Network Size Reduction Preserving Optimal Modularity and Clique Partition

Alexander Belyi[1(✉)] and Stanislav Sobolevsky[1,2]

[1] Department of Mathematics and Statistics, Faculty of Science, Masaryk University, Brno, Czech Republic
{bely,sobolevsky}@math.muni.cz
[2] Center for Urban Science and Progress, New York University, Brooklyn, NY, USA

Abstract. Graph clustering and community detection are significant and actively developing topics in network science. Uncovering community structure can provide essential information about the underlying system. In this work, we consider two closely related graph clustering problems. One is the clique partitioning problem, and the other is the maximization of partition quality function called modularity. We are interested in the exact solution. However, both problems are NP-hard. Thus the computational complexity of any existing algorithm makes it impossible to solve the problems exactly for the networks larger than several hundreds of nodes. That is why even a small reduction of network size can significantly improve the speed of finding the solution to these problems. We propose a new method for reducing the network size that preserves the optimal partition in terms of modularity score or the clique partitioning objective function. Furthermore, we prove that the optimal partition of the reduced network has the same quality as the optimal partition of the initial network. We also address the cases where a previously proposed method could provide incorrect results. Finally, we evaluate our method by finding the optimal partitions for two sets of networks. Our results show that the proposed method reduces the network size by 40% on average, decreasing the computation time by about 54%.

Keywords: Network size reduction · Clustering · Community detection · Modularity · Clique partitioning problem · Exact solution

1 Introduction

Data clustering is a common problem in machine learning and data science [20]. Since many real-world systems are naturally represented as networks or graphs (we use these terms interchangeably as synonyms), the task of clustering network nodes also frequently arises [7] and has applications ranging from group technology [27,31], biology [15], and biochemistry [17], to the transportation [10], geoinformatics [4], tourism management [32] and the study of social networks [14]. It is a particular interest of many researchers because most real-world networks possess a community or modular structure [26], which provides essential insights

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
O. Gervasi et al. (Eds.): ICCSA 2022, LNCS 13375, pp. 19–33, 2022.
https://doi.org/10.1007/978-3-031-10522-7_2

into the underlying system's properties and dynamics. Community detection in complex networks became a broad and actively developing field of network science dedicated entirely to the problem of network partitioning [12,21].

Proper clustering could mean different things depending on a network and a particular goal. Intuitively, a community in a network is a group of similar or closely related nodes. One common approach to formulating the clustering problem is to define a function that measures the quality of a partition. Then clustering could be done by finding the best partition according to this quality, that is, by maximizing partition quality as an objective function.

Consider a system represented as a complete directed graph $G = (V, E, W)$, where $E = V \times V$ and $W = \{w_{ij} \in \mathbb{R} \mid (i,j) \in E\}$. Individual objects correspond to vertices V, and every ordered pair of objects is represented as an arc (or directed edge) with weight w_{ij}. If arc weights of graph G represent similarity or dissimilarity between objects, and these weights can be positive or negative, then one way to obtain a meaningful partition could be to group together similar vertices connected with positive arcs while keeping dissimilar vertices in different clusters. In other words, the best partition will be the one that maximizes the sum of arc weights that fall within clusters. This intuitive approach could be mathematically formalized as follows: for the graph G, the best partition C (represented as a mapping from vertices to corresponding cluster labels, $C(i) = c_i, i \in V, c_i \in \mathbb{N}$) is the one where the sum $\sum_{i,j|c_i=c_j} w_{ij}$ is maximal. The problem of finding such partition is called the clique partitioning problem (CPP), and it is NP-hard if there are edges of both signs [30].

However, in many real-world systems, the similarity between objects is unknown or not quantified. Moreover, we would often like clustering to reflect something different, for example, close connections or frequent interactions between objects. In such networks, edges could represent various relations and often are unweighted, or their weights have the same (usually positive) sign. In such cases, a good partition would put apart nodes that are not connected or have weaker connections and group together densely connected nodes. The problem of clustering such networks is known as the community detection problem, and there are many approaches to address it. [13].

