PLOS COMPLEX SYSTEMS

RESEARCH ARTICLE

Troika algorithm: Approximate optimization for accurate clique partitioning and clustering of weighted networks

Samin Aref*, Boris Ng

Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario, Canada

* s.aref@utoronto.ca

OPEN ACCESS

Citation: Aref S, Ng B (2025) Troika algorithm: Approximate optimization for accurate clique partitioning and clustering of weighted networks. PLoS Complex Syst 2(9): e0000062. https://doi.org/10.1371/journal.pcsy.0000062

Editor: Hocine Cherifi, Université de Bourgogne: Universite de Bourgogne, FRANCE

Received: January 20, 2025

Accepted: July 15, 2025

Published: September 10, 2025

Data availability statement: All data used in this study is provided in the manuscript.

Funding: The author(s) received no specific funding for this work.

Competing interests: The authors have declared that no competing interests exist.

Abstract

Clique partitioning is the fundamental network clustering task of identifying an optimal node partition for a weighted graph according to the real-valued edge weights. An optimal partition is one that maximizes the sum of within-cluster edge weights over all possible node partitions. This paper introduces a novel approximation algorithm, named Troika, to solve this NP-hard problem in small to mid-sized networks for instances of theoretical and practical relevance. Troika uses a branch-and-cut scheme for branching on node triples to find a partition that is within a user-specified optimality gap tolerance. Troika offers advantages over alternative methods for clique partitioning like heuristics and integer programming solvers. Unlike heuristics, Troika returns solutions within a guaranteed proximity to global optimality. Compared to the integer programming solver, Gurobi, Troika is faster for most benchmark instances. Besides its advantages for solving the clique partitioning problem, we demonstrate the applications of Troika in community detection and portfolio analysis. Troika returns partitions with higher proximity to optimal compared to eight modularity-based community detection algorithms. When used on networks of correlations among stocks, Troika reveals the dynamic changes in the structure of portfolio networks including downturns from the 2008 financial crisis and the reaction to the COVID-19 pandemic. Our comprehensive results based on benchmarks from the literature and new real and random networks point to Troika as a reliable and accurate method for solving clique partitioning instances with up to 5000 edges on standard hardware.

Author summary

Clique partitioning is an unsupervised network clustering problem with applications across various fields. It involves partitioning the nodes of a real-valued weighted graph to maximize the total within-cluster edge weights. This paper introduces the *Troika*

algorithm for approximating the globally optimal solution of the clique partitioning problem on small to mid-sized networks. Utilizing a branch-and-cut scheme for branching on node triples, Troika provides solutions within a user-specified optimality gap tolerance. Evaluations on benchmark datasets reveal Troika's superior performance compared to existing methods. Troika strikes a tradeoff between speed and solution quality: it is much faster than the Gurobi integer programming solver, while it delivers solutions more accurate than heuristic solutions. Beyond clique partitioning, Troika excels in optimization-based community detection and portfolio analysis, outperforming eight modularity-based algorithms and uncovering dynamic changes to financial portfolios, including the 2008 financial crisis and COVID-19 impacts. Troika reliably provides optimal and near-optimal solutions for graphs with up to 5,000 edges on standard hardware.

1. Introduction

Clustering is the unsupervised task of grouping objects based on their similarities. This task becomes *network clustering* if the objects are modeled as nodes of a graph and their pairwise similarities are modeled as weighted edges. The *clique partitioning* (CP) problem is a specific network clustering problem defined on undirected weighted graphs [1] that have positive and negative real values as edge weights. Network models that have signed edges appear in a wide range of contexts [2] including gene regulatory networks in biology, spin glass models in physics, portfolio networks in finance, networks of international relations, and social ties of opposite nature in social network analysis. Besides its wide applicability, clique partitioning is a problem of crucial importance because it is computationally challenging [3] despite its simple definition. For a weighted signed network as input, the CP problem is defined as clustering the nodes into a partition that maximizes the sum of within-cluster edge weights [4]. Clique partitioning is also important from a theoretical standpoint because other challenging optimization problems that are defined on unsigned networks (such as modularity-based community detection [5]) reduce to it.

The CP problem is NP-hard [3], so the quest for efficient approximation and exact algorithms has led to a variety of approaches including inexact approaches such as heuristics and meta-heuristics. These approaches are scalable to large networks, but do not provide any guarantee for solution quality. For small and mid-sized instances, there are also exact and approximation approaches including methods based on mathematical optimization models [1].

Related work

Grötschel and Wakabayashi formulated the CP problem as an integer linear program in 1989 [1]. In the past 30 years, a myriad of inexact algorithms has been proposed for the CP problem relying on heuristics and meta-heuristics such as simulated annealing [6], neighborhood search [7], iterated tabu search [8,9], and local search [10]. In 2021, Lu et al. [11] introduced a merge-divide memetic algorithm, which incorporates a merge-divide crossover operator along with simulated annealing-based local optimization and pool management. Their method uses a population-based approach to solve "challenging" CP instances, and is the first to have several complementary search components. According to their experiments, the algorithm has the ability to produce high-quality solutions, reporting improved best-known lower bounds for benchmark instances with 2000 nodes.

There are some applications of CP where optimization accuracy is not a concern because network instances are very large-scale. Named entity disambiguation is one such application from natural language processing which involves finding a cluster of candidates from a knowledge base which are likely to be the target of a name mentioned in a text [12]. In a recent study, Belalta et al. proposed a method for this task which heuristically solves a large-scale CP problem under the hood [12].

While heuristics and meta-heuristics are typically fast for small and mid-sized instances, their solutions have no guarantee of proximity to optimal solutions. Compared to them, there are fewer exact and approximation approaches proposed for solving the CP problem. In 2019, Simanchev et al. [13] introduced an exact branch-and-cut method tailored to two specific instances of the CP problem. They proposed a cutting-plane algorithm designed to explore facet inequalities, which was used to construct lower bounds, with special heuristics for searching for upper bounds. Their method is capable of finding optimal solutions for the two particular cases of the CP problem in a fair amount of time (within 3 hours) for networks up to 300 nodes. In 2023, Belyi et al. [14] estimated a tighter upper bound for the CP problem and combined it with a branch-and-bound algorithm to obtain exact solutions. Their method produced exact solutions faster than other existing methods at the time [14].

Some exact and approximation approaches rely on theoretical results from the polyhedral analysis of the IP formulations of the CP problem [15]. Letchford and Sørensen proposed a polynomial-time separation algorithm for finding the valid inequalities for CP [15]. In their more recent work, they proposed two families of new valid inequalities that can speed-up solving the IP models of CP [16]. Irmai et al. have recently generalized another family of valid inequalities for the CP problem [17]. For a detailed review of the literature on CP models and their performance, we refer the reader to [18]. A detailed dichotomy of clique-based partitioning problems is provided in [19].

Our contributions

In this study, we propose a method for approximating the optimal solutions of the CP problem. Our proposed approximation algorithm, named Troika, delivers solutions with guaranteed proximity to the optimal solution. Troika solves integer programs by node triple branching. This type of branching was proven useful [20] for dealing with constraints that involve triples. These constraints are called transitivity constraints and are a common challenge among Integer Programming (IP) models for clustering problems [5,14,21].

The main focus of this study is on developing an exact and approximation approach for solving the CP problem on small and mid-sized networks of practical relevance. Our proposed method pushes the practical limits on solving the CP problem on ordinary computers as demonstrated by comprehensive results on multiple benchmark datasets. While approximating an NP-hard optimization problem cannot possibly scale to large networks, the design of our method offers a key advantage over existing heuristics: it guarantees solution quality in the form of a user-specified optimality gap tolerance. In simpler terms, it allows the user to specify, in advance, their tolerance for the potential optimality gap of the partition. Compared to alternative exact methods, our method offers the advantage of having better time-quality speed-up: given the same time limit, it offers partitions closer to the optimal; and given the same optimality gap tolerance, it converges faster. We conduct extensive experiments to demonstrate the applicability of our proposed method on CP instances and two other use cases. As two side discussions, we analytically demonstrate the connections that convert a modularity maximization instance [22] and a correlation clustering instance [23] into a CP instance and show how Troika compares to eight modularity-based algorithms.

The technical background is outlined in Sect 2. The Troika algorithm is explained in Sect 3. Sect 4 provides comparative analysis results for Troika on five datasets demonstrating its practical advantages over the existing methods. Sect 5 discusses a use-case in community detection and demonstrates the advantages of Troika over eight modularity-based algorithms. Sect 6 deals with the applicability of Troika for portfolio analysis. The main results are discussed in Sect 7. Finally, materials and methods are provided in Sect 8.

2. Mathematical background

We represent the weighted graph G with node set V, edge set E, and weight matrix $\mathbf{W}$ as $G = (V, E, \mathbf{W})$. Graph G may have self-loops, but has at most one edge per each pair of nodes. Graph G has $|V| = n$ nodes and $|E| = m$ undirected weighted edges. Its symmetric weight matrix (weighted adjacency matrix) $\mathbf{W} = [w_{ij}]$ has real-valued entries $w_{ij} \in \mathbb{R}$. In some definitions of the CP problem, the graph G is restricted to be a complete graph [19] and therefore the clusters are actual cliques. We study the more general version of the CP problem where the input graph G is not necessarily complete $E = \{(i,j) \in V^2, i \leq j, w_{ij} \neq 0\}$. The non-zero entry $w_{ij} \in \mathbb{R}$ indicates the weight of the undirected edge $(i,j) \in E$ between node i and node j. Both positive and negative weights must exist in graph G for CP to be a non-trivial problem. The degree of node i is calculated by $d_i = \sum_j w_{ij}$.

The node set V of the input graph G can be partitioned into (any unspecified number k of) disjoint clusters based on partition $P = \{V_1, V_2, \ldots, V_k\}$ such that $\bigcup_1^k V_i = V$ and $V_i \cap V_j = \varnothing$. Given partition P, the relative cluster assignment of a pair of nodes (i,j) is same (represented by $x_{ij} = 0$) or different (represented by $x_{ij} = 1$). The partition P can therefore be alternatively represented as the symmetric binary matrix $\mathbf{X} = [x_{ij}]$ which is interpreted as follows: The binary entry x_{ij} indicates the relative cluster assignments of nodes i and j. The diagonal entries x_{ii} are 0's. Given partition $\mathbf{X}$, edges with endpoints in the same cluster (different clusters) are called internal (external) edges.

2.1. Problem statement

The clique partitioning problem [1] for the graph $G = (V, E, \mathbf{W})$ is defined below. Given graph G and partition $\mathbf{X}$, the *weight* (the sum of within-cluster edge weights) of the partition $W_{(G,\mathbf{X})}$ is computed according
to Eq (1).

$$W_{(G,\mathbf{X})} = \sum_{(i,j)\in E} w_{ij}(1 - x_{ij}) \tag{1}$$

In the CP problem, we look for an *optimal* partition: a partition $\mathbf{X}^*_{(G)}$ whose weight is maximum over all possible partitions: $\mathbf{X}^*_{(G)} = \arg\max_{\mathbf{X}} W_{(G,\mathbf{X})}$. Any partition of G that is not an optimal partition is a *sub-optimal* partition.

2.2. Integer programming formulations

The CP problem can be formulated as the IP model [1] in Eq (2).

$$\begin{aligned} \mathrm{CP}(G): \quad & \max_{x_{ij}} W = \sum_{(i,j)\in E} w_{ij}(1 - x_{ij}) \\ \text{s.t.} \quad & x_{ik} + x_{jk} \geq x_{ij} \quad \forall (i,j,k) \in T \\ & x_{jk} + x_{ij} \geq x_{ik} \quad \forall (i,j,k) \in T \\ & x_{ij} + x_{ik} \geq x_{jk} \quad \forall (i,j,k) \in T \\ & x_{ij} \in \{0,1\} \quad \forall (i,j) \in E \end{aligned} \tag{2}$$

In Eq (2), the optimal objective function value equals the optimal weight (maximum within-cluster weight) for the input graph G. An optimal partition is characterized by the optimal values of the x_{ij} variables. T indicates the set of all unique node triples $T = \{(i,j,k) \in V^3 | 1 \leq i < j < k \leq n\}$ for graph G. The 3 constraints defined for each triple in T are called *transitivity constraints*. They are a common challenge of network clustering problems formulated as integer programs [5,14,21].

