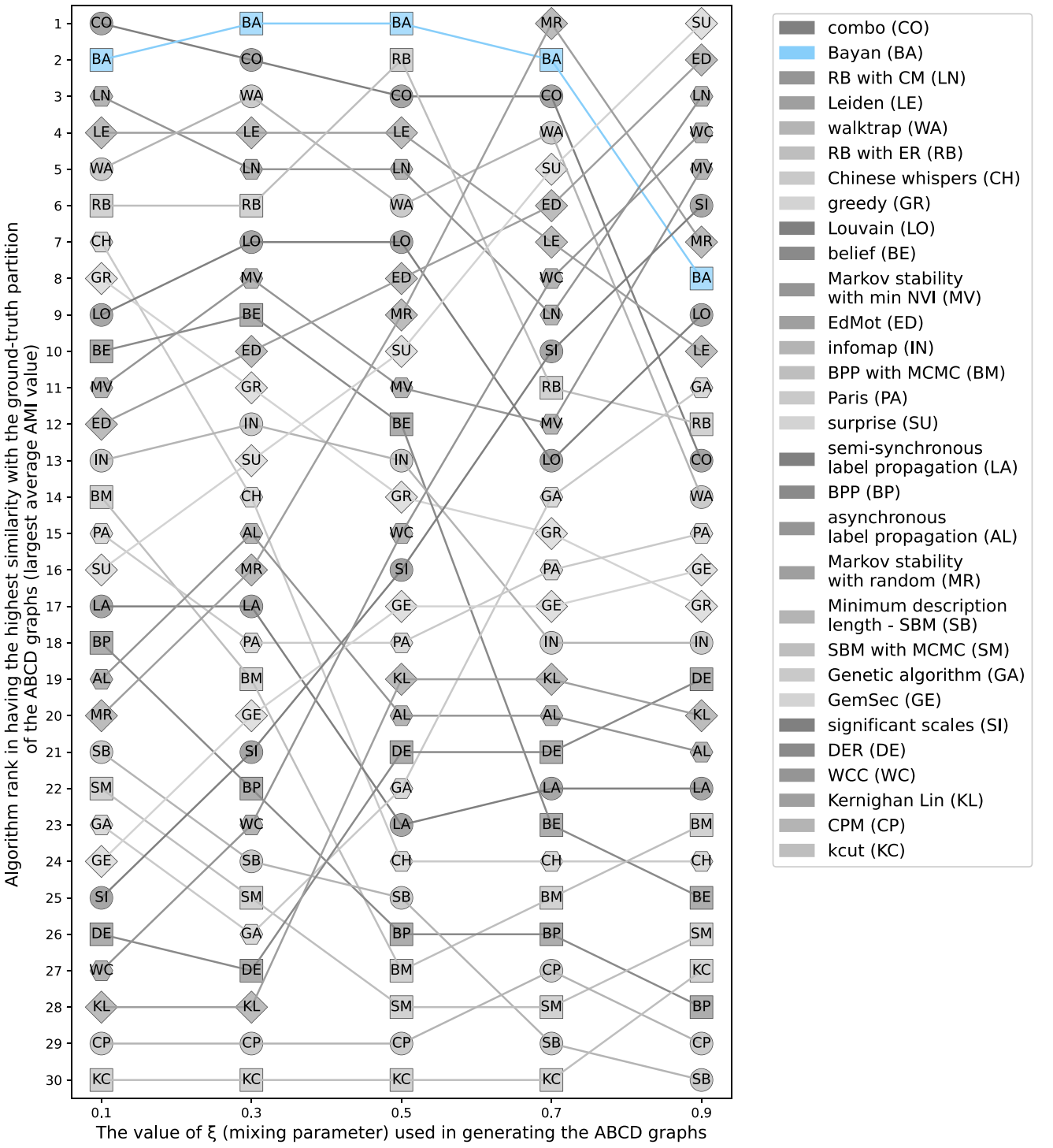

FIG. 2. Performance ranking of 30 CD algorithms based on the AMI averaged over 100 ABCD graphs for each data point. (Magnify the high-resolution figure on screen for details.)

whereas, other algorithms like CPM, kcut, and SBM return partitions with a less straightforward correspondence to the node metadata. Some algorithms like Walktrap, which comparatively had good performance on LFR and ABCD graphs, return some partitions with low similarity to node attributes in this assessment. A no-free-lunch theorem for CD [44] implies that no single method can consistently outperform others in accurately retrieving the possible node attributes of real networks. Maximum-modularity partitions are (happen to be) reasonably similar to the node attributes for these networks.

So far, we have discussed the advantage of maximum-modularity partitions in retrieving planted communities in LFR and ABCD graphs, and discovering partitions with strong associations with node attributes in some commonly

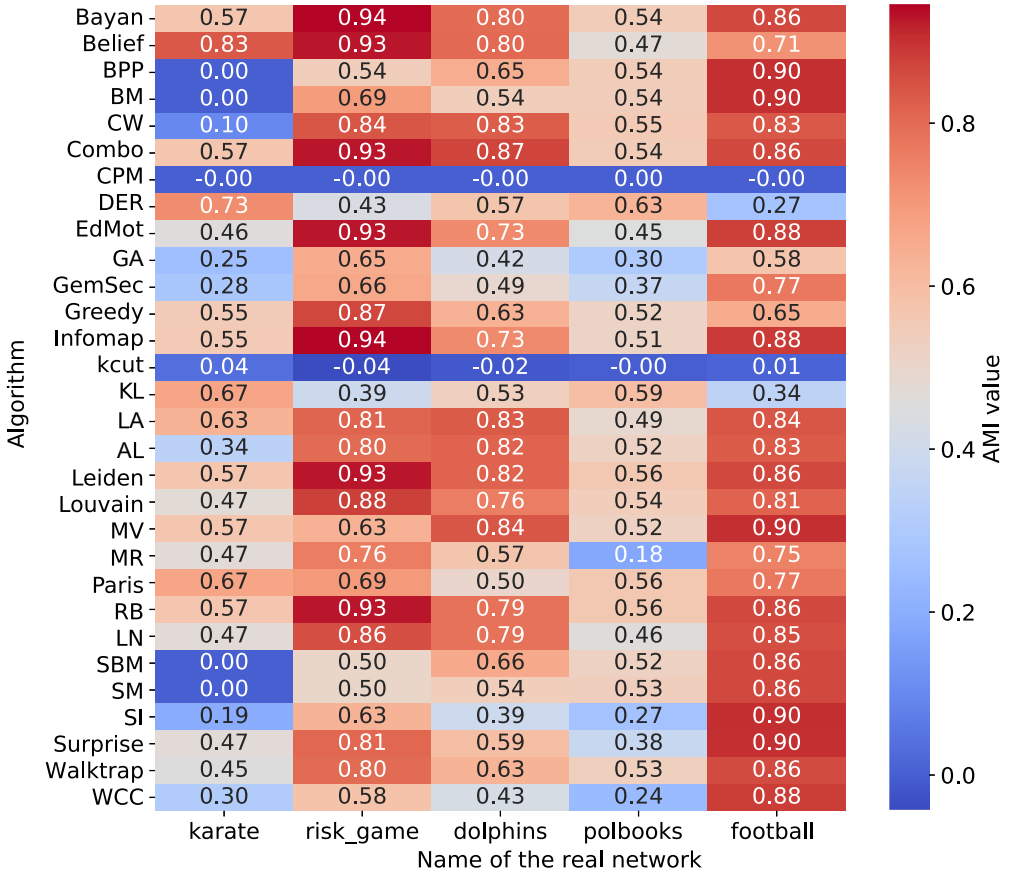

FIG. 3. AMI values of the CD algorithms indicating the similarity of their partitions with node attributes for five real networks. (Color version online. Magnify the high-resolution figure on screen for details.)

used real networks. Next, we report six comparisons of the 30 algorithms based on partition quality measures which do not depend on partition similarity measures.

VIII. PARTITION QUALITY COMPARISONS

This section provides the second comparison out of the three comparisons of the 30 CD algorithms. Here, we assess the algorithms based on six partition quality measures on the same LFR and ABCD networks.

We compute and report the six partition quality measures (defined in Sec. VI C) for the partitions produced by the 30 algorithms on the 500 LFR networks and the 500 ABCD networks in Figs. 4–9. The algorithms with the most desirable median value are shown in the leftmost positions in Figs. 4–9.

Figure 4 illustrates a box plot for the 500 description length values of each algorithm separately for LFR networks and ABCD networks. Bayan, Louvain, and Leiden have the best (lowest) median values of description length among the 30 algorithms, respectively, for both the LFR and the ABCD instances. Unlike the inferential methods, these three modularity-based algorithms are not designed to minimize the description length. However, they produce partitions with a lower median description length compared to the four inferential algorithms SBM, SM, BPP, and BM. This paradoxical result highlights the importance of a practical comparison of algorithms. Some theoretical motivations seem to be unreliable for choosing algorithms because they may not materialize in practice as illustrated for the inferential algorithms SBM, SM, BPP, and BM in Fig. 4.

Figure 5 illustrates a box plot for the 500 modularity values of each algorithm separately for LFR networks and ABCD networks. As expected, Bayan has the best median value of modularity among the 30 algorithms for both the LFR and the ABCD instances. We do not consider this comparison as a fair assessment and only provide it for completeness. Using any single partition quality measure for choosing a CD algorithm is contestable. Moreover, modularity is not a suitable partition quality measure[6] [62].