One widely adopted way to detect communities is to first replace the original network G with a new network $G' = (V, E' = V \times V, W')$ having the same set of vertices V but new edge weights $W' = \{q_{ij} \in \mathbb{R} \mid (i,j) \in E'\}$, and then cluster network G'. Every new weight q_{ij} indicates how different the original weight w_{ij} was from the weight edge (i,j) would have in a null model, i.e., if weights were assigned randomly, preserving nodes' incoming and outgoing strengths $s_i^{in} = \sum_j w_{ji}$ and $s_i^{out} = \sum_j w_{ij}$. The expected edge weight in this model could be approximated by $\frac{s_i^{in} s_j^{out}}{S}$, where $S = \sum_{i,j} w_{ij}$. Then the new edge weight is calculated as

$$q_{ij} = \frac{1}{S}\left(w_{ij} - \frac{s_i^{in} s_j^{out}}{S}\right).$$

The weights are normalized by S so that the sum of all weights cannot get larger than 1. These weights can be positive or negative. Larger values of q indicate that the connection between corresponding nodes is stronger than one would expect in a random graph, meaning that it must be an important connection, and the nodes should be grouped. Analogously, a larger absolute value of a negative weight means that corresponding nodes should be assigned to different groups. This way, we arrived at the clique partitioning problem: the best partition into communities is the one that maximizes the sum of weights q within communities. This sum is called a modularity function. It was proposed by Girvan and Newman [14] and is usually denoted as

$$Q = \sum_{i,j|c_i=c_j} q_{ij}.$$

Note that q_{ij} and q_{ji} contribute or do not contribute to the sum together, and values q_{ii} contribute to the sum independently of partition C. Therefore, if we define $D = \sum_i q_{ii}$ and redefine q_{ij} as $q_{ij} := q_{ji} := (q_{ij} + q_{ji})/2$, we can rewrite Q as follows:

$$Q = 2 \sum_{i<j|c_i=c_j} q_{ij} + D.$$

To unify our reasoning in the rest of the paper, even if we consider the clique partitioning problem for the original graph G, we will refer to the individual edge weights as q and assume $q_{ij} = q_{ji}$.

Even though modularity scores are not arbitrary and have some structure that could be exploited when maximizing Q, the problem of finding the exact maximum is still NP-hard [9]. That is why most researchers develop heuristics to quickly find high-quality partitions [8,18,23,29], while only a few methods guarantee to find the exact maximum [2,24].

2 Exact Solution of the CPP and Modularity Maximization

While a couple of approaches rely on the branch-and-bound technique [5,6,19], most of the exact methods for solving the CPP employ integer linear programming (ILP) formulation of the problem initially proposed in the seminal works of Grötschel and Wakabayashi [15,16] and later applied to modularity maximization by Agarwal and Kempe [1]. In addition, recent works showed how standard optimization packages could be successfully applied to solving the CPP [11,22,24].

To formulate the ILP problem, for every edge (i, j), define a binary variable x_{ij} that equals 1 when the edge falls within a community and 0 otherwise. Then the objective of the CPP is to

$$\begin{aligned} \text{maximize:} \quad & Q = 2 \cdot \sum_{i<j} q_{ij} \cdot x_{ij} + D, \\ \text{subject to:} & \\ x_{ij} + x_{jk} - x_{ik} \leq 1, \quad & \text{for all } 1 \leq i < j < k \leq n \\ x_{ij} - x_{jk} + x_{ik} \leq 1, \quad & \text{for all } 1 \leq i < j < k \leq n \\ -x_{ij} + x_{jk} + x_{ik} \leq 1, \quad & \text{for all } 1 \leq i < j < k \leq n \\ x_{ij} \in \{0,1\}, \quad & \text{for all } 1 \leq i < j \leq n. \end{aligned} \tag{1}$$