The computational complexity of the classic formulation in Eq (2) becomes impracticable for large networks due to the voluminous number of $3|T|$ constraints, scaling as $3\binom{n}{3}$ which is $\mathcal{O}(n^3)$ [24]. The formulation in Eq (2) comprises numerous redundant constraints. The redundant constraints [24] are as follows:

$$\begin{aligned} x_{ik} + x_{jk} \geq x_{ij} \quad & \forall\, 1 \leq i < j < k \leq n,\ w_{ij} < 0 \wedge w_{jk} < 0 \\ x_{ik} + x_{jk} \geq x_{ij} \quad & \forall\, 1 \leq i < j < k \leq n,\ w_{ij} < 0 \wedge w_{ik} < 0 \\ x_{ij} + x_{ik} \geq x_{jk} \quad & \forall\, 1 \leq i < j < k \leq n,\ w_{jk} < 0 \wedge w_{ik} < 0 \end{aligned}$$

The classic formulation of the problem [1] in Eq (2), can be strengthened by using negative edges for removing the redundant constraints [24]. The redundancy of these constraints is due to the pressure from the maximization objective function. After removing the redundant constraints [24] and applying other simplifications [21], we arrive at the model $\mathrm{RP}^*(G)$ as in Eq (3).

$$\begin{aligned} \mathrm{RP}^*(G): \quad \max_{x_{ij}} W = & \sum_{(i,j)\in E} w_{ij}(1 - x_{ij}) \\ \text{s.t.} \quad x_{ik} + x_{jk} \geq x_{ij} \quad & \forall (i,j,k) \in T^k_+ \\ x_{jk} + x_{ij} \geq x_{ik} \quad & \forall (i,j,k) \in T^j_+ \\ x_{ij} + x_{ik} \geq x_{jk} \quad & \forall (i,j,k) \in T^i_+ \\ x_{ij} \in \{0,1\} \quad & \forall (i,j) \in E \end{aligned} \tag{3}$$

In the formulation $\mathrm{RP}^*(G)$ which is proposed by [21], the set T is replaced by the subsets T^k_+, T^j_+, T^i_+ which are defined as follows:

$$\begin{aligned} T^k_+ &= \{(i,j,k) \in T \mid w_{ik} > 0 \ \vee \ w_{jk} > 0\} \\ T^j_+ &= \{(i,j,k) \in T \mid w_{jk} > 0 \ \vee \ w_{ij} > 0\} \\ T^i_+ &= \{(i,j,k) \in T \mid w_{ij} > 0 \ \vee \ w_{ik} > 0\}. \end{aligned} \tag{4}$$

The optimal solution from $\mathrm{RP}^*(G)$ is required to go through a linear-time ($\mathcal{O}(m)$) post-processing step, called **pp** and described in [21], to ensure that an optimal solution has been obtained. This post-processing step ensures the solution found from solving the $\mathrm{RP}^*(G)$ formulation does not violate the transitivity constraints in the classic formulation in Eq (2). Miyauchi et al. [21] demonstrated that the $\mathrm{RP}^*(G)$ formulation combined with the **pp** post-processing step is a more efficient approach for solving the CP problem compared to solving the classic formulation [1] in Eq (2). In recent years, several attempts have been made for obtaining more efficient IP formulations for CP [4]. Koshimura et al. have proposed two new IP formulations with fewer constraints than $\mathrm{RP}^*(G)$. Like $\mathrm{RP}^*(G)$, both new models require a post-processing step to ensure that the optimal solution represents a feasible partition. Numerical results suggest that these two new formulations are only sometimes faster than RP^* [4]. Therefore, we use $\mathrm{RP}^*(G)$ as the base IP foundation on which we build the Troika algorithm.

Despite the efficiency gain in using $\mathrm{RP}^*(G)$, this strengthened formulation does not fully take advantage of the structural characteristics of the input graph. We address this shortcoming in formulation by using the graph structure for developing pre-processing steps (discussed in Sect 8).

3. The Troika algorithm

The development of Troika heavily relies on the lessons learned from the Bayan algorithm [20]. Bayan was developed for another network clustering problem that has the same challenge of transitivity constraints [22]. Inspired by the key components of the Bayan algorithm, like branching on node triples, Troika approximates the optimal solution of the CP problem. The two most important technical components of Troika are discussed in the following two Sects 3.1–3.2. Additional details about the Troika algorithm are provided in Sect 8 including a flowchart.

3.1. The branch and cut implementation

The feasible space of the $\mathrm{RP}^*(G)$ formulation in Eq (3) is defined by constraints on node triples. Consider $\mathcal{T}$ to be the union of the subsets T_+^k, T_+^j, T_+^i defined in Eq (4). $\mathcal{T}$ is the relevant set of node triples over which the transitivity constraints from [21] are defined. Given a node triple, $(i,j,k) \in \mathcal{T}$, the three transitivity constraints are equivalent to the logical disjunction of Eqs (5) and (6).

$$x_{ij} + x_{ik} + x_{jk} = 0 \tag{5}$$

$$x_{ij} + x_{ik} + x_{jk} \geq 2 \tag{6}$$

Unlike Bayan, Troika starts by obtaining one lower bound and one upper bound before forming any IP models. The upper bound is obtained using the method proposed by Belyi et al. in [14]. The lower bound is obtained using the CP version of the Combo algorithm [25] (from the Python library PyCombo [26]). Combo is a heuristic network optimization algorithm that can be reconfigured to solve the CP problem. Solving the natural LP relaxation (resulted from dropping the integrality constraints) of the IP model $\mathrm{RP}^*(G)$ [21] provides an additional upper bound. The minimum of the two upper bounds is used as the tight upper bound for starting the branch-and-cut scheme. Note that we have used the Gurobi LP solver [27] for solving all the LP models involved within the Troika algorithm.

At the root node, if the two bounds differ more than the optimality gap tolerance (as explained in Sect 3.2), the algorithm does not terminate. It selects a triple of nodes whose corresponding values (from the LP solution) violate both Eqs (5) and (6). Adding each of the violated constraints Eqs (5) and (6) forms a cut to the root node problem and divides the problem into left and right sub-problems. Recursively, for each of the two sub-problems, Gurobi LP and Combo are used to obtain an upper and a lower bound. This recursive process creates a search tree where node triples are used for branching.

Within Troika and after branching on the node triple (i,j,k), we use additional techniques to obtain the lower bound in the right and left branch respectively, to speed up the convergence. In the search for lower bound in the right branch, the pre-defined value δ is subtracted from the edge weights associated with nodes i, j, k. This adjustment can enhance the likelihood of identifying heuristic solutions that adhere to the constraint $x_{ij} + x_{ik} + x_{jk} \geq 2$ on the right branch. The δ value is set to be the absolute value of the median of all edge weights within the graph. This choice of value can ensure that the edge weights associated with nodes

i, j, k get small enough to deter the Combo algorithm from grouping those triples together. This alteration does not ensure the compliance of the heuristic solution with the constraint in Eq (6); however, such compliance is not a prerequisite for Troika's convergence to optimality.

Conversely, the constraint $x_{ij} + x_{ik} + x_{jk} = 0$ is added to the left sub-problem in the branching process. The associated nodes i, j, k are grouped together and represented by the supernode ijk. This supernode gets connected to all neighbours of i, j, k. The edges between the three nodes are conserved as a weighted self-loop on the supernode ijk. This ensures the Combo algorithm groups the node triple together in the returned partition, adhering to the constraint in the left sub-problem. The branching process is a key component among several other components of the Troika algorithm which are discussed in Sect 8 where a schematic representation of the Troika algorithm is also provided as a flowchart.

Exploration of search tree continues through branching on new triples whose LP solution violates both Eqs (5) and (6). After completing the computations of all Branch and Bound (B&B) nodes at a given level of the search tree, Troika determines the *incumbent* and the *best bound*. The incumbent is chosen as the higher value between the best heuristic solution and the best integer solution discovered during the search. The highest upper bound value from the level is recorded as the best bound. During the branching process, a B&B node is considered *fathomed* under three circumstances: when the LP solution turns integral; when the LP becomes infeasible; or when the LP objective function value falls below the current incumbent. Under these conditions, further branching from the B&B node is halted, and it is subsequently closed.

3.2. Search termination criteria

Troika is designed with two search termination criteria that grant users the flexibility to choose between computational efficiency and the precision of solutions. These criteria include *optimality gap tolerance* and *solve time limit*.

During the search process, Troika aims for the convergence of the best bound and incumbent to identify globally optimal solutions. This convergence is indicative of the exactness of the solution as per the branch-and-cut method, demonstrating the algorithm's capability to deliver globally optimal results under a stringent criterion for the optimality gap tolerance.

Alternatively, one can use a larger optimality gap tolerance to obtain an approximate solution efficiently. The *optimality gap*, denoted by g, is the percentage difference between the current incumbent, i, and the best bound, b, according to the equation $g = (b - i)/b$. The optimality gap tolerance is a user-specified threshold for the acceptably low optimality gap to terminate the search. Using an optimality gap tolerance of $0 < 1 - \alpha < 1$ makes Troika an α-*approximation* algorithm for CP and terminates it once the solution gets within the $1 - \alpha$ proximity of the optimal value.

Furthermore, the criterion of solve time limit defines a maximum duration for the search process. This criterion is particularly suitable for scenarios where timely results are desirable. Additional technical details about the Troika algorithm are provided in Sect 8.

4. Results on solution quality and time

In this section, we compare three methods: (1) Troika, (2) the Combo heuristic algorithm for CP [25], and (3) the $\mathrm{RP}^*(G)$ formulation [21], solved using the commercial IP solver Gurobi [27] followed by the **pp** post-processing step. As a shorthand, we refer to the latter method as Gurobi IP. Gurobi is considered to be among the fastest mathematical solvers for solving IP problems [28], setting a challenging baseline for Troika to be compared against.

To ensure a thorough and unbiased evaluation, we use a diverse set of five datasets and use the average and standard deviation of three runs for each method and instance. Our five comparative analyses in Sects 4.1–4.5 show the performance of each of the three methods for solving the CP problem in terms of solve time and solution quality. The primary metrics for assessing performance include (1) *solve time* - the time taken by each method to produce its final partition on each instance and (2) the *Extent of Sub-optimality* (EOS) for the partition produced by each method for each instance. We define and use the EOS for method/algorithm A on graph G as $EOS_{(G,A)} = 1 - O_{(G,\mathbf{X}_A)}/O^*_G$. In this equation, $O_{(G,\mathbf{X}_A)}$ is the objective function value corresponding to the partition $\mathbf{X}_A$ returned by method A for graph G. O^*_G denotes the globally maximum objective function value for graph G.

A recent study by Sørensen and Letchford [29] has consolidated known and new challenging instances of the CP problem, addressing the difficulty of locating benchmark instances scattered across literature. They classify the CP instances into 3 distinct classes: "easy", "non-trivial" and "challenging" [29]. For "easy" instances, their standard LP relaxation yields at least one optimal solution that is integral. Such instances were solved at the root node by the branch-and-cut implementation in [29], either due to the LP solution being an integer solution or because their heuristic's lower bound matches the LP's upper bound. Instances that do not qualify as easy but are solvable by the algorithm in [29] within an hour are considered "non-trivial". Lastly, "challenging" instances are defined as those that cannot be solved by the algorithm in [29] within an hour time, making them particularly complex. Sørensen and Letchford [29] have shared their CP instances and the optimal solutions that were available in a public GitHub repository https://github.com/MMSorensen/CP-Lib.

In Sects 4.1–4.4, we use instances from four datasets of [29] for which the optimal solutions are known. In Sect 4.5, we generate a dataset of homogeneous synthetic networks and obtain its optimal partitions. The synthetic networks allow us to compare the three methods under varying time restrictions.

All computational experiments were conducted using Python 3.11 on a MacBook computer equipped with an Apple M1 Pro and 16 GB of RAM, operating under macOS 15. All experiments in Sects 4.1–4.4 are run with a time limit of 10 minutes per instance. Unlike Combo, Troika and Gurobi IP take optimality gap tolerance and time limit as optional user inputs. All experiments of Sect 4 are run with an optimality gap tolerance of 0.05 used for Troika and Gurobi IP to put them on an equal footing that is also reasonable for comparison with Combo. We have empirically observed that using the *start separate* flag in Combo is crucial for obtaining high quality partitions and therefore configured Combo with it for all experiments in Sect 4.