Figure 6 illustrates a box plot for the 500 average conductance values of each algorithm separately for LFR networks and ABCD networks. Similarly, the distributions of coverage values are provided in Fig. 7 and the distributions of performance values are shown in Fig. 8.

Bayan, Infomap, and SBM have the best median values of average conductance, respectively, on the LFR networks in Fig. 6. The same three algorithms also have the best median values for coverage and performance on LFR networks as shown in Figs. 7 and 8. On ABCD instances, Bayan, Louvain, and Leiden have the best median values of average conductance, respectively. The same three algorithms also have the

[6]There is a distinction between a quality measure (a score function) and an objective function (loss function).

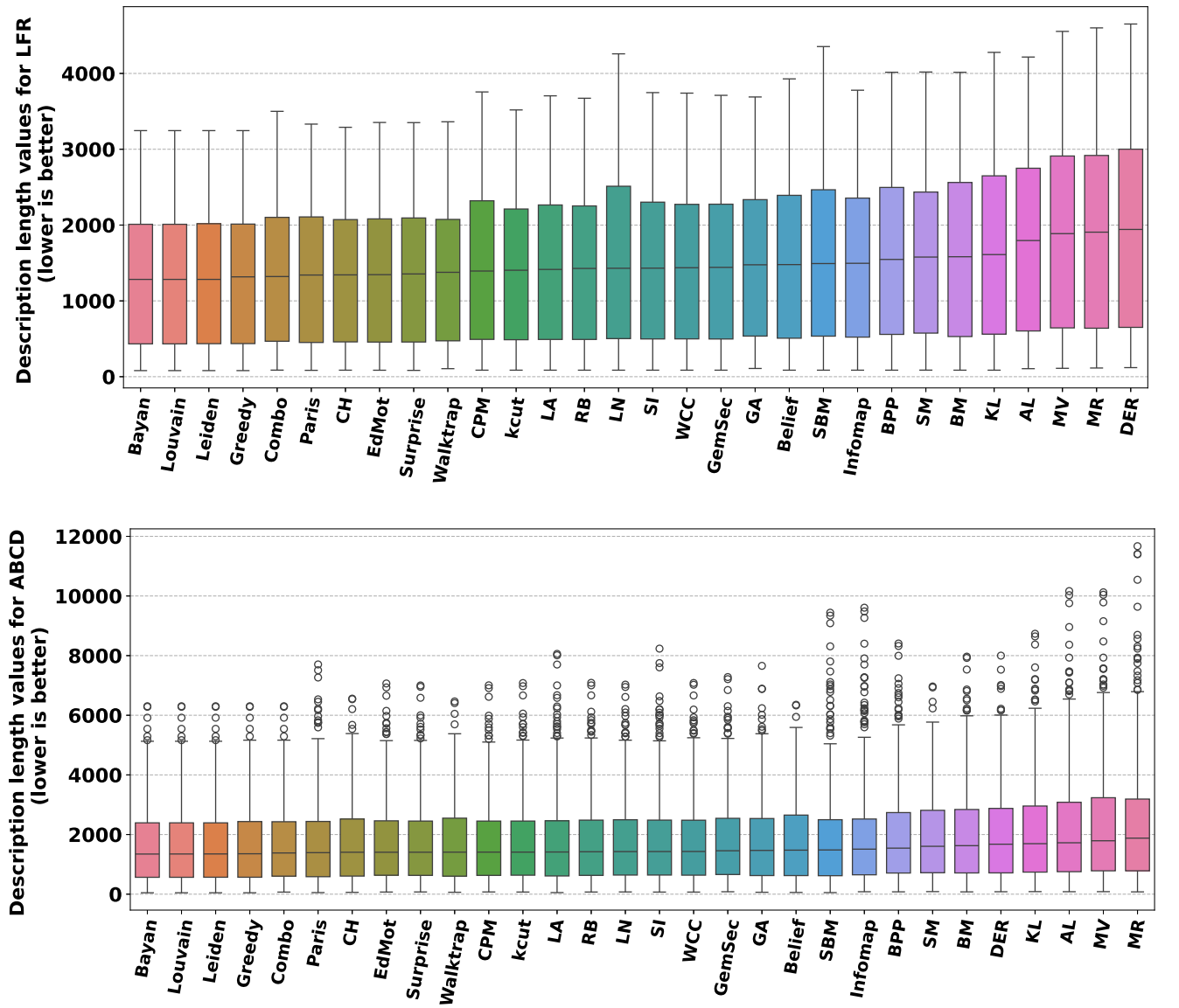

FIG. 4. The distribution of description length values for the partitions produced by each algorithm on the 500 LFR networks (top panel) and the 500 ABCD networks (bottom panel). The algorithms are sorted from right to left based on having a more desirable median description length.

best median values for coverage and performance on ABCD networks as shown in Figs. 7 and 8.

The results for well clusteredness are reported slightly differently in Fig. 9. Given that well clusteredness is a binary value, we report for each algorithm the percentage of LFR graphs that are well clustered and separately report the similar percentage for ABCD graphs in Fig. 9. The majority of the 30 algorithms including Bayan produce well-clustered partitions on all LFR networks. For ABCD networks, there are 14 algorithms including Bayan that produce well-clustered partitions on all instances. For both LFR and ABCD networks, the four inferential algorithms, SBM, SM, BPP, and BM, produce partitions that are not well clustered, more often than not. The two algorithms CPM and kcut never produce well-clustered partitions on either LFR or ABCD networks.

In this section, we compared the 30 algorithms based on partition quality measures which do not depend on the planted (ground-truth) partitions. Next, we focus on how the algorithms compare from a run time perspective.

IX. RUN TIME COMPARISONS

This section provides the last of the three comparisons for the 30 CD algorithms which assesses them based on their run times. We also include two alternative exact modularity maximization methods in Sec. IX B. These run time analyses are conducted in PYTHON 3.9 using a notebook computer with an Intel Core i7-11800H @ 2.30 GHz CPU and 64 GBs of RAM running Windows 10.

A. Comparison of 30 algorithms based on empirical run time

In this section, we compare 30 algorithms, including Bayan, based on their empirical run time on the same 500

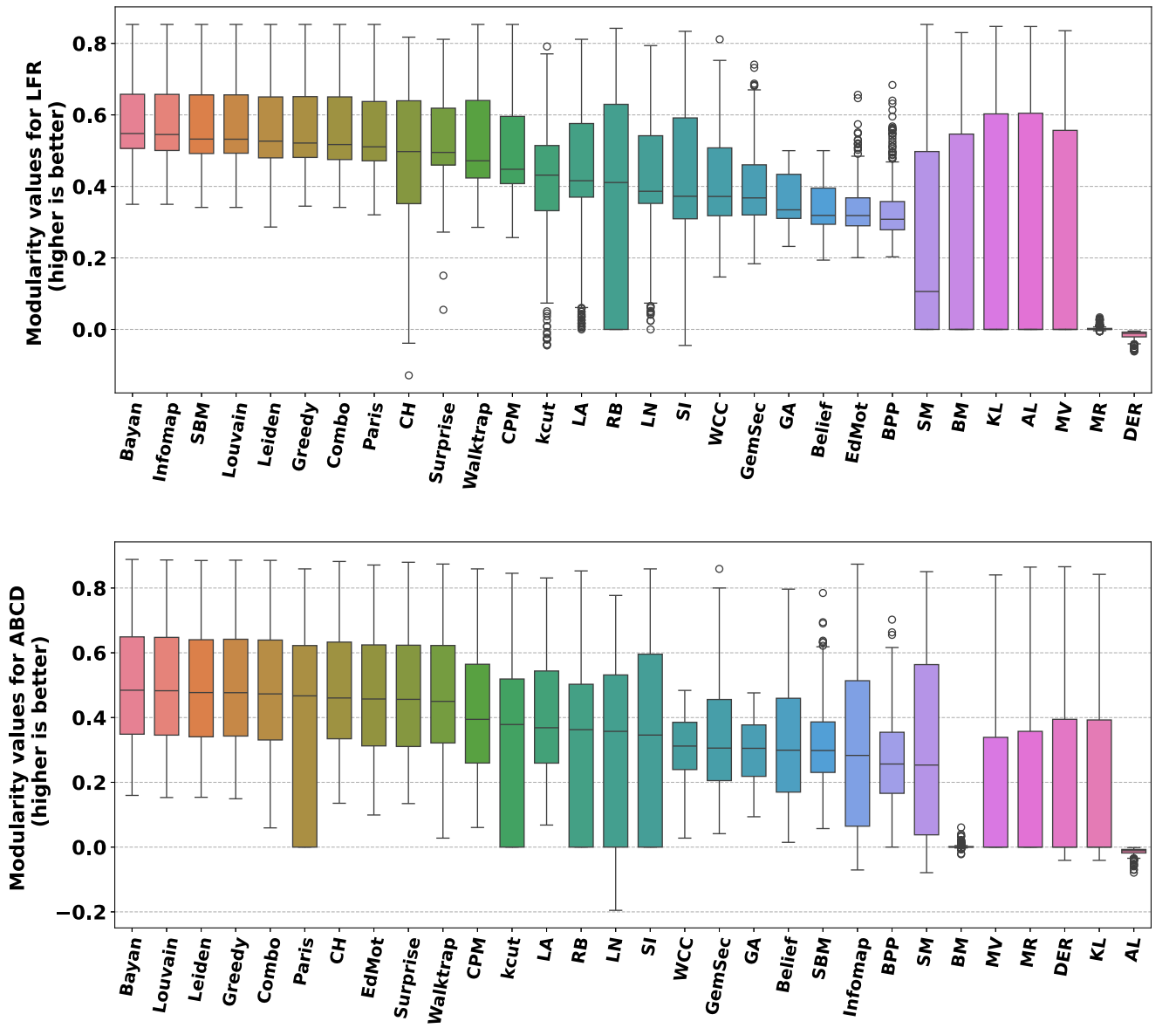

FIG. 5. The distribution of modularity values for the partitions produced by each algorithm on the 500 LFR networks (top panel) and the 500 ABCD networks (bottom panel). The algorithms are sorted from right to left based on having a more desirable median modularity.