Constraints are called *triangle inequalities* and ensure consistency of partition, i.e., if both edges (i, j) and (j, k) connect nodes from the same cluster, then edge (i, k) does too. Grötschel and Wakabayashi [15] empirically showed that there are many redundant constraints in (1). More recently, Miyauchi and Sukegawa [25] proved that it is sufficient to include only inequalities where at least one of the edges corresponding to two positive terms has non-negative weight. So, the problem can be rewritten as:

$$\begin{aligned} \text{maximize} \quad & Q = 2 \cdot \sum_{i<j} q_{ij} \cdot x_{ij} + D, \\ \text{subject to:} & \\ x_{ij} + x_{jk} - x_{ik} \leq 1, \quad & \forall 1 \leq i < j < k \leq n, q_{ij} \geq 0 \vee q_{jk} \geq 0 \\ x_{ij} - x_{jk} + x_{ik} \leq 1, \quad & \forall 1 \leq i < j < k \leq n, q_{ij} \geq 0 \vee q_{ik} \geq 0 \\ -x_{ij} + x_{jk} + x_{ik} \leq 1, \quad & \forall 1 \leq i < j < k \leq n, q_{jk} \geq 0 \vee q_{ik} \geq 0 \\ x_{ij} \in \{0,1\}, \quad & \forall 1 \leq i < j \leq n. \end{aligned} \tag{2}$$

Standard optimization packages can be directly applied to this problem. However, the problem size grows very quickly with the size of the network, so the exact solution is only feasible for networks with up to several hundreds of vertices.

3 Network Size Reduction

The number of variables in the ILP formulation (2) of the CPP is quadratic, and the number of constraints is cubic in the number of graph vertices, while the computational time needed to solve the problem is not bounded by a polynomial. Thus, even a modest reduction in the number of graph vertices can significantly improve our ability to solve the CPP exactly.

Two studies were dedicated to network size reduction aiming to speed up community detection algorithms [3,22]. However, we are not aware of any methods of graph size reduction preserving the CPP objective function. In what follows, first, we discuss known size reduction methods preserving modularity and then describe our generalization and extension for the general case of the CPP.

Arenas et al. [3] derived conditions when nodes with only one connection (which they call *hairs*, see Fig. 1a) could be merged with their neighbors, preserving the