4.1. Comparisons on ABR benchmarks

Some early works on the CP problem were based on modeling and solving CP instances in the context of qualitative data analysis [1,3]. This context of the CP problem is known as "Aggregation of Binary Relations into an equivalence relation" (*ABR*). One instance of the problem is denoted through z "objects", each possessing q qualitative "attributes". These objects, along with their attribute values, can be organized into a matrix, with each element representing the value of attribute v for object i [29]. For every pair of objects and for each attribute v, the following binary constant is defined:

$$r^v_{ij} = \begin{cases} 1, & \text{if attribute } v \text{ has the same value for objects i and j} \\ 0, & \text{otherwise.} \end{cases}$$

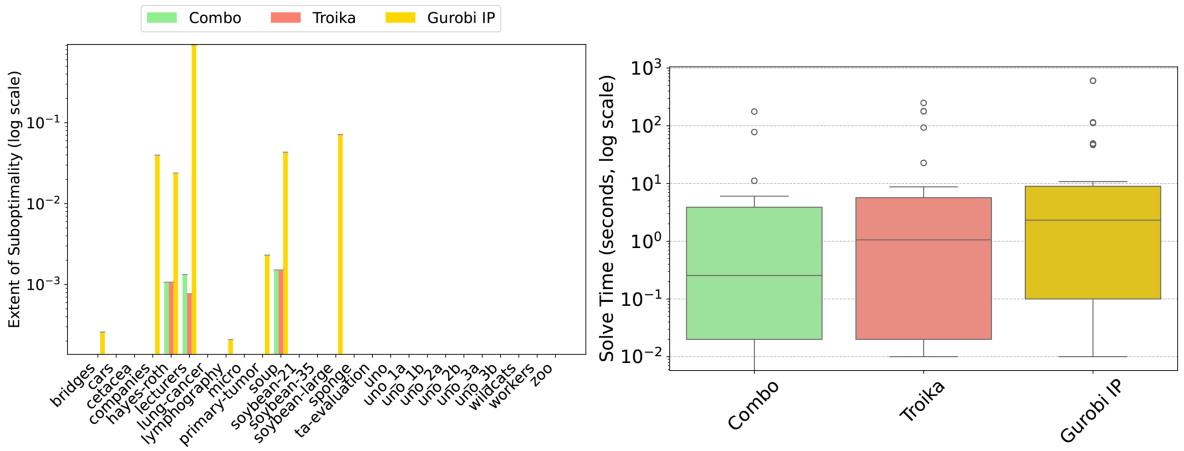
The similarities between objects i and j regarding the q attributes are then quantified through the edge weight $w_{ij} = 2\sum_{v=1}^{q} r_{ij}^{v} - q$ [1]. In the case of missing r_{ij}^{v} values, some adjustments (using the approach from [30]) were made to create the weighted graphs as discussed in [29].

The 26 ABR test instances that we use from [29] consist of real-life use cases of the ABR from the literature [30–32]. They feature node counts n ranging between 30 and 797, with edge counts m ranging from 381 to 306,915. Most instances are deemed as "easy" and with four classified as "non-trivial" and one as "challenging" [29]. Next, we present the performance results for each of the three methods on the 26 ABR instances.

Fig 1 provides a comparison between the three methods based on EOS and solve time. Fig 1(a) shows that most ABR instances are solved to global optimality by these methods because EOS is zero for them. For the few instances where EOS is not zero, the difference in solution quality between Troika and Gurobi IP is substantial. Fig 1(b) indicates that Combo expectedly has the lowest median solve time on the ABR dataset followed by Troika and Gurobi IP respectively. Troika and Combo arrive at optimal solutions in 23 out of 26 ABR instances. Gurobi IP reached the globally optimal solutions in only 18 instances though.

More detailed results on objective values and solve times are provided in S1 Table (in the supporting information). An important distinction is observed in the "lecturers" instance, where Gurobi IP reaches the time limit and returns a substantially inferior partition compared to Troika and Combo. Additional experiments showed that Gurobi IP fails to converge for the "lecturers" instance within a four-hour limit. As shown in Fig 1(a), the optimal partition of this instance is unattainable by Troika and Combo as well, while they produce partitions with orders-of-magnitude lower EOS compared to Gurobi IP. Besides Troika producing higher quality solutions, the time columns of S1 Table (in the supporting information) demonstrate that Troika is faster than Gurobi IP in 23 out of 26 ABR instances.

The quality of partitions produced by Combo and Troika is comparable for the ABR instances. Combo is particularly faster than Troika for the "hayes-roth", "lecturers", and "soup" instances because the extra computations of Troika (to ensure the optimality gap tolerance is met) take substantial time. Among these 26 instances, the mean solves times for Troika,



(a) EOS on ABR benchmarks **(b)** Solve time on ABR benchmarks

Fig 1. Two comparative performance measures for the three methods Troika, Combo, and Gurobi IP on the ABR benchmark dataset: (a) Extent of sub-optimality, (b) solve time.

https://doi.org/10.1371/journal.pcsy.0000062.g001

Gurobi IP, and Combo are 22.53, 56.59, and 11.37 seconds, respectively, illustrating that Troika, on average, executes over 2.51 times faster than Gurobi IP and around 1.98 times slower than Combo, on average.

4.2. Comparisons on equicut benchmarks

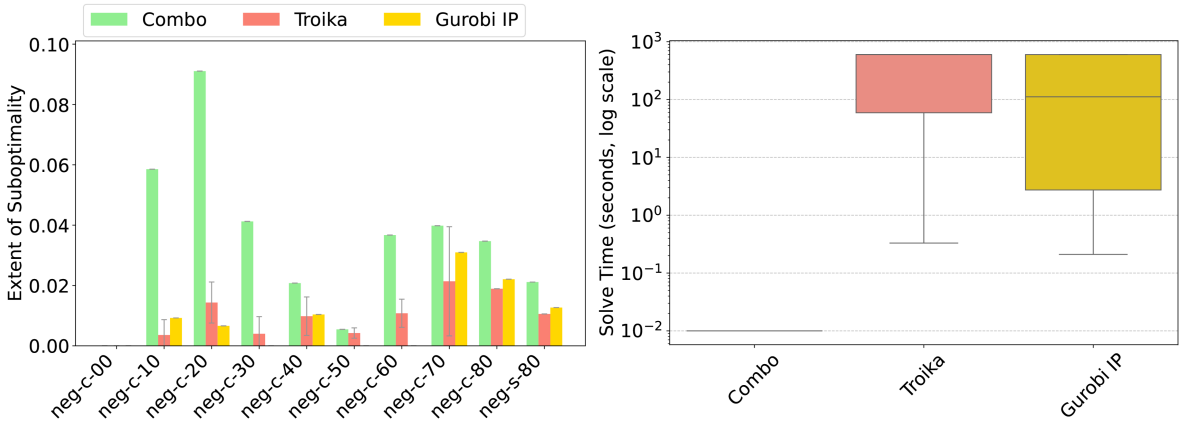
Some benchmark instances of [29] are derived from the *equicut* problem. The "equicut" or "equipartition" problem is similar to the CP problem, but has the additional constraint that the partition must be a partition of two clusters of equal or almost equal size. Sørensen and Letchford used the instances from the equicut literature that had negative edges and defined CP instances based on them by removing the cluster count and cluster size restrictions [29].

We solve ten challenging and non-trivial equicut benchmark instances using the three methods. They have 50 nodes, except the last instance which has 60 nodes. Fig 2 shows EOS and solve time for each method on the ten equicut benchmark instances. Error bars in Fig 2(a) show one standard deviation for EOS values obtained over three runs for each algorithm. Fig 2(a) shows that the partitions of Combo for nine out of ten equicut instances get improved by the extra work that Troika does. On five instances, Troika has a better (lower) average EOS compared to Gurobi IP. Fig 2(b) shows that satisfying the optimality gap tolerance of 0.05 on these instances requires extra work from Troika or Gurobi IP that is substantially more time-consuming than obtaining a single heuristic solution from Combo.

4.3. Comparisons on correlation benchmarks

A set of new benchmark instances proposed in [29] are called *correlation* benchmarks. These benchmarks are produced based on two steps: (1) creating a matrix with uniformly random entries from the unit interval, (2) defining the weighted edge (i,j) to be the correlation coefficient between columns i and j of the matrix.

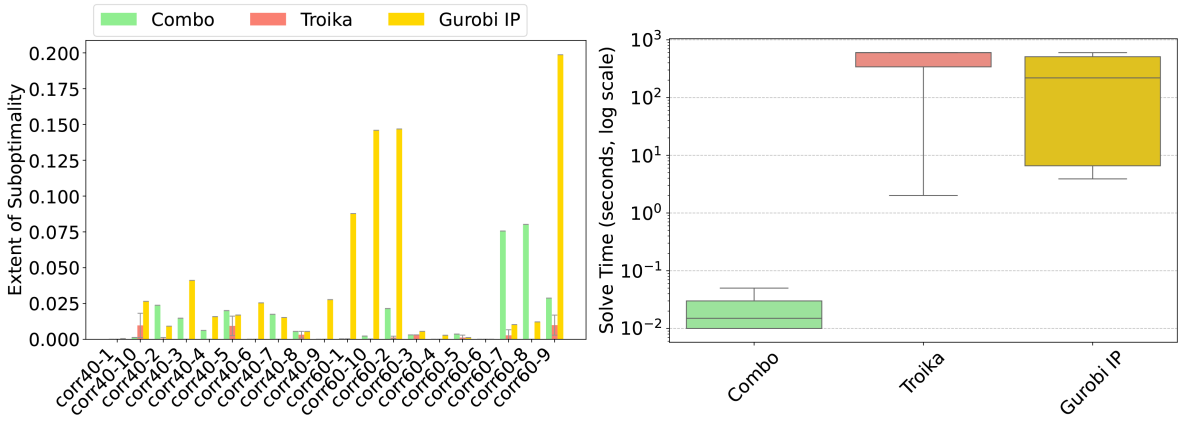
We solve 20 challenging and non-trivial correlation instances using the three methods. The instance name denotes the number of nodes. Fig 3(a) shows that the partitions from Troika have much lower EOS compared to the two other methods on almost all these 20 instances. Fig 3(b) shows Troika median solve time to be higher than that of Gurobi IP on



(a) EOS on equicut benchmarks **(b)** Solve time on equicut benchmarks

Fig 2. Two comparative performance measures for the three methods Troika, Combo, and Gurobi IP on the equicut benchmark dataset: (a) Extent of sub-optimality, (b) solve time.

https://doi.org/10.1371/journal.pcsy.0000062.g002



(a) EOS on correlation benchmarks **(b)** Solve time on correlation benchmarks

Fig 3. Two comparative performance measures for the three methods Troika, Combo, and Gurobi IP on the correlation benchmark dataset: (a) Extent of sub-optimality, (b) solve time.

https://doi.org/10.1371/journal.pcsy.0000062.g003

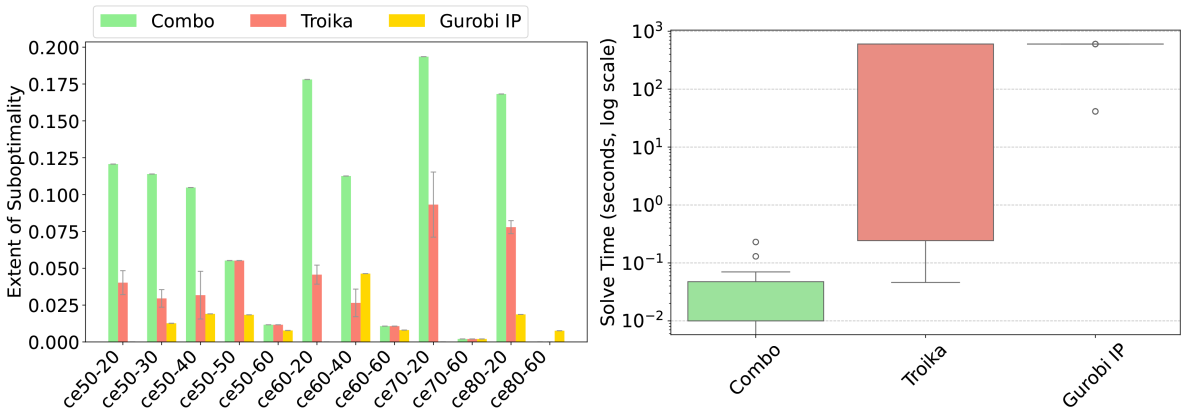
these instances; this is justifiable by the higher quality solutions that Troika produces compared to Gurobi IP parametrized with the same time limit and optimality gap tolerance.