LFR and 500 ABCD instances. Figure 10 shows scatter plots of the run time (y axis) for each of the 30 algorithms on LFR networks, based on the number of edges (x axis). The AL algorithm is the fastest algorithm whose run times are all in the order of 1×10^{-5} s and are therefore not visible in the plot. The run times of Bayan have the largest variation ranging mostly in the orders of 1×10^{-3} to 1×10^{3}, with larger networks typically taking a longer time. For most algorithms except Infomap, we observe a clear increase in run time as the number of edges goes up. The algorithms Bayan, GA, GemSec, and Belief, are the slowest algorithms on the LFR networks. There are many instances for which Bayan has the longest run time.

Figure 11 illustrates the run times on ABCD networks for each the 30 algorithms in a scatter plot. The patterns are quite similar to those observed in Figure 10. The AL algorithm is again observed to be the fastest algorithm with run times around 1×10^{-5} s that are not visible in the scatter plot. The run times of Bayan have the largest variation ranging mostly in the orders of 1×10^{-2} to 1×10^{4} for the ABCD instances (which are on average larger networks than our LFR instances). Except for Infomap, we observe a clear increase in run time as the number of edges goes up. The four algorithms Bayan, GA, GemSec, and Belief are the slowest algorithms on the ABCD networks as well. There is a difference of several orders of magnitude between the run times of some of these CD algorithms on the same network instances.

Next, we focus on the efficiency of Bayan and two alternative exact approaches for obtaining maximum-modularity partitions.

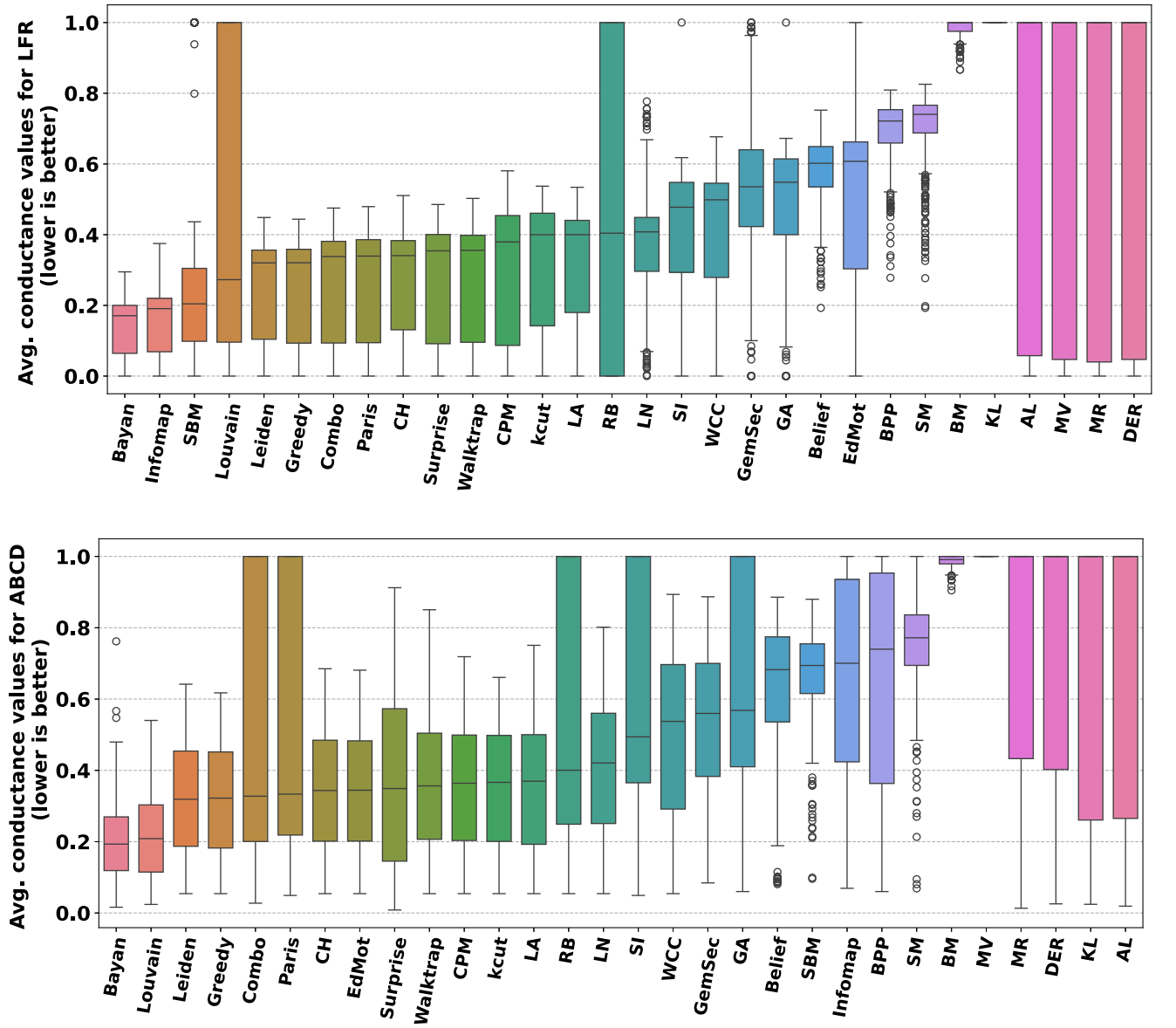

FIG. 6. The distribution of average conductance values for the partitions produced by each algorithm on the 500 LFR networks (top panel) and the 500 ABCD networks (bottom panel). The algorithms are sorted from right to left based on having a more desirable median of average conductance.

B. Comparison of three approaches for exact modularity maximization based on time and optimality

In this section, we compare Bayan with two alternative approaches for exact modularity maximization based on the run time and highest modularity found within a maximum limited time.

The first alternative approach is using the function community_optimal_modularity [91] from the igraph library [92] (IG for short).[7] IG calls the open-source solver, GNU Linear Programming Kit (GLPK), to solve an IP formulation [30] of the modularity maximization problem and returns an optimal partition when/if the optimization problem is solved.

The second alternative approach is solving the sparse IP formulation of the modularity maximization problem [27] from Eq. (2) using the commercial solver Gurobi [49]. Recent performance assessments from March 2024 show Gurobi to be one of the fastest mathematical solvers for IP models [93]. This sets a high performance baseline to compare Bayan against. Out-of-the-box Gurobi IP has been used in [5] to solve Eq. (2) and was shown to be feasible for networks with up to 2812 edges, given a relatively long computation time.

To compare IG, Gurobi IP, and Bayan, we consider 94 structurally diverse networks including real networks from different contexts and random graphs. These 94 instances

[7]The IG algorithm has not been explicitly proposed as a method for CD, but as a complementary tool in the igraph library [92].