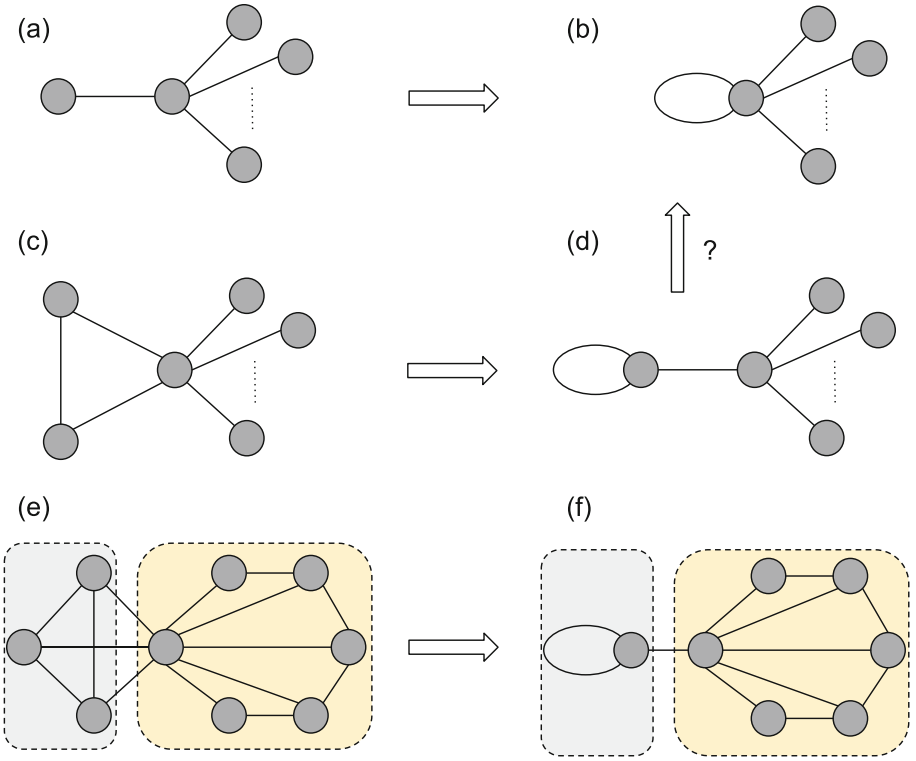

Fig. 1. Examples of network size reduction: a to b (*hair*) and c to d (*triangular hair*) were proposed by Arenas et al. [3], e to f (*peripheral clique*) was proposed by Lorena et al. [22]. In figures e and f, the partition of optimal modularity for the shown graph is highlighted. It must be noted that in figure f, the node on the left falls in its own community separate from other nodes.

optimal partition's modularity. Such merging involves replacing two connected nodes with one node and a loop edge, so a graph from Fig. 1a becomes a graph from Fig. 1b. The authors also showed when two connected nodes with one common neighbor and no other neighbors (*triangular hairs*, see Fig. 1c) could be merged into one node, becoming a hair with a self-loop (Fig. 1d), and then possibly (under some conditions on the weights of connecting edge and self-loop) merged with its neighbor (so, in Fig. 1, graph c becomes graph d, and then in some cases – graph b). They showed that this approach considerably reduced their modularity optimization heuristic's running time and improved obtained modularity scores. They also argued that their approach would allow the merging of 40 to 83% of nodes in scale-free networks and networks with exponential degree distribution.

In recent work, Lorena et al. [22] extended the work of Arenas et al. They hypothesized that any *peripheral clique* could be merged into one node (in Fig. 1, that corresponds to reducing graph e to graph b). While this is often the case,

and it was true for all the networks they used in their experiments, this is not always correct. For example, the graph from Fig. 1e can only be reduced to the graph in Fig. 1f and not to the graph in Fig. 1b, which agrees with the previous result of Arenas et al. As we will prove next, any *peripheral clique* could be merged into a *hair* with a loop edge, but merging this *hair* with its neighbor could be done only under some conditions that must be checked.

Let us now consider a general CPP and two nodes, i and j. We start by describing what it means to merge i and j, preserving objective function and optimal partition. When we merge i and j, we replace the initial network G with a new network G^* identical to G except that nodes i and j and all their adjacent edges are replaced by a new node i^* connected to itself with a loop edge $q_{i^*i^*} = q_{ii} + q_{jj} + q_{ij} + q_{ji}$, and to every other node k with an edge of weight $q_{ki^*} = q_{i^*k} = q_{ik} + q_{jk}$. We say that such merging preserves objective function because any partition of G where i and j belong to one cluster has the same quality as the corresponding partition of G^*. Merging two nodes preserves the optimal partition if these two nodes belong to one cluster in the optimal partition.

There are at least two cases when we can quickly determine that i and j will be in the same cluster in the optimal partition. The first one is when i and j are connected with a very strong edge, so strong that keeping them apart and not including the edge would worsen any partition. The sufficient condition for this is

$$q_{ij} \geq \sum_{k \in V \setminus \{i,j\} | q_{ik} > 0} q_{ik} - \sum_{k \in V \setminus \{i,j\} | q_{ik} < 0} q_{ik}. \tag{3}$$

Indeed, in any partition where i and j are kept apart, if (3) is true, then moving i to the cluster of j increases the value of the objective function. A pair of *strongly connected* nodes i, j that satisfies (3) covers the case of a *hair* node in modularity maximization: Arenas et al. [3] showed that if i is a *hair* connected only to j and does not have a self-loop, inequality (3) holds.

The following theorem describes the second case when two nodes can be merged, preserving optimal partition.