4.4. Comparisons on clusedit benchmarks

Another set of CP benchmarks from [29] are defined based on the "cluster editing" (*clusedit*) problem. This problem, also known as the "correlation clustering" problem [33], is defined on a signed graph $G = (V, E^-, E^+)$. The edges in the set E^- all have the weight of -1 (are negative edges) and the edges in the set E^+ all have weight of +1 (are positive edges). The correlation clustering problem is the task of finding a partition of nodes into any number of clusters to minimize the total count of intra-cluster negative edges and inter-cluster positive edges. To convert instances of this problem to CP instances, Sørensen and Letchford have defined the task of maximizing the sum of within-cluster edge weights for clusedit instances that have 20% to 60% negative edges.

We solve 12 challenging and non-trivial instances of these clusedit benchmarks using the three methods. The instance name denotes the number of nodes followed by the fraction of negative edges. Fig 4(a) demonstrates that Troika substantially improves the partitions from Combo, but rarely have better EOS compared to the partitions from Gurobi IP. Fig 4(b) shows Troika to have some relative advantages in terms of solve time compared to Gurobi IP on the clusedit benchmarks.

The results provided in Figs 2–4 show that for non-trivial and challenging instances across four benchmark datasets, Troika consistently returns partitions with objective function values that are closer to the globally optimal solutions compared to the Combo heuristic. This result is expected given the design of the Troika algorithm which relies on Combo for lower bounds and does some extra work for improving those partitions. Note that the descriptive comparisons provided in Figs 2–4 are not all statistically significant because some performance differences between these algorithms are marginal as shown in the error bars in Figs 2–4. While running Combo is considerably faster than Troika, the extra time that Troika spends leads to obtaining higher quality partitions as illustrated in Figs 2–4. While the results showed that generally Troika has some advantage in solution quality and/or time compared to Gurobi IP; there are some instances on which Gurobi IP outperforms Troika.



(a) EOS on clusedit benchmarks **(b)** Solve time on clusedit benchmarks

Fig 4. Two comparative performance measures for the three methods Troika, Combo, and Gurobi IP on the clusedit benchmark dataset: (a) Extent of sub-optimality, (b) solve time.

https://doi.org/10.1371/journal.pcsy.0000062.g004

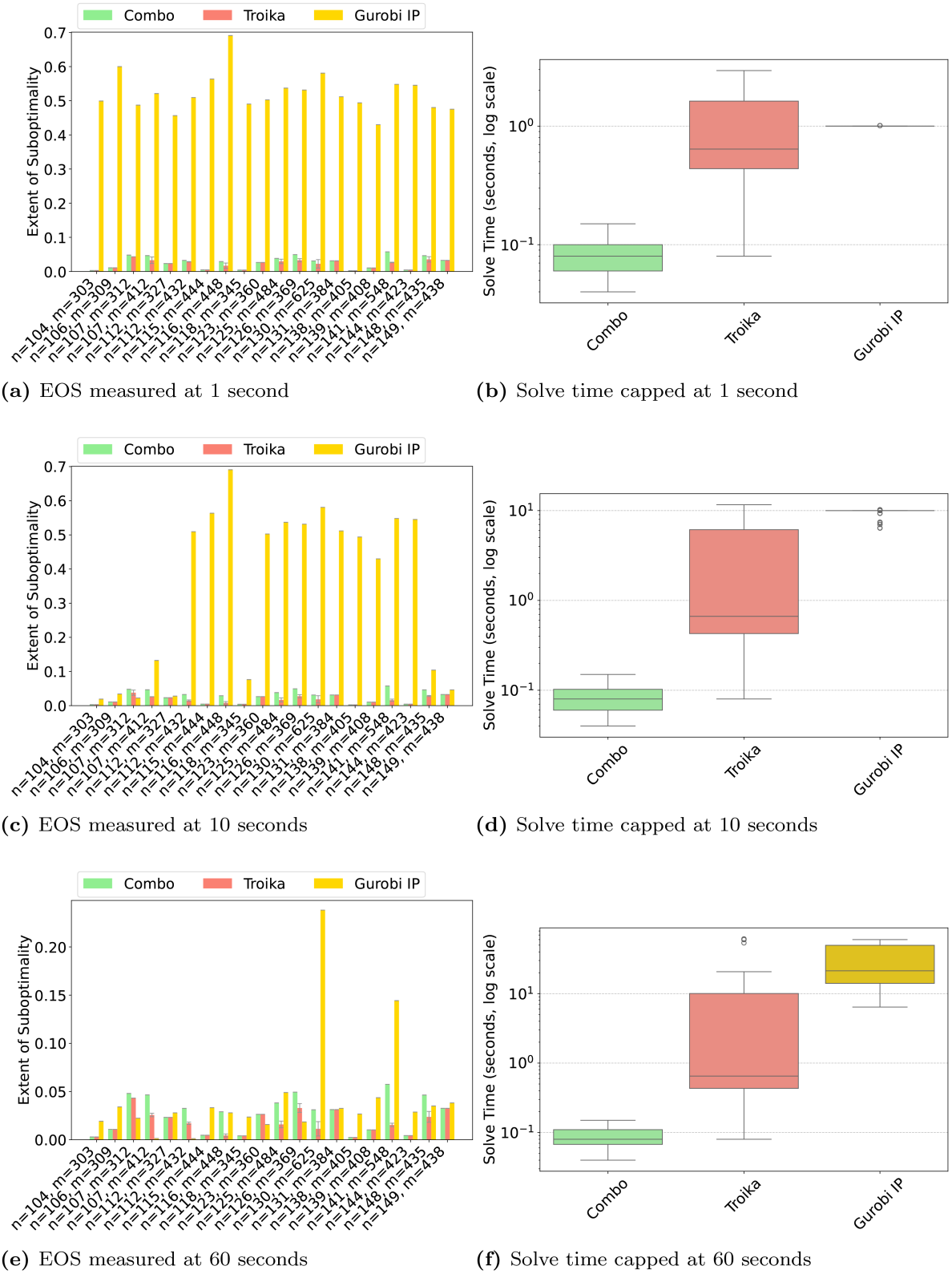
4.5. Comparisons on Barabási Albert graphs

Besides instances from four datasets of [29], we create random weighted graphs based on the Barabási-Albert (BA) graph generation process [34]. We use them for evaluating Troika on graphs that mirror real network structures. The Barabási Albert process grows the network through preferential attachment [34], which mirrors the heterogeneous connectivity inherent in some real-world networks, such as social, technological, and biological systems [35]. The BA graphs are reflective of certain real-world contexts where preferential attachment is a reasonably realistic model despite its simplicity.

We generate 20 BA graphs to test all the three methods under varying time restrictions. These graphs are generated with the number of nodes n sampled from the discrete uniform distribution of [100,150]. For the number of edges to attach between a new node and existing nodes, the discrete uniform distribution of [3,6] was used. Finally, each edge was assigned a random weight from the discrete uniform distribution of [−10,10]. The data for these 20 BA networks are available in a FigShare repository [36]. Before running our comparison between the three methods, we obtain the globally optimal partitions of these instances by solving the RP^* formulation using the Gurobi solver parametrized with an optimality gap tolerance of $1e{-}5$. Then, we use the optimal values for calculating EOS for the three methods. Unlike Sects 4.1–4.4, we define three experiment settings for assessing the three methods on these BA instances. The three experiment settings correspond to the time limits of 1, 10, and 60 seconds consistently used for the methods.

In Fig 5, we observe that Troika consistently improves solutions from Combo on these 20 BA instances. We also see that this improvement increases as the time limit increases from 1 to 10 and then to 60 seconds. On a few instances, Troika's partition is the same as Combo's partition; this is because the optimality gap of Combo's partition is below 0.05 and therefore Troika terminates without further attempting to improve the partition. Note that the top whiskers for the boxplots of Troika show that its solve time exceeds the time limit in some cases. This marginal breach of the time limit is due to the graceful termination of Troika which is only possible after each branching step is complete.

Across the three time limit settings, Troika's EOS is lower than Combo's in around half of the BA instances. This indicates that even a split second of additional time gives Troika the



(a) EOS measured at 1 second

(b) Solve time capped at 1 second

(c) EOS measured at 10 seconds

(d) Solve time capped at 10 seconds

(e) EOS measured at 60 seconds

(f) Solve time capped at 60 seconds

Fig 5. Extent of sub-optimality and solve time for the three methods Troika, Combo, and Gurobi IP on the BA instances.

https://doi.org/10.1371/journal.pcsy.0000062.g005

potential to produce a better solution than Combo. Fig 5 shows that the median solve time of Troika is lower than that of Gurobi IP across the three time limit settings. Taken together with the consistently lower EOS values of Troika in Fig 5, we see that Troika outperforms Gurobi IP in both time and solution quality on almost all BA instances. Note that for very few BA instances, Gurobi IP terminates faster than Troika, but typically with a partition that has higher EOS.

The comprehensive results provided in Figs 1–5 show the practical advantages of Troika over two existing alternatives for obtaining approximate solutions for the CP problem. In Sect 5, we investigate the applicability of Troika for another use case: community detection.

5. Applicability of Troika in community detection

Descriptive community detection (CD) is the data-driven task of clustering nodes of an input (unsigned) graph into groups (communities) [37,38]. Among a wide range of methods for community detection [38], optimization-based algorithms are common approaches for CD [39]. They aim to optimize a network-level objective function, such as modularity [40], across all possible partitions of the input graph. We use the modularity objective function as an example to make an explicit connection between the CP problem and optimization-based community detection.

5.1. The modularity maximization problem

Given the NP-hard nature [22] of the modularity maximization problem, most modularity-based community detection algorithms are heuristics. We discuss solving the CP problem by the Troika algorithm as an indirect method for community detection through approximating maximum modularity.

The modularity function is directly extendable to graphs that have nonnegative edge weights. Therefore, we define the Modularity Maximization (MM) problem [5,22] for the simple (unsigned but generally weighted) graph $G = (V, E, \mathbf{W})$ whose edge weights are nonnegative ($w_{ij} \geq 0$). The modularity matrix of graph G is represented by $\mathbf{B} = [b_{ij}]$ whose entries are $b_{ij} = w_{ij} - \gamma d_i d_j / \sum_{(i,j)\in V^2} w_{ij}$. The resolution parameter for the modularity function [41] is denoted by γ. Without loss of generality, we use $\gamma = 1$.

Given partition $\mathbf{X}$ (defined earlier in Sect 2), for a pair of nodes (i,j), their cluster assignment is same (represented by $x_{ij} = 0$) or different (represented by $x_{ij} = 1$). Given weighted graph G and partition $\mathbf{X}$, the *modularity* of the partition $Q_{(G,\mathbf{X})}$ is computed according to Eq (7).

$$Q_{(G,\mathbf{X})} = \frac{1}{\sum_{(i,j)\in V^2} w_{ij}} \sum_{(i,j)\in V^2} b_{ij}(1 - x_{ij}) \tag{7}$$

In the modularity maximization problem, we look for a partition $\mathbf{X}^*$ whose modularity is maximum over all possible partitions: $\mathbf{X}^*_{(G)} = \arg\max_{\mathbf{X}} Q_{(G,\mathbf{X})}$

5.2. Converting an MM instance into a CP instance

The feasibility space of MM and CP problems is the same: all possible partitions of the input graph nodes into non-overlapping clusters. In CP, only the edges ($w_{ij} \neq 0$) contribute to the objective function if they become internal edges ($x_{ij} = 0$). In MM, every pair of nodes (including pair of nodes without an edge) that have $b_{ij} \neq 0$ contributes to the objective function if they are assigned to the same cluster ($x_{ij} = 0$). Therefore, as long as the input graph is not a complete graph, an IP instance of MM on the graph $G = (V, E, \mathbf{W})$ has more decision variables

compared to a CP instance of the same graph. MM requires a decision variable for node pairs with $b_{ij} \neq 0$ compared to CP which requires a variable for node pairs with $w_{ij} \neq 0$. Therefore, it is expected that solving an instance of MM on the non-complete graph G will be harder compared to solving an instance of the CP problem on a non-complete graph with the same number of nodes.