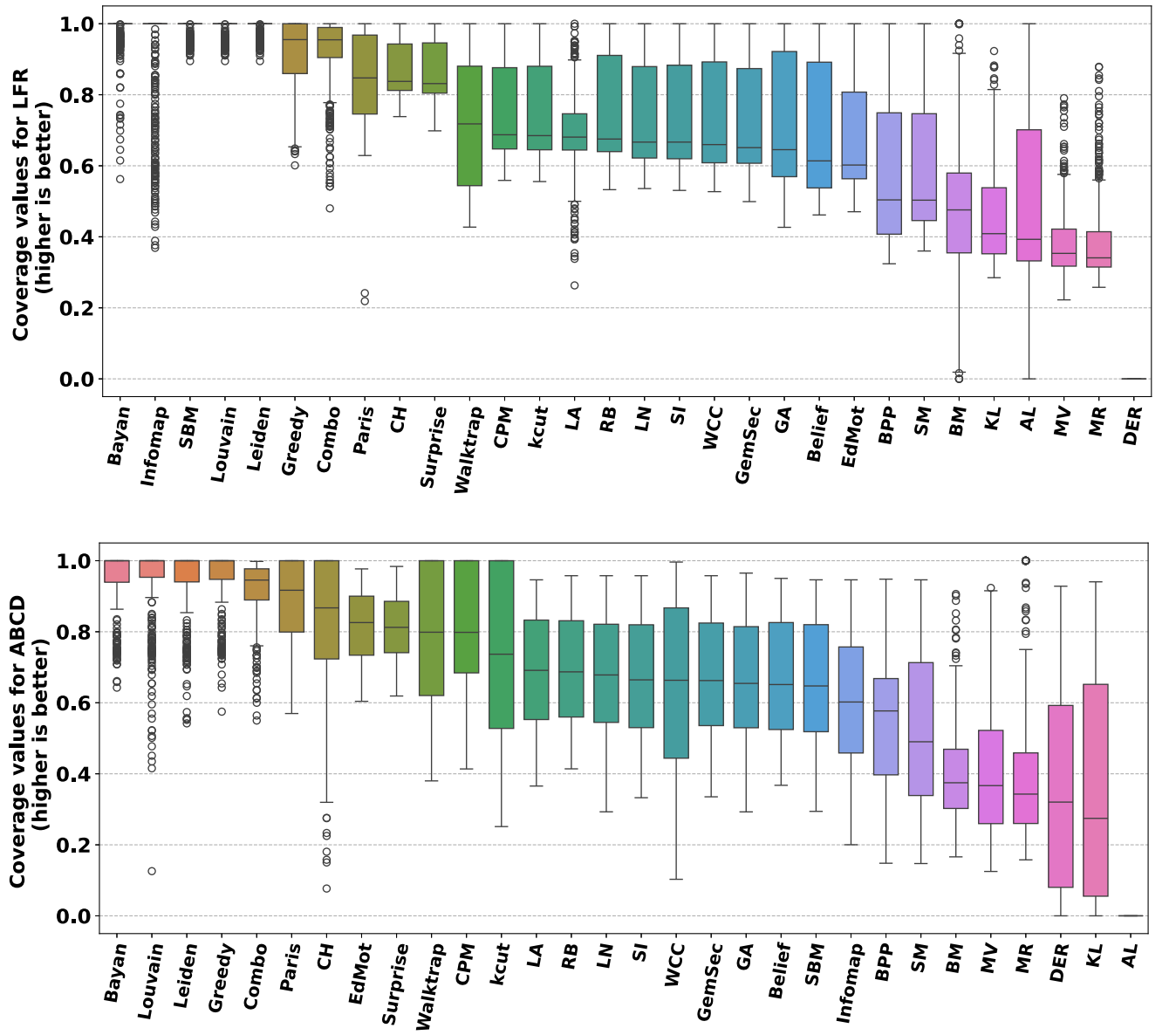

FIG. 7. The distribution of coverage values for the partitions produced by each algorithm on the 500 LFR networks (top panel) and the 500 ABCD networks (bottom panel). The algorithms are sorted from right to left based on having a more desirable median coverage.

include 74 real networks with no more than 8405 edges as well as 10 Erdős-Rényi graphs and 10 Barabási-Albert graphs.[8] For each of these 94 instances, we use Bayan, Gurobi IP, and IG with a time limit of 4 h and compare their empirical run times and objective function values (highest modularity found at termination).

There are ten instances for which none of the three methods finds a guaranteed optimal partition within the time limit of 4 h (see Table S3 in the SM). For these ten instances, Bayan returns a feasible partition with a higher modularity (49.4% higher on average) compared to the feasible partition from Gurobi IP at the time limit.[9]

Among the remaining 84 instances, Gurobi IP and IG fail to converge within the time limit, respectively, on 6 and 23 other instances, respectively. However, all 84 instances are solved to global optimality by Bayan (detailed results are provided in Tables S4, S5, and S6 in the SM). The six networks *adjnoun*, *london_transport*, *copenhagen*, *macaques*, *malaria_genes*, and *celegans_2019*[10] cannot be

[8]These 94 networks are loaded as simple undirected unweighted graphs. Additional details and information on accessing these 94 networks are provided in the SM.

[9]Differences in modularity values are reported not as a CD score/quality function, but to compare the objective function values of Bayan and Gurobi IP on the challenging instances.

[10]See the network repository Netzschleuder for more information on the networks.

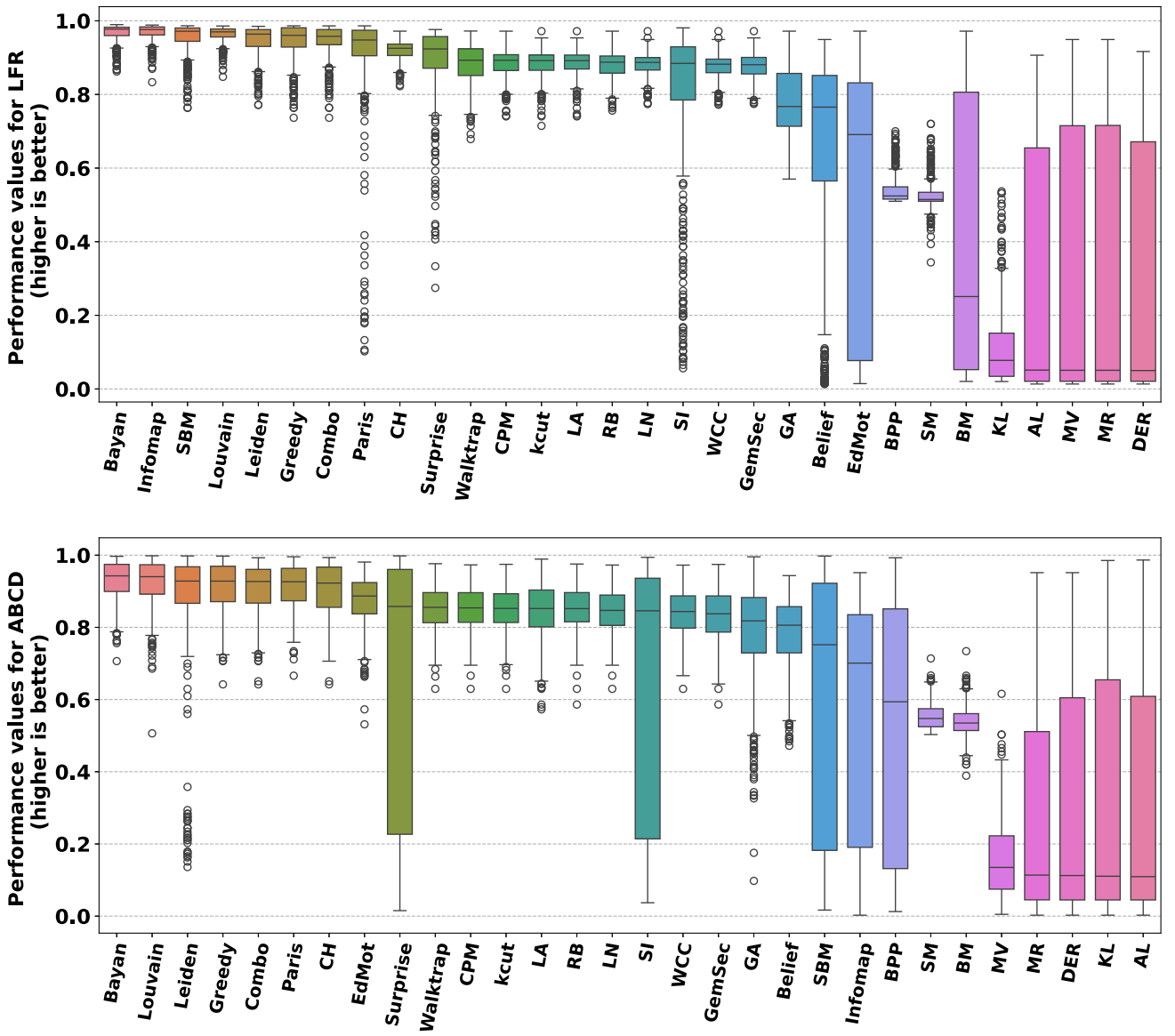

FIG. 8. The distribution of performance values for the partitions produced by each algorithm on the 500 LFR networks (top panel) and the 500 ABCD networks (bottom panel). The algorithms are sorted from right to left based on having a more desirable median performance.

solved by Gurobi IP or IG within the time limit. To our knowledge, the guaranteed maximum-modularity partitions for these six networks have not been previously determined [5,8,26–28,94,95].

On these 84 instances, the average run times of Bayan, Gurobi IP, and, IG are 295.08, 1110.60, and 4588.52 s, respectively. This implies that on average, Bayan is 815.52 s per instance (or more than 3.7 times) faster than the Gurobi IP, and 4289.15 s per instance (or more than 15.5 times) faster than the IG on these 84 instances. Both of these run-time comparisons are conservative because Bayan solves all 84 instances to global optimality while Gurobi IP and IG fail to terminate within the time limit for 6 and 23 networks, respectively. On all these failed cases, we have conservatively considered 4 h as the run time for Gurobi IP and IG while the actual run time may be substantially higher. Therefore, the actual performance comparison would be even more favorable to Bayan. The complete results on run times and modularity values of the three exact methods are provided in the SM (see Tables S3–S6).

Overall, these results indicate that Bayan is considerably faster than IG and Gurobi IP for most of the networks considered. The run-time advantage of Bayan makes a practical difference on the real network instances and especially the six challenging instances noted earlier which are solvable by Bayan but not solvable by IG or Gurobi IP (or any other exact method to the best of our knowledge).

X. DISCUSSION AND FUTURE DIRECTIONS

There are two fundamental questions to be answered regarding the Bayan algorithm:

(1) How suitable is Bayan for maximizing modularity?

(2) How suitable is Bayan for community detection?