Theorem 1. *Nodes i and j with a non-negative edge between them ($q_{ij} \geq 0$), identically connected to every other node in G (i.e., with edges of the same weight $q_{ik} = q_{jk}$ for all $k \in V \setminus \{i, j\}$), will appear in the same cluster in the optimal partition.*

Proof. Consider a partition where nodes i and j belong to different communities $c_i \neq c_j$. If we move node i from c_i to c_j, the objective function will change by the value

$$\Delta Q_{i \to j} = q_{ij} + \sum_{k \neq j | c_j = c_k} q_{ik} - \sum_{k \neq i | c_i = c_k} q_{ik},$$

and if we move j from c_j to c_i, then Q will change by

$$\Delta Q_{j \to i} = q_{ij} + \sum_{k \neq i | c_i = c_k} q_{jk} - \sum_{k \neq j | c_j = c_k} q_{jk}.$$

Since in the sums, all $q_{ik} = q_{jk}$,

$$\Delta Q_{i\to j} + \Delta Q_{j\to i} = 2q_{ij} \geq 0.$$

It follows then that either $\Delta Q_{i\to j} \geq 0$ or $\Delta Q_{j\to i} \geq 0$, which means that overall partition quality Q could be improved (or at least will not decrease) either by moving i to the community of j or by moving j to the community of i. □

The conditions described in Theorem 1 are satisfied, for example, by the nodes in a *peripheral clique*: the modularity score of every edge within such a clique is the same and positive, and every node is connected to the same set of nodes. It follows then that each pair of nodes from the clique will be in the same cluster in the optimal partition, so they can be merged into one node, as shown in Fig. 1e–f. With a correction regarding the articulation point connecting the clique to the rest of the graph, Theorem 1 proves what Lorana et al. [22] showed empirically, that a *peripheral clique* could be merged into one node, preserving modularity. However, Theorem 1 allows the merging of other nodes too. The following section describes algorithms that implement finding and merging nodes satisfying inequality (3) or Theorem 1.

4 Nodes Merging Algorithms

First, we introduce an auxiliary procedure to merge two nodes, preserving objective function and optimal partition. It formalizes the steps described in the previous section.

Algorithm 1: Merge two nodes

input : Graph $G = (V, E = V \times V, W = \{q_{ij}\})$, nodes u and v to be merged
output: New graph G^*

define node w that will represent merged nodes u and v in G^*;
$G^* = \big(V^* = \{w\} \cup V \setminus \{u, v\}, E^* = V^* \times V^*, W^* = \{q^*_{ij}\}\big)$;
$q^*_{ww} = q_{uu} + q_{vv} + 2 \cdot q_{uv}$;
foreach *node* $i \in V \setminus \{u, v\}$ **do**
    $q^*_{wi} = q^*_{iw} = q_{ui} + q_{vi}$;
    foreach *node* $j \in V \setminus \{u, v\}$ **do**
        $q^*_{ij} = q_{ij}$; `// All other weights remain the same`
return G^*;

Theorem 1 defines an equivalence relation on the set of nodes V. To reduce network G, we first compare each node to all other nodes, assign it to the corresponding equivalence class, and then merge the nodes of each class into one node representing it. Algorithm 2 formalizes this procedure.

Algorithm 2: Find and merge *identically connected* nodes

input : Graph $G = (V, E = V \times V, W = \{q_{ij}\})$
output: Updated graph G

assign each node to represent its own equivalence class: $classes(i) = i$;
foreach *node* $i \in V$ **do**
  foreach *node* $j \in V \setminus \{i\}$ **do**
    if $q_{ij} \geq 0$ *and* $q_{ik} == q_{jk}$ *for* $\forall k \in V \setminus \{i, j\}$ **then**
      $classes(i) = classes(j) = \min(classes(i), classes(j))$;
foreach *node* $i \in V$ **do**
  if $i \neq classes(i)$ **then**
    $G =$ `MergeTwoNodes(`G, i, $classes(i)$`)` ; `// Algorithm 1`
return G;

When G is a complete graph, Algorithm 2 uses tree nested for-loops twice: to find identically connected nodes and merge them. Therefore, it has time complexity $O(n^3)$, where $n = |V|$ is the size of the graph.

Next, we provide an algorithm that finds and merges the nodes satisfying inequality (3).