Solving the MM problem on graph $G = (V, E, \mathbf{W})$ that has a modularity matrix $\mathbf{B}$ is equivalent to solving the CP problem for graph G' whose weight matrix equals $\mathbf{B}$. Therefore, an instance of the MM problem on graph G can be converted into an instance of the CP problem for graph G' through using the weight matrix $\mathbf{B} = [b_{ij}]$ based on the formula $b_{ij} = w_{ij} - d_i d_j / \sum_{(i,j)\in V^2} w_{ij}$. This implies that algorithms for solving the CP problem (including Troika) are also capable of solving the modularity maximization problem. Beside modularity maximization, the Troika algorithm also solves two other fundamental graph clustering problem as discussed in Sect 8.5.

5.3. Baselines, data, and measures

We compare Troika with eight modularity maximization algorithms (baselines). These eight algorithms are (1) Clauset-Newman-Moore (CNM) [42], (2) Louvain [43], (3) Belief [44], (4) Paris [45], (5) Leiden [46], (6) EdMot [47], (7) Combo for MM [25], and (8) graph neural network (GNN) [48]. For algorithms (1-6), we use their Python implementation from the Community Discovery library (*CDlib*) version 0.2.6 [49]. To access (7) Combo for MM, we use the Python library PyCombo [26]. For the GNN algorithm, we use its Python implementation (the GNN-100 variation) from its public GitHub repository referenced in [48].

For this comparison, we consider 100 networks comprising 53 real networks from a wide range of contexts, and 47 structurally diverse synthetic networks. The data for all these 100 networks are available in a FigShare repository [36]. The 47 synthetic networks are produced based on the graph generation models known as Lancichinetti-Fortunato-Radicchi (LFR) [50] and the Artificial Benchmark for Community Detection (ABCD) [51]. They include 20 LFR networks with small mixing parameter values $\mu \in \{0.01, 0.1\}$ and 27 ABCD networks with small mixing parameter values $\xi \in \{0.01, 0.1, 0.3\}$. The choice of using small mixing parameters ensures that these 47 synthetic networks have modular structures.

Our extent of sub-optimality measure, $EOS_{(G,A)} = 1 - O_{(G,\mathbf{X}_A)}/O^*_G$, is applicable in the context of modularity maximization as well. We use it to compare Troika to the eight MM algorithms based on solution quality. In cases where an algorithm returns a partition with a non-positive modularity value, we set $EOS = 1$ to facilitate easier interpretation of proximity to optimality based on non-negative EOS values.

5.4. Results on Troika for modularity maximization

Fig 6 shows the EOS for the partitions produced by nine algorithms including Troika, all of which aiming to maximize modularity. Among the nine algorithms, four algorithms have the best median performance indicated by a median EOS of zero: Troika, Combo (for MM), Leiden, and GNN. Among these algorithms, Troika has the lowest mean EOS. On these 100 networks, Troika returns globally optimal solutions for 88 networks. The median and mean of EOS for all algorithms are provided in S2 Table (in the supporting information). The results in Fig 6 demonstrates that Troika can be reliably used to partition unsigned networks by approximating maximum modularity. In Sect 6, we move on to demonstrating a different use case of Troika: portfolio analysis.

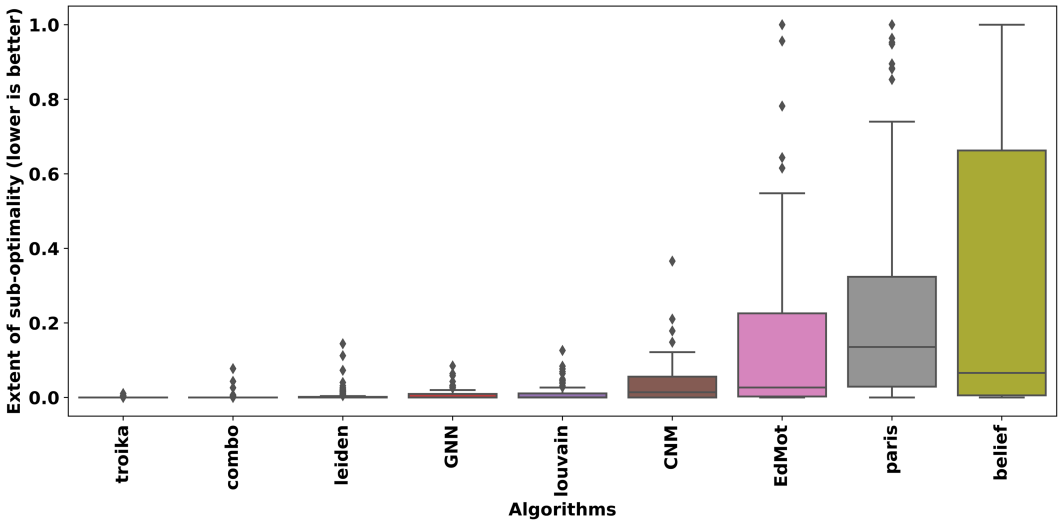


Fig 6. The boxplot of each algorithm illustrates the extent of sub-optimality for their partitions on the 100 real and synthetic networks.

https://doi.org/10.1371/journal.pcsy.0000062.g006

6. Applicability of Troika in portfolio analysis

In portfolio analysis and management, gaining insights from the correlations between financial assets is paramount. Portfolio analysis is challenging due to the dynamic and networked nature of financial portfolios, where asset correlations play a crucial role in diversification-based risk reduction (e.g. hedging) and maximizing returns. In this section, we focus on using Troika's solutions for the CP problem to provide temporal and portfolio-level interpretations of stock return correlations within a well-known market index from 2000 to 2024. The Standard and Poor's 500 market index (S&P 500 for short) tracks the stock performance of 500 largest companies in the US stock exchanges.

6.1. Creating weighted networks from correlations of returns

For each year between 2000 and 2024, we create a weighted network from the correlation matrix of the constituents (individual stocks within the S&P 500). Almost all pairs of stocks have a non-zero correlation coefficient. Therefore, using all correlation coefficients as edge weights produces a complete graph without any particular structure. We create edges from correlations that are statistically significantly high (in absolute terms) using critical points of a normal distribution -2 and +2 (for the conventional significance threshold of 0.05). However, correlations data are often not normally distributed. Therefore, a transformation is required before using standard normal theory [52]. The *Fisher transformation* of correlation coefficients yields a variable that is approximately normally distributed [52]. The Fisher transformation takes the correlation coefficient r_{ij} corresponding to the stock i and j and returns $z_{ij} = 0.5 \ln(\frac{1+r}{1-r})$. After the transformation, the distribution of transformed values z_{ij} for each year can be used to decide which pairs have strong positive or strong negative correlations. We create an edge (i,j) with the weight of r_{ij} for each transformed variable z_{ij} that differs from the mean z value by more than 2 standard deviations. The data for these 25 financial networks are available in a FigShare repository [36].

6.2. Troika's results on portfolio networks

In these networks, individual stocks are represented as nodes, and strong positive or strong negative correlations between them as weighted edges. We run Troika on these networks and obtain their globally optimal partitions in 22.14 ± 20.88 seconds per network. Using the optimal partitions returned by Troika on each network, we evaluate the changes in the clusters inferred from correlations of returns within the S&P 500 index over 24 years.

The analysis of the optimal partitions produced by Troika on these networks reveals the dynamics in the structure of a major part of the US stock market over time. S3 Table (in the supporting information) provides detailed results on the cluster size and the number of clusters for the partitions of the S&P 500 networks obtained by the Troika algorithm. The optimal partition for each network is visualized in Fig 7 where clusters are shown using different node colours. The optimal partition for the year 2000 has 11 clusters (of positively correlated stocks) and an average cluster size of 13.54 nodes per cluster. In the three years leading to the 2008 financial crisis, we observe the number of clusters monotonically decreasing from 12 to 1 while the average cluster size monotonically increases from 16.3 to 294. In the network for year 2008, all edge weights are positive (from the consistently negative returns for most stocks). This causes the optimal partition to become the trivial solution in which all nodes belong to a single cluster. Such an unusual optimal partition structure is only observed for the year 2008 when the most severe economic crisis of the contemporary era impacted the US stock market. The results in Fig 7 show that Troika can handle these portfolio networks and provide optimal solutions that reveal patterns from financial correlation data modeled as networks.

Moving on to the more recent time, the optimal partitions also show a structural shift during the COVID-19 pandemic. In the year 2019 (and before the pandemic started), the network had 348 nodes and was predominantly characterized by one major cluster comprising 283 stocks and the rest scattered among 10 substantially small clusters. The optimal partition shows that the returns from the 283 stocks were overall positively correlated while there were 65 stocks in the network whose returns had different patterns and therefore created 10 separate clusters. The structure of the optimal partition goes through a major change in 2020, where a single dominant cluster emerges, consisting of over 97% of the stocks (298 out of the total 306 stocks), a change possibly attributable to the market's reaction to the global pandemic conditions at the time. The year 2021 saw a reversion to a structure somewhat reminiscent of the 2019 structure, showcasing 13 clusters with one predominant cluster housing 292 constituents. By 2023, the optimal partition had changed into five clusters, with two primary clusters comprising 189 and 231 constituents, respectively. Except for the years 2008 and 2020, multiple major clusters exist in all optimal partitions distinguishing years of financial downturn (visible with distinct node colours in Fig 7).

In summary, the optimal partitions obtained by Troika allow us to infer clusters from correlations which in turn highlight the temporal changes of a portfolio or market index over the years. Compared to the network of a portfolio with one cluster encompassing all the network (the year 2008) or almost all the network (the year 2020), the observed presence of multiple sizable negatively correlated clusters is interpretable as a balanced portfolio with diverse sectors which offer opportunities for hedging against sector-specific risks.

7. Discussion and conclusion

We proposed an approximation algorithm for the CP problem and demonstrated its performance and applicability. The comparative analysis on five datasets provided in Sect 4 indicates the practical advantages of Troika in solving the CP problem, in comparison to two existing

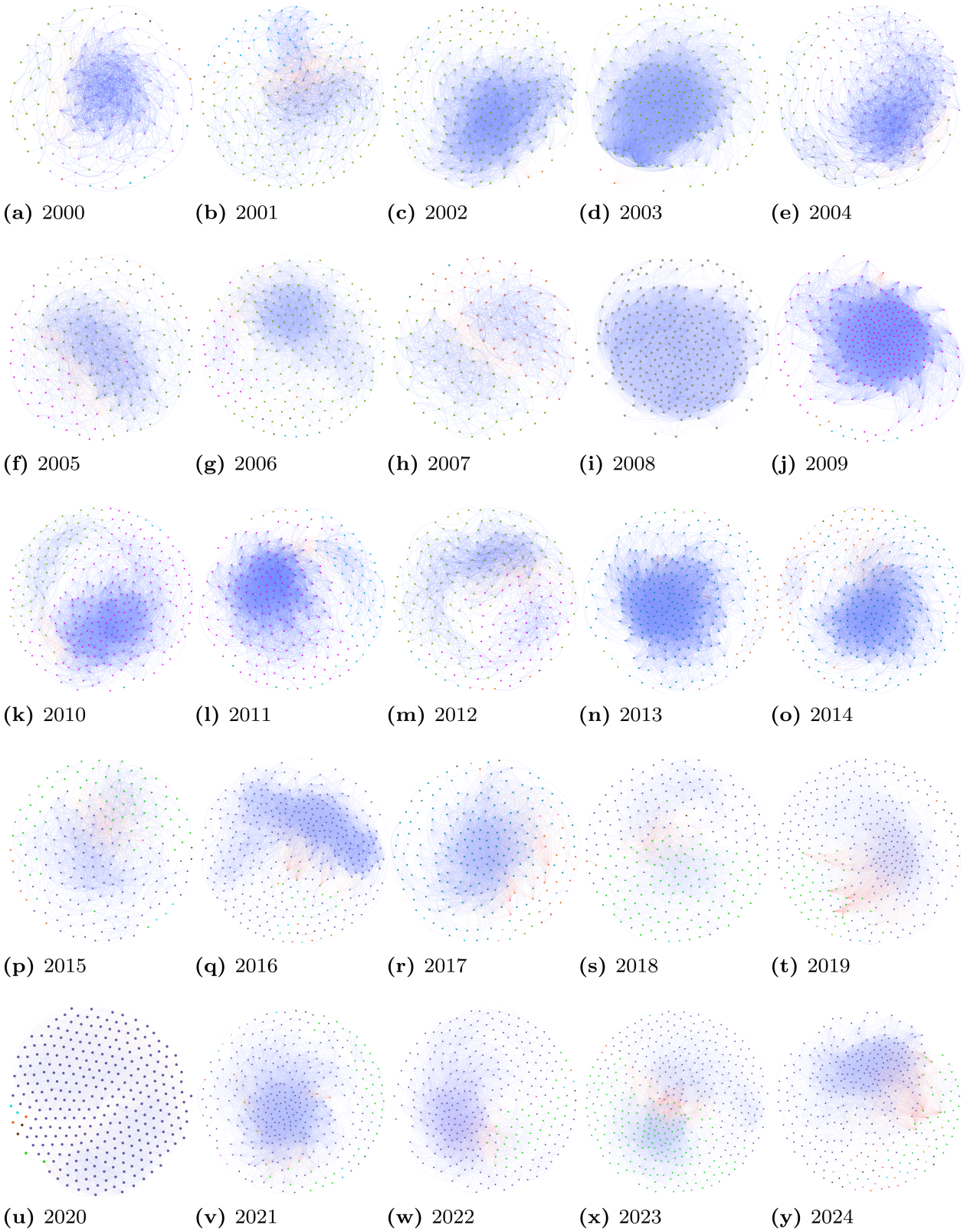


Fig 7. The respective partitions returned by Troika for the networks of *S&P 500* market index from 2000 to 2024. Red and blue edge colours represent negative and positive correlations respectively. Different node colours represent the major clusters. Magnify the high-resolution figure on screen for the details.

https://doi.org/10.1371/journal.pcsy.0000062.g007

alternatives. Troika improves upon partitions from the Combo algorithm and returns solutions that are substantially closer to optimal. Note that there are some CP benchmark instances that are randomly generated and therefore are less particular in terms of network structure. The *Random* dataset from [29] includes such instances which are used in multiple studies [18,53]. On eight challenging and non-trivial instances from the Random dataset [29], we observed that Troika outperforms Combo, but shows a limitation for achieving any time-quality tradeoff advantage over Gurobi IP on these random unstructured instances. However, Troika outperforms the state-of-the-art Gurobi IP solver in solution quality and/or solve time on most CP instances as shown in Figs 1–5.