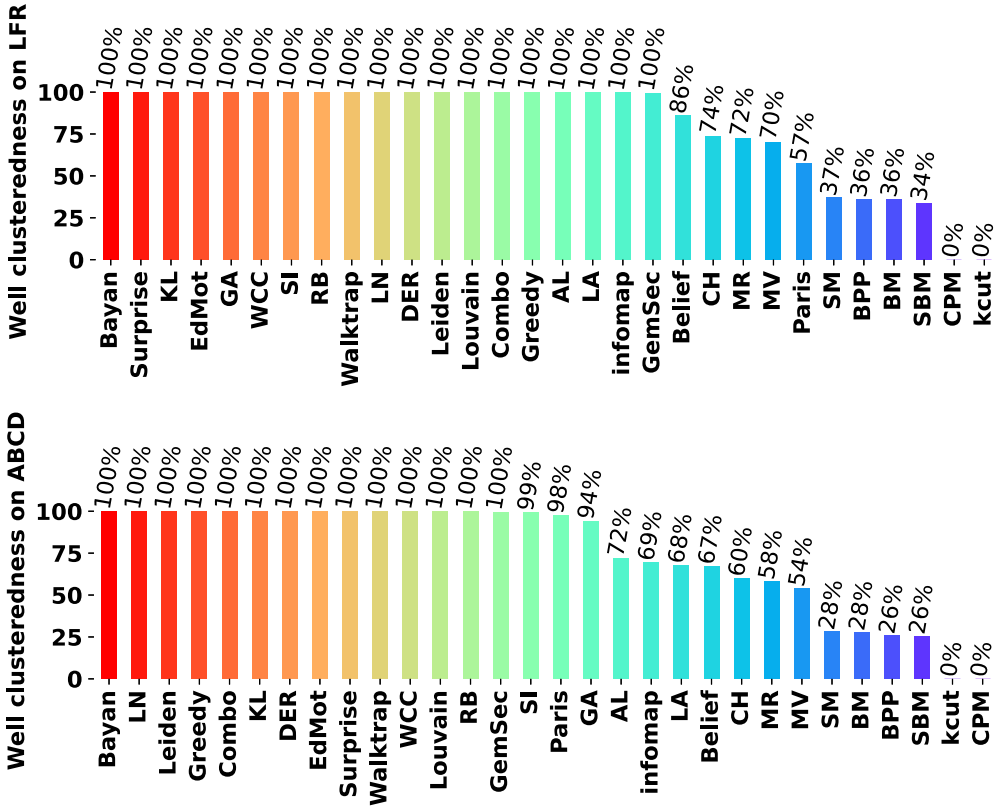

FIG. 9. The percentage of LFR (top panel) or ABCD (bottom panel) instances for which each algorithm produces a well-clustered graph. Higher values are better. The algorithms are sorted from right to left based on having a more desirable percentage of well clusteredness.

For the first question, Fig. 5 provides some answers. For more conclusive answers about global maximization of modularity, one can also refer to another study [50], where we have compared ten modularity-based methods based on the extent to which they globally maximized modularity. This included Bayan, eight modularity-based heuristics, and a graph neural network algorithm [89]. Using real and synthetic networks with fairly modular structure, we showed the following: At the cost of longer computation time, Bayan has the advantage of reaching global optimality at higher rates and returning partitions more similar to the globally optimal partitions, compared to the other nine modularity maximization algorithms. Bayan was followed by the Combo algorithm as the heuristic method demonstrating the most promising performance in returning partitions that were optimal or were similar to an optimal partition.

The results provided in Secs. VII and VIII allow us to answer the second question from a practical standpoint. The results in Secs. VII A–VII B demonstrate the comparative suitability of Bayan based on retrieving planted partitions of the LFR and ABCD benchmarks. Bayan was observed to be among the top algorithms in retrieving planted partitions across different mixing parameters. The results in Sec. VII C show that, for networks where node labels are aligned with the structure, Bayan retrieves partitions that are similar with node labels. Overall, the results in Sec. VII demonstrate the suitability of Bayan as a community detection method for networks with up to 3000 edges. Our results in Sec. VIII show Bayan's competitive advantages in comparison with 29 other methods based on five partition quality measures other than modularity: description length, average conductance, coverage, performance, and well clusteredness. In what follows, we discuss five key insights from our results.

a. Global modularity maximization retrieves planted partitions at rates higher than other existing algorithms. The results provided in Sec. VII indicate the practical relevance of maximum-modularity partitions in comparison with 29 other alternatives for CD. Despite the well-documented theoretical flaws of modularity [11], the high accuracy and stability of globally maximum-modularity partitions (shown in this study) are not challenged by other existing methods for community detection.

b. In modularity optimization, guaranteed closeness to optimality matters. The results from Figs. 1 and 2 and 4–9 offer additional insights when we focus on the performance of the nine algorithms that attempt to maximize modularity. The comparative performance of nine modularity-based algorithms (Bayan, greedy, Louvain, LN, Combo, Belief, Paris, Leiden, and EdMot) shows the practical benefits of global optimization in the context of using modularity for CD. The practical benefit of global optimization is in both achieving better performance in retrieving planted communities and achieving better partition quality measures (other than modularity). The key lesson learned is that in the context of community detection through modularity optimization, guaranteed closeness to optimality matters. This crucial, yet often overlooked [4], lesson was foreshadowed in [25, pp. 012811–012815] which reads "to stress the importance of

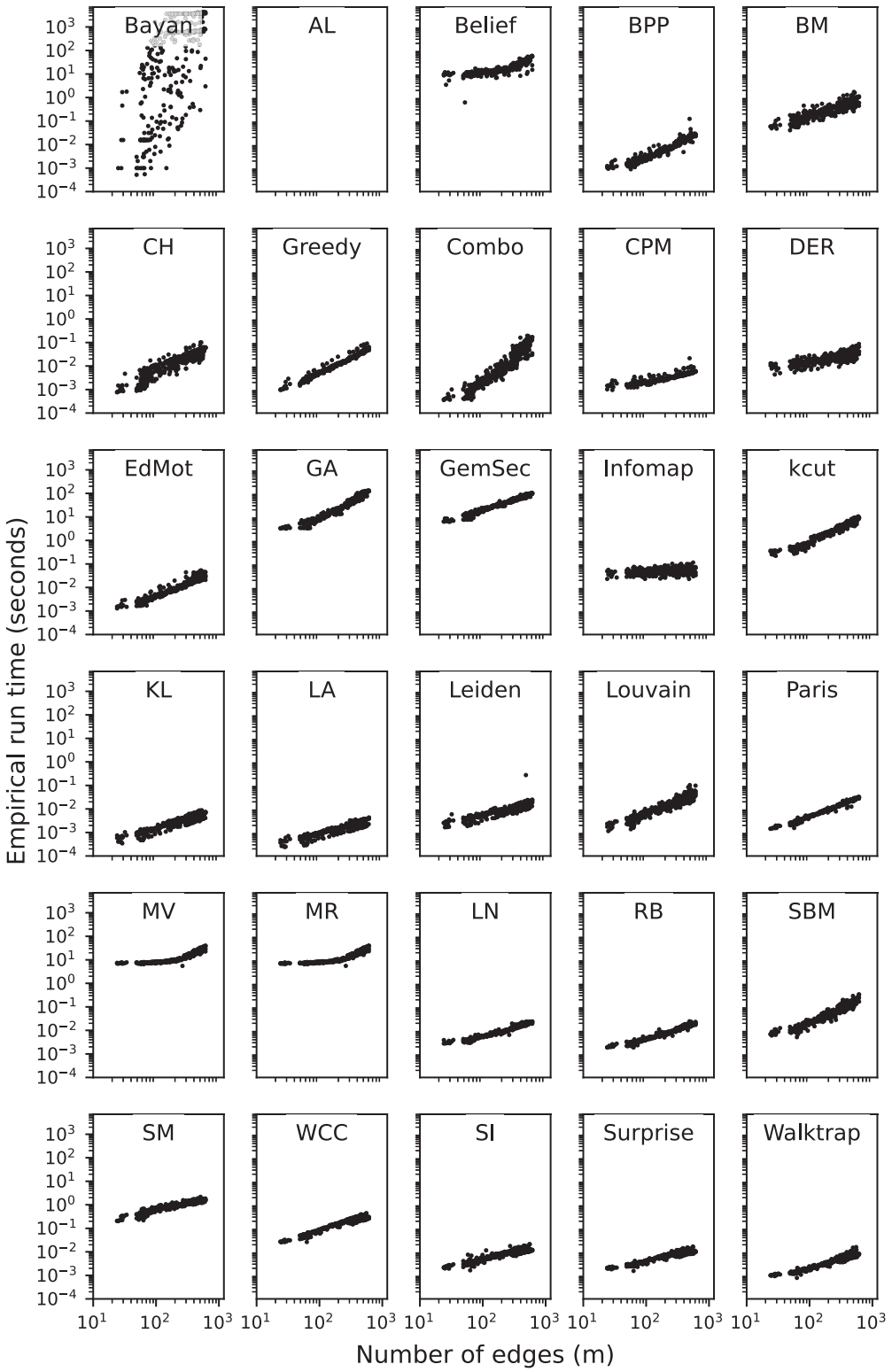

FIG. 10. The run time of 30 algorithms on 500 LFR networks. Both axes are scaled logarithmically (base 10).