Algorithm 3: Find and merge *strongly connected* nodes

input : Graph $G = (V, E = V \times V, W = \{q_{ij}\})$
output: Updated graph G

foreach *node* $i \in V$ **do**
  $sum_positive = \sum_{j \in V \setminus \{i\} | q_{ij} > 0} q_{ij}$;
  $sum_negative = \sum_{j \in V \setminus \{i\} | q_{ij} < 0} q_{ij}$;
  $max_q = \max_{j \in V \setminus \{i\}} q_{ij}$;
  $j = \arg\max_{j \in V \setminus \{i\}} q_{ij}$;
  if $max_q \geq sum_positive - max_q - sum_negative$ **then**
    $G =$ `MergeTwoNodes(`G, i, j`)` ; `// Algorithm 1`
return G;

Algorithm 3 might merge almost all nodes, spending $O(n^2)$ operations on each merge. Hence its time complexity is also $O(n^3)$.

Applying Algorithm 2 can produce new *strongly connected* and *identically connected* nodes, as well as applying Algorithm 3 can produce new *identically connected* and new *strongly connected* nodes (see Fig. 2 for some examples). So, to decrease the graph size as much as possible, we apply Algorithms 2 and 3 in turn as long as at least one of them reduces the graph. Since the size of the graph decreases on each iteration, there are no more than n steps, and the overall time complexity of our size reduction procedure is $O(n^4)$. Nevertheless, our experiments show that actual execution time is still negligible for the networks where the CPP can be solved exactly.

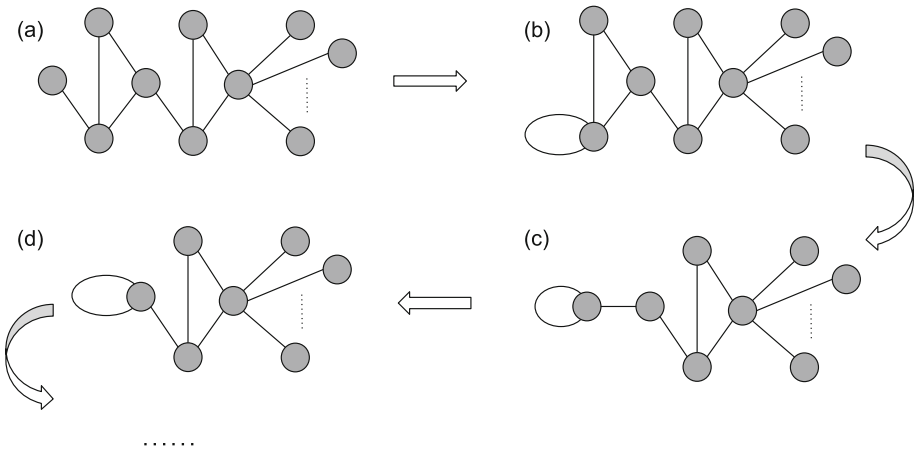

Fig. 2. An example of network size reduction iterations: after merging two *strongly connected* nodes (a to b), a new pair of *identically connected* nodes appeared and was merged (b to c), which in turn created a pair of *strongly connected* nodes, which were merged (c to d), creating another pair of *strongly connected* nodes, and so on. The weights of shown positive and omitted negative edges could be easily adjusted to satisfy all necessary conditions and are not shown in the figure to preserve clarity.

5 Evaluation and Results

To evaluate our approach, we wrote a program that loads networks, applies Algorithms 2 and 3 to reduce network sizes, creates the ILP problem (2) for the CPP or modularity maximization, and uses the commercial solver IBM ILOG CPLEX 20.1.0 to solve the ILP models obtaining the optimal partition. We implemented our algorithms in C++, compiled the program using Clang 12.0.5, and ran the experiments on a laptop with a 3.2 GHz CPU and 16 GB RAM. We tested our method on two sets of real-world networks previously used in the literature on the exact solution of the modularity maximization problem and the CPP. All our codes and the datasets analyzed in this work are freely available on GitHub[1].