Unlike common heuristic algorithms that rely on local or greedy optimization approaches, Troika is an approximation optimization algorithm for the CP problem and returns partitions with a guaranteed proximity to global optimality. Note that the descriptive comparisons we provided are not all statistically significant because some performance differences between these methods are marginal. A future study can be aimed to rank alternative CP methods. Such a study can use a Friedman test [54] followed by a post-hoc Li test [55] to determine the statistically significant performance differences among existing CP methods.

Troika handles networks with up to 5000 edges providing close-to-optimal solutions within a reasonable amount of time on standard hardware. For large-scale challenging instances (which are not solvable within an hour according to [29]), Troika returns high quality solutions within 10 minutes. On a wide range of benchmark instances, Gurobi IP returns solutions of lower quality if operationalized with the same time limit and optimality gap tolerance as Troika. For most benchmark instances, we showed that Troika improves the lower quality partitions of Combo even if a split second of extra time is available. Another advantage of Troika over a heuristic CP method is that for high-quality partitions of Combo, Troika ensures that the partition satisfies the user-specified optimality gap tolerance. In certain cases, networks with more than 5000 edges can also be processed by Troika to obtain a guaranteed approximation of the optimal partition within a reasonable time. This was exemplified in the analysis of the "lecturers" network instance which has over 300,000 edges and nearly 800 nodes. A solution for this large instance is approximated within 0.01 of the optimal in 248.65 seconds by Troika. Remarkably, Gurobi IP fails to converge or even reach Troika's approximate solution for the "lecturers" instance in 4 hours.

Despite the relative efficiency of Troika, achieving global optimality in the CP problem for certain network structures and larger networks remains a practical challenge that no exact or approximation method has solved to the best of our knowledge. For these networks, Troika offers flexibility by allowing the user to specify an optimality gap tolerance or a specific time limit as stopping criteria. This ensures that the algorithm returns a partition alongside the maximum optimality gap, when operating under constrained conditions.

We made a connection between the CP problem and optimization-based community detection. Using the modularity objective function as an example to make this connection explicit, we provided a reduction that converts any modularity maximization instance into a CP instance. This reduction makes the Troika algorithm capable of approximating the maximum modularity of a network and finding the partition that satisfies a user-specified optimality gap tolerance. Comparing this secondary usage of Troika to eight algorithms that were deliberately designed for modularity maximization, we observed that Troika outperforms them in returning partitions with closer proximity to the globally maximum modularity partitions.

We also demonstrated a real-world use case of the CP problem by deploying the Troika algorithm for analyzing the networks of correlations of returns among the S&P 500 stocks. Partitions obtained by Troika reveal the network-level temporal changes for a major part of

the US stock market over the analyzed period of 2000-2024. The results showed that Troika is useful for clustering networks of correlations. Specifically for portfolio networks, it uncovers temporal changes to the network structure, including the 2008 financial crisis and COVID-19 impacts on the clusters of positively correlated stocks within the S&P 500 market index.

From a practical perspective, Troika addresses a challenge in computational science by offering an effective method for CP filling a much-needed gap on approximating globally optimal solutions for small and mid-sized instances of practical relevance. In the future, exploring alternative lower bound heuristics will be crucial in developing CP approximation algorithms of higher efficacy. We hope this work facilitates future developments in network clustering and optimization.

8. Materials and methods

This section provides the technical details of the Troika algorithm. These technical details are the building blocks of the Troika algorithm that allow it to approximate the optimal solution of the CP problem as demonstrated in Sects 4–6. In Sect 8.1, we explain two graph pre-processing steps that reduce the size of the graph input that Troika receives and builds optimization models for. In Sect 8.2, a trick from integer programming is operationalized in Troika to increase its efficiency by fixing values of certain binary decision variables. In Sect 8.3, another integer programming technique is operationalized in Troika which (1) generates additional cuts that strengthen the optimization model and (2) fixes the values of additional decision variables. Finally, Sect 8.4 discusses the design choices of Troika on how node triples are prioritized and selected to be branched on for an efficient exploration of the feasible space. A schematic representation of the key steps in the Troika algorithm is provided as a flowchart in Fig 8.

8.1. Graph pre-processing

Troika leverages two graph pre-processing steps: pendant clique and node reduction, and component-wise processing for disconnected graphs.

8.1.1. Component-wise processing for disconnected graphs. If the input Graph G is not fully connected, it is divided into its separate connected components. In the context of the CP problem, since only internal weights are considered in the objective function, each component's objective value can be optimized separately, with the collective partition forming the output. This straightforward decomposition of a problem substantially reduces the total number of variables involved, thus enhancing the performance of the algorithm in networks that have several components (disconnected graphs).

8.1.2. Pendant cliques and node reduction. Additionally, the Troika algorithm benefits from recognizing specific structural patterns, such as pendant nodes and cliques, to expedite feasible space exploration. In any optimal solution, each pendant node (i.e. node incident on precisely one edge) that has a positive degree, always belongs to the same cluster as its sole neighbour. We transform the original graph G into a reduced graph G' where each positive-degree ($d_i > 0$) pendant node i is replaced with a self-loop at the neighboring node with a weight of d_i. The weight of the self-loop reflects the contribution of the reduced positively weighted edge to the optimal objective function value. If the edge weight is negative, a separate cluster is created for the pendant node to be later appended to the output partition. Pendant node reduction is illustrated in Fig 9.

Cliques are defined as complete sub-graphs with all internal edges bearing positive weights. Considering a node to be a 1-clique, the pendant node reduction idea can be generalized to reductions for pendant cliques of arbitrary size s. An s-clique is pendant if all its nodes

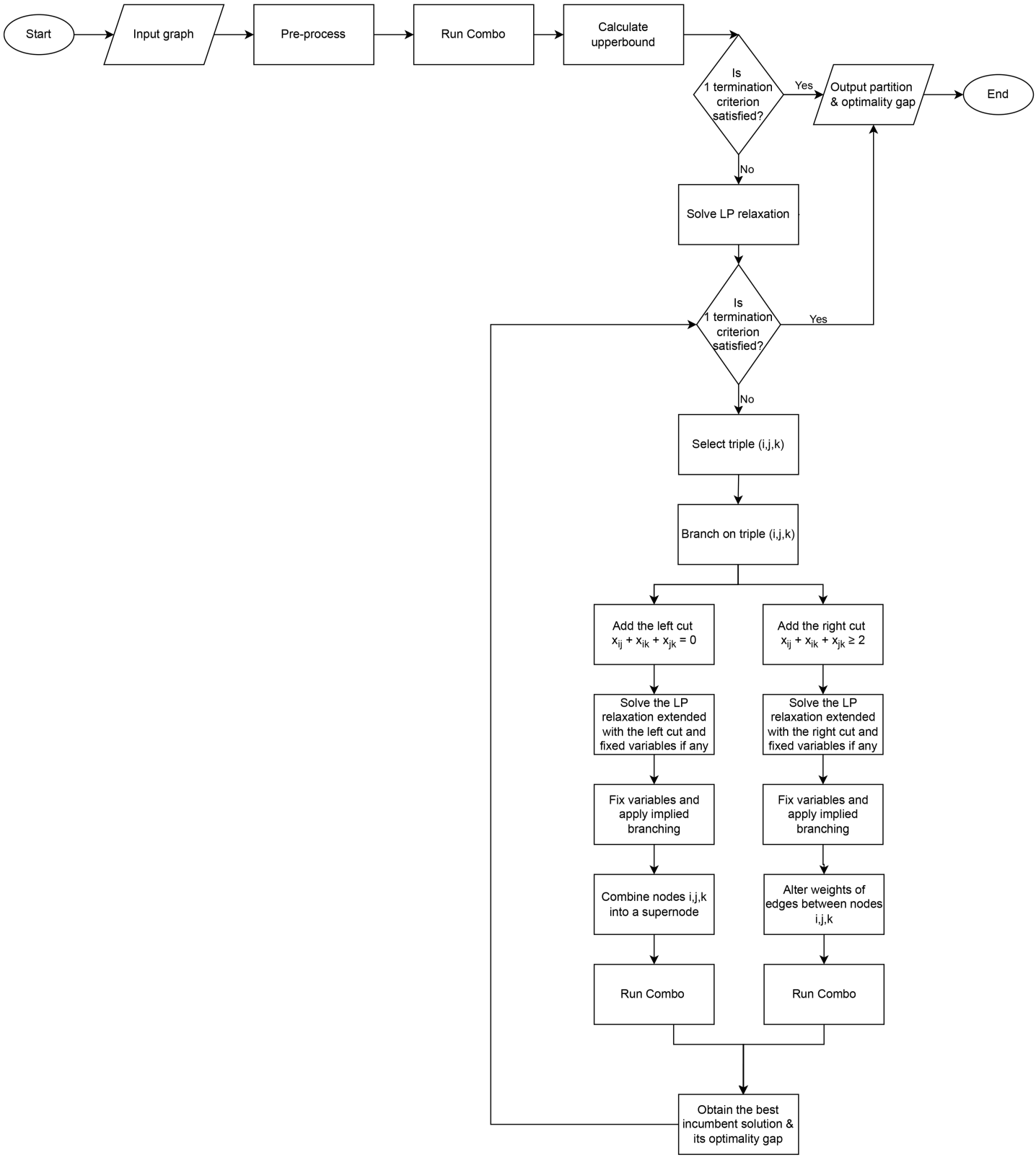


Fig 8. A flowchart of the key parts of the Troika algorithm.

https://doi.org/10.1371/journal.pcsy.0000062.g008

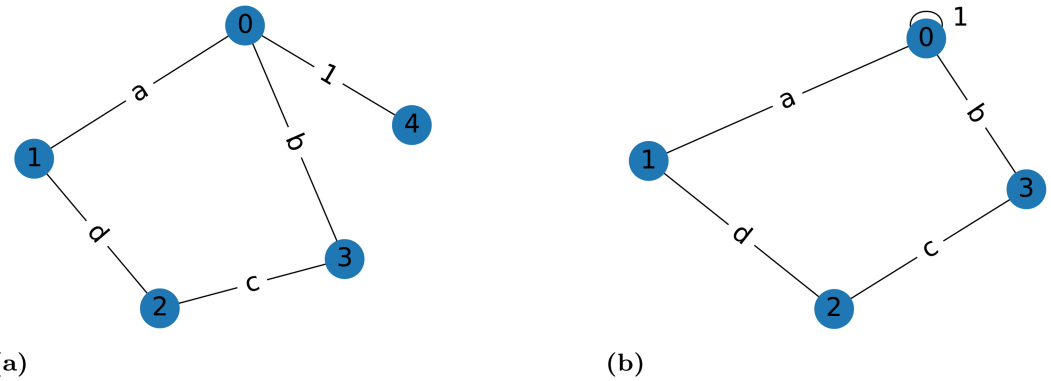


Fig 9. (a) Pendant node 4 is positively connected to its sole neighbor node 0. (b) After reduction, node 4 and its edge are replaced with a self-loop at node 0 that has the same weight as the edge (0,4).

https://doi.org/10.1371/journal.pcsy.0000062.g009

are incident on s–1 edges, except one node that is incident on s edges. The exceptional node is called a *connector*. Pendant cliques can be replaced by a self-loop on the connector node whose weight accounts for the contribution of the clique to the optimal objective function value. So, each pendant clique (of any size) can be condensed into a self-loop on the corresponding connector node, ensuring the allocation of all nodes of the clique to the same cluster. Pendant clique reduction is shown in Fig 10. This pre-processing step may substantially reduce the number of variables and constraints in the optimization models within the Troika algorithm. An alternative and more general pre-processing approach for CP is discussed in [56].