looking for even the minor gains in the modularity score, [...] relatively small changes in this partition quality function can be reflected by macroscopic variation of the communities involved."

c. Exact modularity maximization is within reach for small networks, but requires substantially longer computation. The scatter plots provided in Figs. 10 and 11 demonstrate the substantial differences between the run times of algorithms. As expected and being an exact method, Bayan takes considerably longer than most heuristic methods. However, it is, on average, more than 15.5 times faster than using the open-source IP solver, GLPK, and more than 3.7 times faster than

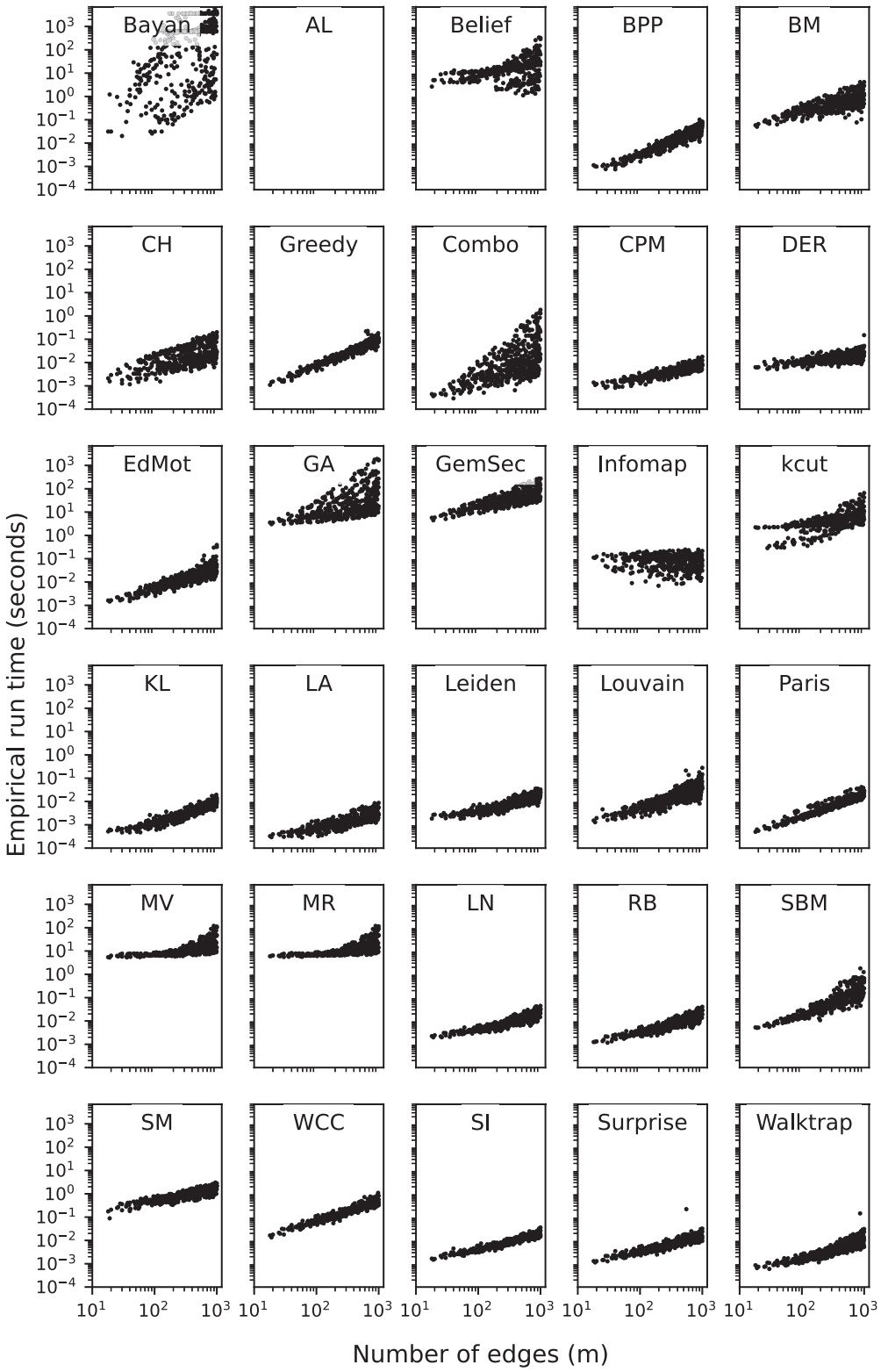

FIG. 11. The run time of 30 algorithms on 500 ABCD networks. Both axes are scaled logarithmically (base 10).

using the commercial solver, Gurobi, for modularity maximization. Also, we reiterate that Bayan (as well as any attempt at solving an NP-hard problem exactly) does not scale to large instances and suffers from asymptotic worst-case running times that are far from ideal. The NP-hardness of the problem implies that efficient running times of scalable heuristics (like Leiden) cannot be beaten by developing an exact method for solving it; scaling to large networks has not been a purpose of this study but is the topic of tempting and recurring questions. Having said that, exact and approximate optimization has provided more accurate and reliable solutions to increasingly larger instances of many similar NP-hard graph partitioning

problems [96–100] and modularity maximization is not an exception. Using decades-old heuristics (including methods that were groundbreaking for their time) is not justified for small networks and comes at the cost of lower performance.

d. Bayan's mathematically rigorous operationalization of modularity maximization is a reliable choice for CD in small networks with up to 3000 edges. While it is tempting to interpret our results as suggesting modularity is a better objective function than the objective functions of other optimization-based algorithms considered, we refrain from drawing such a conclusion because our comparative results of algorithms in Figs. 1 and 2 and 4–9 are confounded by the difference in the objective functions and the difference between mathematically rigorous optimization vs local/greedy/heuristic optimization. We simply recommend using mathematically rigorous optimization in optimization-based tasks for small input. Our results justify the exact optimization approach for modularity where optimality matters [25] and suboptimality has a disproportionate cost [5]. Future research may investigate the potential advantages of using global optimization and guaranteed approximations for other objective functions (e.g., Markov stability [43], description length [16], modularity density [21], surprise [20,22], or a new objective function inspired by the map equation [14]) for CD. To the best of our knowledge, exact optimization of these alternative objectives is underexplored, and therefore each has the potential to outperform maximum-modularity partitions (and in turn outperform most other algorithms considered in this study) in retrieval tests and partition quality measures. We hope that our publicly available data and algorithm facilitate these prospective advances in the field. As a specialized algorithm [44] for accurate CD in small networks, Bayan maximizes modularity in networks with up to 3000 edges (in their largest connected component) and approximates maximum modularity in slightly larger instances on ordinary computers. Prospective advances in IP solvers (like Gurobi which is used in Bayan and is actively improved) pushes this limit further. Exact maximization of modularity for larger networks may require a substantially longer time or high-performance computing resources. For slightly larger networks on an ordinary computer, one may run Bayan with a desired optimality gap tolerance (e.g., 0.1) or a desired time limit so that it returns a partition with a guaranteed proximity to the maximum modularity.

e. Bayan contributes computational capabilities of both methodological and empirical relevance. On the methodological side, the contributions of Bayan are threefold. (1) Bayan offers a principled way of quantifying the optimality of current and future modularity-based heuristics for CD. (2) Bayan demonstrates the advantages of a mathematically rigorous implementation of a long-lasting yet paradoxically underexplored optimization idea. (3) Bayan reaffirms that the practical relevance of modularity as an objective function, despite its theoretical flaws [11], is yet to be challenged by a practical method that is more accurate and stable. On the empirical side, Bayan tackles a practical task in network science for small input, thereby improving upon open-access computational capabilities for a narrow but relevant set of tasks in networked systems.

A PYTHON implementation of the Bayan algorithm (bayanpy) is publicly available through the package installer for PYTHON (pip). All network data used in the experiments and evaluations of the Bayan algorithm are available in a FigShare data repository [101].

ACKNOWLEDGMENTS

The authors acknowledge Sanchaai Mathiyarasan for technical assistance and Giulio Rossetti, Rémy Cazabet, Pawel Pralat, and Tiago P. Peixoto for helpful discussions. This study has been supported by the Data Sciences Institute at the University of Toronto.

S.A. conceptualized the work; S.A. and H.C. curated the data; S.A. and M.M performed the formal analysis; S.A. aquired the funding; S.A. and M.M. performed investigations; S.A. and M.M. designed the methodology; S.A. administered the project; S.A., M.M., and H.C. aquired resources; S.A., M.M., and H.C. developed the software; S.A. and M.M. supervised the work; S.A., M.M., and H.C. validated the results; M.M. produced the figures; S.A. and M.M. wrote and prepared the original draft; S.A., M.M., and H.C. wrote, reviewed and edited the paper.

APPENDIX A: DETAILED RESULTS OF 30 COMMUNITY DETECTION ALGORITHMS ON 500 LFR GRAPHS

Our retrieval tests on LFR benchmarks produce 100 AMI values for each of the 30 algorithms in each of the 5 LFR experiment settings. The average AMIs are reported in Table I.

TABLE I. Average AMIs between the partitions of each algorithm and the ground-truth partitions of LFR graphs. Each column shows the average AMI for 100 LFR graphs generated with a specific value for the parameter μ.

Algorithm \ μ	0.01	0.1	0.3	0.5	0.7
Bayan	0.914977	0.865469	0.222458	0.120004	0.048301
Belief	0.807955	0.779013	0.15946	0.048613	0.021865
BPP	0.883869	0.682873	5.39×10^{-17}	5.3×10^{-17}	5.3×10^{-17}
BM	0.884644	0.680429	5.39×10^{-17}	5.3×10^{-17}	5.3×10^{-17}
CH	0.900439	0.660127	0.114033	0.043228	0.015612
Combo	0.911275	0.81159	0.20871	0.10669	0.043502
CPM	5.57×10^{-14}	-9.3×10^{-15}	-8.4×10^{-15}	-2.4×10^{-14}	-1.7×10^{-14}
DER	0.566681	0.511116	0.090219	0.048349	0.023683
EdMot	0.812163	0.799894	0.215178	0.116701	0.046678
GA	0.729344	0.6357	0.178047	0.098336	0.03072
GemSec	0.471457	0.464521	0.159129	0.099382	0.036814
Greedy	0.906994	0.803937	0.194709	0.102737	0.046088
Infomap	0.892984	0.7848	0.217829	0.124187	0.045835
kcut	0.026286	0.004739	−0.00038	−0.00243	−0.00216
KL	0.507826	0.457056	0.087676	0.036781	0.016813
AL	0.747412	0.68016	0.209216	0.111949	0.033243
LA	0.839582	0.716382	0.204512	0.095657	0.03089
Leiden	0.869695	0.803111	0.215096	0.111227	0.042612
Louvain	0.880473	0.798383	0.202008	0.108993	0.038464
MV	0.224506	0.693942	0.178752	0.104611	0.038645
MR	0.400319	0.61916	0.207398	0.116707	0.04327
Paris	0.992753	0.457702	0.052421	0.03215	0.017041
LN	0.900007	0.805143	0.213839	0.110176	0.043872
RB	0.851303	0.808343	0.233574	0.12645	0.045674
SBM	0.785009	0.60195	5.39×10^{-17}	5.3×10^{-17}	5.3×10^{-17}
SM	0.78293	0.637965	5.39×10^{-17}	5.3×10^{-17}	5.3×10^{-17}
SI	0.565777	0.465077	0.193266	0.102577	0.038334
Surprise	0.015216	0.014727	0.003753	0.001559	−0.00053
Walktrap	0.948559	0.903521	0.219744	0.117848	0.035803
WCC	0.417899	0.452965	0.162383	0.107579	0.038922

APPENDIX B: DETAILED AMI RESULTS OF 30 COMMUNITY DETECTION ALGORITHMS ON 500 ABCD GRAPHS

Our retrieval tests on ABCD benchmarks produce 100 AMI values for each of the 30 algorithms in each of the 5 ABCD experiment settings. The average AMIs are reported in Table II.

TABLE II. Average AMIs between the partitions of each algorithm and the ground-truth partitions of ABCD graphs. Each column shows the average AMI for 100 ABCD graphs generated with a specific value for the parameter ξ.

Algorithm \ ξ	0.1	0.3	0.5	0.7	0.9
Bayan	0.904262	0.768595	0.516145	0.172341	0.039422
Belief	0.872381	0.68601	0.402638	0.044745	0.006225
BPP	0.757256	0.476529	0.042167	0	0
BM	0.805145	0.505006	0.040564	6.63×10^{-6}	0
CH	0.887096	0.64501	0.09825	0.020215	0.006525
Combo	0.906193	0.763176	0.497164	0.170715	0.033017
CPM	-1.21×10^{-14}	1.16×10^{-14}	2.72×10^{-14}	-5.62×10^{-15}	-3.96×10^{-14}
DER	0.529986	0.405244	0.220585	0.069547	0.01587
EdMot	0.845659	0.684886	0.450476	0.163103	0.044221
GA	0.670259	0.416827	0.217894	0.139202	0.035661
GemSec	0.638734	0.480496	0.26845	0.108372	0.029334
Greedy	0.878954	0.675435	0.37166	0.135652	0.025767
Infomap	0.830611	0.670542	0.378866	0.102234	0.020956
kcut	−0.0019	0.000836	0.001912	−0.00024	0.002985
KL	0.452558	0.384772	0.244365	0.072133	0.014224
AL	0.74617	0.628092	0.233951	0.069981	0.011225
LA	0.764068	0.573115	0.167497	0.05184	0.010451
Leiden	0.901538	0.744484	0.492217	0.161479	0.035678
Louvain	0.877696	0.713213	0.453911	0.147747	0.039236
MV	0.870493	0.705155	0.408535	0.14907	0.040766
MR	0.744999	0.615564	0.424862	0.181526	0.040006
Paris	0.799557	0.511702	0.265896	0.108647	0.03216
LN	0.902227	0.735016	0.482786	0.158218	0.043241
RB	0.897224	0.73453	0.500524	0.149088	0.034994
SBM	0.72774	0.46625	0.077345	−0.0005	−0.00352
SM	0.708577	0.44547	0.040516	−0.00614	0.005209
SI	0.598734	0.478368	0.33308	0.150461	0.04024
Surprise	0.790037	0.645862	0.42343	0.165148	0.049853
Walktrap	0.899926	0.745977	0.475442	0.170127	0.032478
WCC	0.524841	0.474244	0.346609	0.1596	0.041969

[1] M. T. Schaub, J.-C. Delvenne, M. Rosvall, and R. Lambiotte, Appl. Network Sci. **2**, 4 (2017).

[2] S. Fortunato and M. E. J. Newman, Nat. Phys. **18**, 848 (2022).

[3] M. E. J. Newman, Proc. Natl. Acad. Sci. USA **103**, 8577 (2006).

[4] V. Cohen-Addad, A. Kosowski, F. Mallmann-Trenn, and D. Saulpic, in *Advances in Neural Information Processing Systems 33 (NeurIPS '20)*, edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Curran Associates, Inc., New York, 2020), Vol. 33, pp. 4055–4066.

[5] S. Aref, M. Mostajabdaveh, and H. Chheda, in *Computational Science (CCS 2023)*, edited by J. Mikyška, C. de Mulatier, M. Paszynski, V. V. Krzhizhanovskaya, J. J. Dongarra, and P. M. Sloot (Springer Nature Switzerland, Cham, 2023), pp. 612–626.

[6] T. Kawamoto and Y. Kabashima, Phys. Rev. E **99**, 010301(R) (2019).

[7] B. H. Good, Y.-A. deMontjoye, and A. Clauset, Phys. Rev. E **81**, 046106 (2010).

[8] T. N. Dinh, X. Li, and M. T. Thai, in *2015 IEEE International Conference on Data Mining* (IEEE, Piscataway, NJ, 2015), pp. 101–110.

[9] M. E. J. Newman, Phys. Rev. E **94**, 052315 (2016).

[10] S. Fortunato and D. Hric, Phys. Rep. **659**, 1 (2016).

[11] T. P. Peixoto, *Elements in the Structure and Dynamics of Complex Networks* (Cambridge University Press, Cambridge, UK, 2023).

[12] S. Fortunato and M. Barthélemy, Proc. Natl. Acad. Sci. USA **104**, 36 (2007).

[13] M. Rosvall and C. T. Bergstrom, Proc. Natl. Acad. Sci. USA **104**, 7327 (2007).

[14] M. Rosvall and C. T. Bergstrom, Proc. Natl. Acad. Sci. USA **105**, 1118 (2008).

[15] B. Karrer and M. E. J. Newman, Phys. Rev. E **83**, 016107 (2011).