8.2. Variable fixing

Troika utilizes a variable fixing technique to solve the IP faster. Variable fixing can be used to determine the definitive value of certain variables at a point in the branch and cut process and for all subsequent LPs. Specifically, a binary variable x_{ij} can be fixed to either zero or one

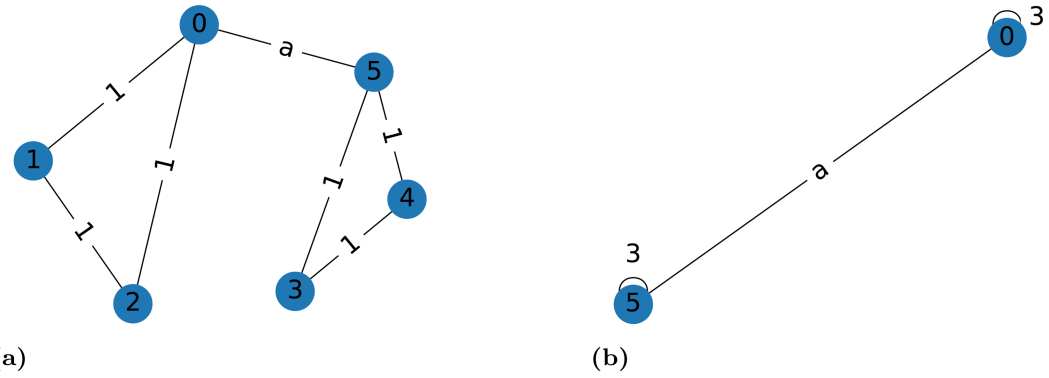


Fig 10. (a) Input graph G with two distinct positive cliques: the first comprising nodes 0, 1, and 2, and the second consisting of nodes 3, 4, and 5. (b) The post-reduction graph G', where each of the two cliques is replaced with a self-loop at their corresponding two connector nodes 0 and 5. Weights of each self-loop equals the sum of positive weights of the clique that it has replaced.

https://doi.org/10.1371/journal.pcsy.0000062.g010

when its reduced cost surpasses a certain threshold in the current LP's optimal solution. Suppose x_{ij} is set to zero with a reduced cost of c_{ij}, in an optimal LP solution, where the optimal objective function value is denoted as z^*_{LP}. It is observed that c_{ij} takes a negative value when x_{ij} is at its lower bound (i.e., zero). Let LB represent the objective value of the incumbent solution within the B&B search tree. It is deduced that any feasible solution with $x_{ij} = 1$ will have an objective value not exceeding $z^*_{LP} + c_{ij}$. Consequently, x_{ij} is fixed to zero if $z^*_{LP} + c_{ij} \leq LB$. Conversely, c_{ij} will be positive if x_{ij} is at its upper bound (i.e., one), prompting us to fix x_{ij} to one if $z^*_{LP} - c_{ij} \leq LB$.

Upon applying these conditions to fix variables, Troika proceeds to determine the states of additional variables through logical deductions. For instance, if nodes i and j belong to the same cluster (x_{ij} fixed to zero) and nodes k and j do not share the same cluster (x_{jk} fixed to one), it logically follows that nodes i and k must be in different clusters, leading to x_{ik} being fixed to one. This logical implication is formalized as follows:

$$x_{ij} = 0 \wedge x_{jk} = 1 \rightarrow x_{ik} = 1 \tag{8}$$

8.3. Implied branching and fixing

We further explore the synergies between branching and variable fixing to increase the convergence speed of the algorithm. By drawing logical conclusions from the states of fixed variables and the already established branching cuts, further variables can be fixed and additional cuts can be introduced to enhance the separation in both the right and left branches of the B&B tree [20].

Consider a scenario in the right branch where the constraint $x_{ij} + x_{jk} + x_{ik} \geq 2$, pertaining to the triple (i,j,k), has been added to the LP formulation. Suppose there exists a fixed variable related to one of these nodes, for instance, $x_{ip} = 0$. This precondition enables the introduction of a novel cut into the LP model as follows:

$$x_{ij} + x_{jk} + x_{ik} \geq 2 \wedge x_{ip} = 0 \rightarrow x_{jk} + x_{jp} + x_{kp} \geq 2. \tag{9}$$

Similarly, in the context of a left branch, consider the constraint $x_{ij} + x_{jk} + x_{ik} = 0$ has been added to the LP model. If a variable x_{ip}, associated with one of the nodes in the triple (i,j,k), is fixed, two more variables can be fixed as illustrated below:

$$x_{ij} + x_{jk} + x_{ik} = 0 \wedge x_{ip} = 0 \rightarrow x_{jp} = 0,\ x_{kp} = 0, \tag{10}$$

$$x_{ij} + x_{jk} + x_{ik} = 0 \wedge x_{ip} = 1 \rightarrow x_{jp} = 1,\ x_{kp} = 1. \tag{11}$$

The integration of logical inferences with variable states serves a dual purpose: it not only simplifies the process of fixing variables but also supports the creation of strategic cuts. This dual functionality considerably accelerates the Troika algorithm.

8.4. Triple selection for branching

Troika attempts to select the best triple for branching, facilitating an earlier detection of infeasibility, and enabling more variable fixing opportunities.

During each search iteration, the algorithm first identifies all triples that violate the two constraints in Eqs (5) and (6). Subsequently, it prioritizes certain triples for further evaluation based on their respective edge weights. Specifically, as defined in Sect 3.1, we introduced $\mathcal{T}$

as the union of the subsets T^k_+, T^j_+, and T^i_+, over which the transitivity constraints are defined. Moreover, $\mathcal{T}$ can be partitioned to $\mathcal{T} = T_3 \cup T_2 \cup T_1$ where

- T_3 denotes the triples that precisely have three strictly positive edge weights,
- T_2 denotes the triples that precisely have two strictly positive edge weights, and
- T_1 denotes the triples that precisely have one strictly positive edge weight.

The Troika algorithm adopts a structured approach for triple selection, starting with the triples in T_3. It only proceeds to select triples from T_2 after all triples in T_3 have been utilized. Similarly, selection from T_1 commences only after all triples in T_2 are exhausted. The selection of the best triple from the chosen subset is then guided by three key B&B-node-specific factors: (1) the greater overlap between its nodes with those in the triples already used for branching, (2) the count of its associated fixed variables, and (3) the absolute degree of its nodes. A binary indicator, β_i, is assigned the value one if node i is included in any branching triples of the parent nodes. The quantity of fixed variables linked to node i is represented as f_i. For every node i, possessing a degree d_i, we compute a score s_i according to Eq (12), incorporating β_i, f_i, and d_i to address the three outlined criteria.

$$s_i = 1 - e^{-f_i} + \beta_i + \frac{|d_i|}{n-1} \tag{12}$$

The collective score for a triple (i,j,k) is calculated as the sum of s_i, s_j, and s_k. Selection of a triple for branching is then carried out using a roulette wheel selection mechanism based on the calculated scores of the candidate triples.

8.5. Troika also solves the correlation clustering problem

The Troika algorithm also solves another fundamental graph clustering problem that can be defined on weighted graphs: the *Correlation Clustering* (CC) problem [23]. In the *MaxAgree* version of the correlation clustering problem with the input $G = (V, E, \mathbf{W})$, we look for a partition $\mathbf{X}$ that corresponds to the maximum value for the $\text{Agreement}_{(G,\mathbf{X})}$ objective function: sum of within-cluster positive edge weights plus the sum of absolute values of between-cluster negative edge weights. This objective function simplifies to:

$$\text{Agreement}_{(G,\mathbf{X})} \tag{13}$$

$$= \sum_{(i,j)\in E, w_{i,j}>0} w_{i,j}(1 - x_{ij}) + \sum_{(i,j)\in E, w_{i,j}<0} |w_{i,j}|(x_{ij}) \tag{14}$$

$$= \sum_{(i,j)\in E, w_{i,j}>0} w_{ij}(1 - x_{ij}) - \sum_{(i,j)\in E, w_{i,j}<0} |w_{ij}|(1 - x_{ij}) + \sum_{(i,j)\in E, w_{i,j}<0} |w_{ij}| \tag{15}$$

$$= \sum_{(i,j)\in E} w_{ij}(1 - x_{ij}) + \sum_{(i,j)\in E, w_{i,j}<0} |w_{ij}| \tag{16}$$

The rightmost term in Eq (16) is a constant function of G and does not depend on the partition $\mathbf{X}$. The other term in Eq (16) is the objective function of the CP problem. Therefore, any optimal partition of CP for input $G = (V, E, \mathbf{W})$ is also an optimal partition for the MaxAgree CC problem [57,Lemma 1]. This reduces the MaxAgree CC problem to a CP problem. Maximum agreement of G equals its maximum sum of within-cluster weights plus the constant $\sum_{(i,j)\in E, w_{i,j}<0} |w_{ij}|$.

Another version of the CC problem is the *MinDisagee correlation clustering* [23]. Its objective function is minimizing the sum of absolute values of within-cluster negative edge weights

plus the sum of between-cluster positive edge weights. MaxAgree and MinDisagree CC are equivalent in the sense that one is a linear transformation of the other [57] and the two problems have the same optimal partitions. This can be explained by the observation that agreement and disagreement objective functions for any partition add up to a constant value (the sum of absolute values of weights). Given this equivalence, Troika also solves the MinDisagree correlation clustering problem.

Note that in some alternative definitions of the correlation clustering problem [58], every pair of nodes (i,j) has two nonnegative weights: a similarity weight sim_{ij} and a dissimilarity weight dis_{ij}. In such a case, the edge weights for the input graph $G = (V, E, \mathbf{W})$ can be obtained according to $w_{ij} = \text{sim}_{ij} - \text{dis}_{ij}$ [57].

8.6. Accessing the Troika code and the network data

A Python implementation of the Troika algorithm is publicly available on GitHub https://github.com/saref/troika.

Each of the 53 real networks used in Sect 5 was loaded from the network repository Netzschleuder as simple unweighted and undirected graph G. Then, a CP problem was defined according to the weighted graph G' whose edge weights are the entries of the modularity matrix of G. The data for all these real networks (G) are available in a FigShare data repository [36]. The same FigShare repository contains data on all synthetic networks (LFR and ABCD networks) used in Sect 5 as well as all BA instances used in Sect 4.5 and all portfolio networks used in Sect 6. Other networks used in our study were from [29]. Sørensen and Letchford [29] have shared their CP instances and the optimal solutions that were available in a public GitHub repository https://github.com/MMSorensen/CP-Lib.

Supporting information

The supplementary information for this article includes:

S1 Table. Detailed numerical results of the three methods on ABR instances.
(PDF)

S2 Table. Numerical results for Troika and eight modularity-based algorithms on 100 real and synthetic networks.
(PDF)

S3 Table. Details of the optimal partitions obtained by Troika on the S&P 500 networks.
(PDF)

Acknowledgments

Troika algorithm is built on a foundation of earlier developments made accessible by other researchers. We are thankful to Alexander Belyi, Stanislav Sobolevsky, Alexander Kurbatski, Carlo Ratti, Riccardo Campari, Philipp Kats, Mahdi Mostajabdaveh, and Hriday Chheda for making their algorithms publicly accessible. We are also thankful for the helpful comments from Martijn Gösgens, the anonymous reviewers of this journal, and those of the 2024 workshop on Complex Networks in Banking and Finance.

Author contributions

Conceptualization: Samin Aref.

Data curation: Samin Aref.