[16] T. P. Peixoto, Phys. Rev. E **89**, 012804 (2014).
[17] L. Zhang and T. P. Peixoto, Phys. Rev. Res. **2**, 043271 (2020).
[18] X. Liu, B. Yang, H. Chen, K. Musial, H. Chen, Y. Li, and W. Zuo, ACM Trans. Knowl. Discovery Data **15**, 1 (2021).
[19] B. Serrano and T. Vidal, Pattern Recog. **152**, 110487 (2024).
[20] V. A. Traag, R. Aldecoa, and J.-C. Delvenne, Phys. Rev. E **92**, 022816 (2015).
[21] K. Sato and Y. Izunaga, Comput. Operations Res. **106**, 236 (2019).
[22] E. Marchese, G. Caldarelli, and T. Squartini, Commun. Phys. **5**, 132 (2022).
[23] A. Paul and A. Dutta, Expert Syst. Appl. **206**, 117794 (2022).
[24] H. Liu, J. Wei, and T. Xu, Expert Syst. Appl. **231**, 120748 (2023).
[25] S. Sobolevsky, R. Campari, A. Belyi, and C. Ratti, Phys. Rev. E **90**, 012811 (2014).
[26] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron, and L. Liberti, Phys. Rev. E **82**, 046112 (2010).
[27] T. N. Dinh and M. T. Thai, Internet Math. **11**, 181 (2015).
[28] S. Sobolevsky, A. Belyi, and C. Ratti, arXiv:1712.05110.
[29] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevE.110.044315 for Supplemental information, technical details, and additional results.
[30] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner, IEEE Trans. Knowl. Data Eng.g **20**, 172 (2007).
[31] X. Zhao, J. Liang, and J. Wang, Inf. Sci. **551**, 358 (2021).
[32] A. Clauset, M. E. J. Newman, and C. Moore, Phys. Rev. E **70**, 066111 (2004).
[33] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, J. Stat. Mech. (2008) P10008.
[34] E. A. Leicht and M. E. J. Newman, Phys. Rev. Lett. **100**, 118703 (2008).
[35] P. Zhang and C. Moore, Proc. Natl. Acad. Sci. USA **111**, 18144 (2014).
[36] T. Bonald, B. Charpentier, A. Galland, and A. Hollocou, in *MLG 2018-14th International Workshop on Mining and Learning with Graphs* (Association for Computing Machinery, New York, 2018).
[37] V. A. Traag, L. Waltman, and N. J. van Eck, Sci. Rep. **9**, 5233 (2019).
[38] P.-Z. Li, L. Huang, C.-D. Wang, and J.-H. Lai, in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), pp. 479–487.
[39] N. X. Vinh, J. Epps, and J. Bailey, J. Mach. Learn. Res. **11**, 2837 (2010).
[40] A. A. Hagberg, D. A. Schult, and P. J. Swart, in *Proceedings of the 7th PYTHON in Science Conference*, edited by G. Varoquaux, T. Vaught, and J. Millman (Los Alamos National Laboratory, Los Alamos, NM, 2008), pp. 11–15.
[41] T. P. Peixoto, FigShare, 2014, https://doi.org/10.6084/m9.figshare.1164194.
[42] G. Rossetti, L. Milli, and R. Cazabet, Appl. Network Sci. **4**, 52 (2019).
[43] A. Arnaudon, D. J. Schindler, R. L. Peach, A. Gosztolai, M. Hodges, M. T. Schaub, and M. Barahona, ACM Trans. Math. Softw. **50**, 1 (2024).
[44] L. Peel, D. B. Larremore, and A. Clauset, Sci. Adv. **3**, e1602548 (2017).
[45] A. P. Christensen, L. E. Garrido, K. Guerra-Peña, and H. Golino, Behav. Res. Methods **56**, 1485 (2024).
[46] M. J. Brusco, D. Steinley, and A. L. Watts, Behav. Res. Methods **55**, 3549 (2023).
[47] A. Lancichinetti and S. Fortunato, Phys. Rev. E **84**, 066122 (2011).
[48] G. Agarwal and D. Kempe, Eur. Phys. J. B **66**, 409 (2008).
[49] Gurobi Optimization Inc., Gurobi Optimizer Reference Manual (2023), https://gurobi.com/documentation/10.0/refman/index.html date accessed 16 Feb 2023.
[50] S. Aref and M. Mostajabdaveh, J. Comput. Sci. **78**, 102283 (2024).
[51] A. Arenas, J. Duch, A. Fernández, and S. Gómez, New J. Phys. **9**, 176 (2007).
[52] L. H. N. Lorena, M. G. Quiles, and L. A. N. Lorena, in *International Conference on Computational Science and Its Applications* (Springer, New York, 2019), pp. 757–769.
[53] M. Jerdee, A. Kirkley, and M. E. J. Newman, arXiv:2307.01282.
[54] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, J. Stat. Mech. (2005) P09008.
[55] A. J. Gates, I. B. Wood, W. P. Hetrick, and Y.-Y. Ahn, Sci. Rep. **9**, 8574 (2019).
[56] Z. Roozbahani, J. Rezaeenour, and A. Katanforoush, J. Comput. Sci. **67**, 101962 (2023).
[57] D. K. Singh, S. Nandi, T. Chakraborty, and P. Choudhury, J. Comput. Sci. **61**, 101634 (2022).
[58] M. E. J. Newman, G. T. Cantwell, and J.-G. Young, Phys. Rev. E **101**, 042304 (2020).
[59] A. Mahmoudi and D. Jemielniak, Sci. Rep. **14**, 9021 (2024).
[60] A. Lancichinetti, S. Fortunato, and F. Radicchi, Phys. Rev. E **78**, 046110 (2008).
[61] B. Kamiński, P. Prałat, and F. Théberge, Network Science **9**, 153 (2021).
[62] P. Miasnikof, A. Y. Shestopaloff, A. J. Bonner, Y. Lawryshyn, and P. M. Pardalos, J. Complex Networks **8**, cnaa012 (2020).
[63] R. Kannan, S. Vempala, and A. Vetta, J. ACM **51**, 497 (2004).
[64] S. Fortunato, Phys. Rep. **486**, 75 (2010).
[65] M. H. Hansen and B. Yu, J. Am. Stat. Assoc. **96**, 746 (2001).
[66] C. L. Staudt, M. Hamann, A. Gutfraind, I. Safro, and H. Meyerhenke, Appl. Network Sci. **2**, 36 (2017).
[67] G. M. Slota, J. W. Berry, S. D. Hammond, S. L. Olivier, C. A. Phillips, and S. Rajamanickam, in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '19)* (Association for Computing Machinery, New York, NY, 2019).
[68] M. Girvan and M. E. J. Newman, Proc. Natl. Acad. Sci. USA **99**, 7821 (2002).
[69] B. W. Kernighan and S. Lin, Bell Syst. Tech. J. **49**, 291 (1970).
[70] C. Biemann, in *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing (TextGraphs-1)* (Association for Computational Linguistics, 2006), pp. 73–80.
[71] J. Reichardt and S. Bornholdt, Phys. Rev. E **74**, 016110 (2006).
[72] P. Pons and M. Latapy, J. Graph Algorithms Appl. **10**, 191 (2006).
[73] J. Ruan and W. Zhang, in *Seventh IEEE International Conference on Data Mining (ICDM 2007)* (IEEE, New York, 2007), pp. 643–648.

[74] U. N. Raghavan, R. Albert, and S. Kumara, Phys. Rev. E **76**, 036106 (2007).
[75] C. Pizzuti, in *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature — PPSN X - Volume 5199* (Springer-Verlag, Berlin, Heidelberg, 2008), pp. 1081–1090.
[76] G. Cordasco and L. Gargano, in *2010 IEEE International Workshop On Business Applications of Social Network Analysis (BASNA)* (IEEE, New York, 2010), pp. 1–8.
[77] V. A. Traag, P. Van Dooren, and Y. Nesterov, Phys. Rev. E **84**, 016114 (2011).
[78] V. A. Traag, G. Krings, and P. Van Dooren, Sci. Rep. **3**, 2930 (2013).
[79] A. Prat-Pérez, D. Dominguez-Sal, and J.-L. Larriba-Pey, in *Proceedings of the 23rd International Conference on World Wide Web* (Association for Computing Machinery, New York, 2014), pp. 225–236.
[80] M. Kozdoba and S. Mannor, Adv. Neural Inf. Process. Syst. **28**, 1 (2015).
[81] B. Rozemberczki, R. Davies, R. Sarkar, and C. Sutton, in *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (Association for Computing Machinery, New York, NY, 2019), pp. 65–72.
[82] M. Friedman, J. Am. Stat. Assoc. **32**, 675 (1937).
[83] J. D. Li, J. Stat. Plann. Inference **138**, 1521 (2008).
[84] D. Hric, R. K. Darst, and S. Fortunato, Phys. Rev. E **90**, 062805 (2014).
[85] J. Yang and J. Leskovec, Knowl. Inf. Syst. **42**, 181 (2015).
[86] M. E. J. Newman and A. Clauset, Nat. Commun. **7**, 11863 (2016).
[87] T. Kawamoto and Y. Kabashima, Phys. Rev. E **97**, 022315 (2018).
[88] S. Chen, Z.-Z. Wang, L. Tang, Y.-N. Tang, Y.-Y. Gao, H.-J. Li, J. Xiang, and Y. Zhang, PLoS One **13**, e0205284 (2018).
[89] S. Sobolevsky and A. Belyi, Appl. Network Sci. **7**, 63 (2022).
[90] K. Steinhaeuser and N. V. Chawla, Pattern Recognit. Lett. **31**, 413 (2010).
[91] https://igraph.org/python/doc/api/igraph.Graph.html.
[92] G. Csardi and T. Nepusz, InterJournal, Complex Systems **1**, 1695 (2006).
[93] M. Miltenberger, Interactive visualizations of Mittelmann benchmarks, 2024, https://mattmilten.github.io/mittelmann-plots/ date accessed: 2024-03-28.
[94] A. Miyauchi and Y. Miyamoto, Eur. Phys. J. B **86**, 302 (2013).
[95] S. Cafieri, A. Costa, and P. Hansen, Ann. Oper. Res. **222**, 213 (2014).
[96] S. Aref, A. J. Mason, and M. C. Wilson, Networks **75**, 95 (2020).
[97] S. Aref and Z. P. Neal, Sci. Rep. **11**, 19939 (2021).
[98] F. Bonchi, D. García-Soriano, and F. Gullo, *Correlation Clustering* (Morgan & Claypool, San Rafael, CA, 2022).
[99] A. Belyi, S. Sobolevsky, A. Kurbatski, and C. Ratti, Math. Methods Oper. Res. **98**, 269 (2023).
[100] I. Sukeda, A. Miyauchi, and A. Takeda, Eur. J. Oper. Res. **309**, 516 (2023).
[101] S. Aref, H. Chheda, and M. Mostajabdaveh, Dataset of networks used in assessing the Bayan algorithm for community detection (2023), figShare https://doi.org/10.6084/m9.figshare.22442785.