Formal analysis: Samin Aref.

Funding acquisition: Samin Aref.

Investigation: Samin Aref, Boris Ng.

Methodology: Samin Aref.

Project administration: Samin Aref.

Resources: Samin Aref.

Software: Samin Aref, Boris Ng.

Supervision: Samin Aref.

Validation: Samin Aref.

Visualization: Samin Aref, Boris Ng.

Writing – original draft: Samin Aref, Boris Ng.

Writing – review & editing: Samin Aref.

References

1. Grötschel M, Wakabayashi Y. A cutting plane algorithm for a clustering problem. Mathematical Programming. 1989;45(1–3):59–96. https://doi.org/10.1007/bf01589097
2. Aref S, Wilson MC. Balance and frustration in signed networks. Journal of Complex Networks. 2018;7(2):163–89. https://doi.org/10.1093/comnet/cny015
3. Wakabayashi Y. Aggregation of binary relations: algorithmic and polyhedral investigations. 1986.
4. Koshimura M, Watanabe E, Sakurai Y, Yokoo M. Concise integer linear programming formulation for clique partitioning problems. Constraints. 2022;27(1–2):99–115. https://doi.org/10.1007/s10601-022-09326-z
5. Aref S, Mostajabdaveh M, Chheda H. Heuristic Modularity Maximization Algorithms for Community Detection Rarely Return an Optimal Partition or Anything Similar. In: Computational Science–ICCS 2023: 23rd International Conference, Prague, Czechia, Proceedings. 2023. p. 612–26.
6. Gao J, Lv Y, Liu M, Cai S, Ma F. Improving simulated annealing for clique partitioning problems. JAIR. 2022;74:1485–513. https://doi.org/10.1613/jair.1.13382
7. Brimberg J, Janićijević S, Mladenović N, Urošević D. Solving the clique partitioning problem as a maximally diverse grouping problem. Optim Lett. 2015;11(6):1123–35. https://doi.org/10.1007/s11590-015-0869-4
8. Palubeckis G, Ostreika A, Tomkevičius A. An iterated tabu search approach for the clique partitioning problem. ScientificWorldJournal. 2014;2014:353101. https://doi.org/10.1155/2014/353101 PMID: 24737968
9. Pacheco JA, Casado S. A stepped tabu search method for the clique partitioning problem. Appl Intell. 2022;53(12):16275–92. https://doi.org/10.1007/s10489-022-04304-7
10. Zhou Y, Hao J-K, Goëffon A. A three-phased local search approach for the clique partitioning problem. J Comb Optim. 2015;32(2):469–91. https://doi.org/10.1007/s10878-015-9964-9
11. Lu Z, Zhou Y, Hao J-K. A hybrid evolutionary algorithm for the clique partitioning problem. IEEE Trans Cybern. 2022;52(9):9391–403. https://doi.org/10.1109/TCYB.2021.3051243 PMID: 33635807
12. Belalta R, Belazzoug M, Meziane F. A graph based named entity disambiguation using clique partitioning and semantic relatedness. Data & Knowledge Engineering. 2024;152:102308. https://doi.org/10.1016/j.datak.2024.102308
13. Simanchev RYu, Urazova IV, Kochetov YuA. The branch and cut method for the clique partitioning problem. J Appl Ind Math. 2019;13(3):539–56. https://doi.org/10.1134/s1990478919030153
14. Belyi A, Sobolevsky S, Kurbatski A, Ratti C. Subnetwork constraints for tighter upper bounds and exact solution of the clique partitioning problem. Math Meth Oper Res. 2023;98(2):269–97. https://doi.org/10.1007/s00186-023-00835-y
15. Letchford AN, Sørensen MM. In: 2024. 1–3.

16. Letchford AN, Sørensen MM. New facets of the clique partitioning polytope. Operations Research Letters. 2025;59:107242. https://doi.org/10.1016/j.orl.2025.107242
17. Irmai J, Naumann LF, Andres B. Chorded cycle facets of clique partitioning polytopes. arXiv preprint 2024. http://arxiv.org/abs/2411.03407
18. Du Y, Kochenberger G, Glover F, Wang H, Lewis M, Xie W, et al. Solving clique partitioning problems: a comparison of models and commercial solvers. Int J Info Tech Dec Mak. 2021;21(01):59–81. https://doi.org/10.1142/s0219622021500504
19. Levin MSh. On the clique partitioning of a graph. J Commun Technol Electron. 2022;67(S2):S267–74. https://doi.org/10.1134/s1064226922140042
20. Aref S, Mostajabdaveh M, Chheda H. Bayan algorithm: detecting communities in networks through exact and approximate optimization of modularity. Phys Rev E. 2024;110(4–1):044315. https://doi.org/10.1103/PhysRevE.110.044315 PMID: 39562863
21. Miyauchi A, Sonobe T, Sukegawa N. Exact clustering via integer programming and maximum satisfiability. AAAI. 2018;32(1). https://doi.org/10.1609/aaai.v32i1.11519
22. Brandes U, Delling D, Gaertler M, Gorke R, Hoefer M, Nikoloski Z, et al. On modularity clustering. IEEE Trans Knowl Data Eng. 2008;20(2):172–88. https://doi.org/10.1109/tkde.2007.190689
23. Bansal N, Blum A, Chawla S. Correlation clustering. Machine Learning. 2004;56(1–3):89–113. https://doi.org/10.1023/b:mach.0000033116.57574.95
24. Miyauchi A, Sukegawa N. Redundant constraints in the standard formulation for the clique partitioning problem. Optim Lett. 2014;9(1):199–207. https://doi.org/10.1007/s11590-014-0754-6
25. Sobolevsky S, Campari R, Belyi A, Ratti C. General optimization technique for high-quality community detection in complex networks. Phys Rev E Stat Nonlin Soft Matter Phys. 2014;90(1):012811. https://doi.org/10.1103/PhysRevE.90.012811 PMID: 25122346
26. Kats P. PyCombo: A Python wrapper around the C implementation of the network community detection algorithm Combo. https://pypi.org/project/pycombo/
27. Gurobi Optimization Inc. Gurobi Optimizer Reference Manual. 2025.
28. Miltenberger M. Interactive visualizations of Mittelmann benchmarks. 2025. https://github.com/mattmilten/mittelmann-plots
29. Sørensen MM, Letchford AN. CP-Lib: benchmark instances of the clique partitioning problem. Mathematical Programming Computation. 2023. p. 1–19.
30. Nogueira Lorena LH, Goncalves Quiles M, Nogueira Lorena LA, de Carvalho ACPLF, Cespedes JG. Qualitative data clustering: a new Integer Linear Programming model. In: 2019 International Joint Conference on Neural Networks (IJCNN). 2019. p. 1–8. https://doi.org/10.1109/ijcnn.2019.8851969
31. Brusco MJ, Köhn H-F. Clustering qualitative data based on binary equivalence relations: neighborhood search heuristics for the clique partitioning problem. Psychometrika. 2009;74(4):685–703. https://doi.org/10.1007/s11336-009-9126-z
32. Dorndorf U, Pesch E. Fast clustering algorithms. ORSA Journal on Computing. 1994;6(2):141–53. https://doi.org/10.1287/ijoc.6.2.141
33. Aref S, Mason AJ, Wilson MC. A modeling and computational study of the frustration index in signed networks. Networks. 2019;75(1):95–110. https://doi.org/10.1002/net.21907
34. Barabasi A, Albert R. Emergence of scaling in random networks. Science. 1999;286(5439):509–12. https://doi.org/10.1126/science.286.5439.509 PMID: 10521342
35. Zadorozhnyi VN, Yudin EB. Structural properties of the scale-free Barabasi-Albert graph. Autom Remote Control. 2012;73(4):702–16. https://doi.org/10.1134/s0005117912040091
36. Aref S, Ng B. Dataset of networks used in assessing the Troika algorithm for clique partitioning and community detection. 2025. http://doi.org/10.6084/m9.figshare.28835837
37. Schaub MT, Delvenne J-C, Rosvall M, Lambiotte R. The many facets of community detection in complex networks. Appl Netw Sci. 2017;2(1):4. https://doi.org/10.1007/s41109-017-0023-6 PMID: 30533512
38. Aref S, Mathiyarasan S. Robust Markov stability for community detection at a scale learned based on the structure. In: Proceedings of the ACM on Human-Computer Interaction, 2025.
39. Aref S, Mostajabdaveh M. Analyzing modularity maximization in approximation, heuristic, and graph neural network algorithms for community detection. Journal of Computational Science. 2024;78:102283. https://doi.org/10.1016/j.jocs.2024.102283
40. Newman MEJ. Modularity and community structure in networks. Proc Natl Acad Sci U S A. 2006;103(23):8577–82. https://doi.org/10.1073/pnas.0601602103 PMID: 16723398
41. Lancichinetti A, Fortunato S. Limits of modularity maximization in community detection. Phys Rev E Stat Nonlin Soft Matter Phys. 2011;84(6 Pt 2):066122. https://doi.org/10.1103/PhysRevE.84.066122 PMID: 22304170

42. Clauset A, Newman MEJ, Moore C. Finding community structure in very large networks. Phys Rev E Stat Nonlin Soft Matter Phys. 2004;70(6 Pt 2):066111. https://doi.org/10.1103/PhysRevE.70.066111 PMID: 15697438
43. Blondel VD, Guillaume J-L, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. J Stat Mech. 2008;2008(10):P10008. https://doi.org/10.1088/1742-5468/2008/10/p10008
44. Zhang P, Moore C. Scalable detection of statistically significant communities and hierarchies, using message passing for modularity. Proc Natl Acad Sci U S A. 2014;111(51):18144–9. https://doi.org/10.1073/pnas.1409770111 PMID: 25489096
45. Bonald T, Charpentier B, Galland A, Hollocou A. Hierarchical Graph Clustering using Node Pair Sampling. In: MLG 2018 - 14th International Workshop on Mining and Learning with Graphs, London, UK. 2018. p. 1–8.
46. Traag VA, Waltman L, van Eck NJ. From Louvain to Leiden: guaranteeing well-connected communities. Sci Rep. 2019;9(1):5233. https://doi.org/10.1038/s41598-019-41695-z PMID: 30914743
47. Li PZ, Huang L, Wang CD, Lai JH. Mot: an edge enhancement approach for motif-aware community detection. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining; 2019. p. 479–87.
48. Sobolevsky S, Belyi A. Graph neural network inspired algorithm for unsupervised network community detection. Appl Netw Sci. 2022;7(1):1–19.https://doi.org/10.1007/s41109-022-00500-z
49. Rossetti G, Milli L, Cazabet R. CDLIB: a python library to extract, compare and evaluate communities from complex networks. Appl Netw Sci. 2019;4(1). https://doi.org/10.1007/s41109-019-0165-9
50. Lancichinetti A, Fortunato S, Radicchi F. Benchmark graphs for testing community detection algorithms. Phys Rev E Stat Nonlin Soft Matter Phys. 2008;78(4 Pt 2):046110. https://doi.org/10.1103/PhysRevE.78.046110 PMID: 18999496
51. Kamiński B, Prałat P, Théberge F. Artificial Benchmark for Community Detection (ABCD)—fast random graph model with community structure. Net Sci. 2021;9(2):153–78. https://doi.org/10.1017/nws.2020.45
52. Fisher RA. On the “probable error” of a coefficient of correlation deduced from a small sample. Metron. 1921;1:3–32.
53. Jovanovic R, Sanfilippo AP, Voß S. Fixed set search applied to the clique partitioning problem. European Journal of Operational Research. 2023;309(1):65–81. https://doi.org/10.1016/j.ejor.2023.01.044
54. Friedman M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association. 1937;32(200):675–701. https://doi.org/10.1080/01621459.1937.10503522
55. (David) Li J. A two-step rejection procedure for testing multiple hypotheses. Journal of Statistical Planning and Inference. 2008;138(6):1521–7. https://doi.org/10.1016/j.jspi.2007.04.032
56. Belyi A, Sobolevsky S. Network size reduction preserving optimal modularity and clique partition. In: International Conference on Computational Science and Its Applications, 2022. p. 19–33.
57. Gösgens M, van der Hofstad R, Litvak N. The projection method: a unified formalism for community detection. Front Complex Syst. 2024;2. https://doi.org/10.3389/fcpxs.2024.1331320
58. Charikar M, Guruswami V, Wirth A. Clustering with qualitative information. Journal of Computer and System Sciences. 2005;71(3):360–83.