First, we focused on the modularity maximization problem. To compare our results with the results from [22], we mostly followed their evaluation strategy and used the same set of networks. All the networks were downloaded from the Network Repository website [28]. Each network was treated as unweighted and undirected, and only the giant components of the networks were used in the experiments. Table 1 provides a summary of this benchmark. Each network was assigned an ID shown in the first column. The second column shows the network's name on the Network Repository website, n indicates the number of nodes, m is the number of edges, and Q is the modularity of the optimal partition.

[1] https://github.com/Alexander-Belyi/best-partition.

Table 1. Networks in the first benchmark.

ID	Network name	n	m	Q
1	lesmis	77	254	0.56001
2	GD00-A	83	125	0.53091
3	ca-sandi-auths	86	124	0.73683
4	rt-retweet	96	177	0.67967
5	netscience	379	914	0.84859
6	bio-DM-LC	483	997	0.77870
7	power-494-bus	494	586	0.85999
8	bio-diseasome	516	1188	0.83200
9	bio-grid-mouse	791	1098	0.80101
10	ca-CSphd	1025	1043	0.92558

Table 2 shows how our size reduction algorithms shrink the network sizes in terms of the number of nodes. This decrement in the network sizes corresponds to a significant reduction in the number of variables and the number of constraints in the ILP problem (2), as seen from the corresponding columns. Here and in the next table, we denote values obtained by Lorena et al. [22] with the prime symbol ($'$), and values obtained using our method – with an asterisk (*). As seen from the table, our method reduces the problem further for six networks out of ten. In the network bio-grid-mouse, our algorithm merged two nodes differently, reducing only the number of constraints. However, for one network, our method does not merge two nodes that the algorithm from [22] merges. That is because inequality (3) is not satisfied for these two nodes, and it is not known apriori that these *hairs* belong to the communities of their neighbors in the optimal partition.

Table 2. Network and ILP problem size reduction.

ID	n	n'	n^*	n^*/n	n^*/n'	Var	Var.$'$	Var.*	Constr	Constr.$'$	Constr.*
1	77	58	**46**	59.74 %	79.31 %	2926	1653	**1035**	34685	23110	**11817**
2	83	64	64	77.11 %	100.00 %	3403	2016	2016	18434	11581	11581
3	86	57	**52**	60.47 %	91.23 %	3655	1176	**1326**	20369	7085	**7267**
4	96	43	43	44.79 %	100.00 %	4560	903	903	21511	4954	4954
5	379	286	**254**	67.02 %	88.81 %	71631	40755	**32131**	682739	428071	**262472**
6	483	340	**328**	67.91 %	96.47 %	116403	57630	**53628**	946606	566353	**525236**
7	494	344	*346*	70.04 %	100.58 %	121771	58996	*59685*	575323	295959	*299067*
8	516	326	**287**	55.62 %	88.04 %	132870	52975	**41041**	1211786	578838	**324517**
9	791	*380*	*380*	48.04 %	100.00 %	312445	72010	72010	1716725	510436	**506702**
10	1025	332	332	32.39 %	100.00 %	524800	54946	54946	2128786	230136	230136

Table 3. Execution time.

ID	Network name	t (s)	T (s)	T' (s)	T^* (s)	T^*/T	T^*/T
1	lesmis	0.00031	0.61	0.42	**0.21**	34.43 %	50.00 %
2	GD00-A	0.00024	2.23	**1.46**	3.61	161.88 %	247.26 %
3	ca-sandi-auths	0.00039	0.41	0.15	**0.13**	31.71 %	86.67 %
4	rt-retweet	0.00059	0.64	0.18	**0.11**	17.19 %	61.11 %
5	netscience	0.01457	43.40	20.67	**7.20**	16.59 %	34.83 %
6	bio-DM-LC	0.02719	5315.57	1692.86	**985.47**	18.54 %	58.21 %
7	power-494-bus	0.02814	90419.51	**17586.84**	166681.31	184.34 %	947.76 %
8	bio-diseasome	0.04083	22966.99	3046.39	**480.13**	2.09 %	15.76 %
9	bio-grid-mouse	0.14692	26551.02	390.40	**307.46**	1.16 %	78.76 %
10	ca-CSphd	0.33687	2512.05	86.39	**41.23**	1.64 %	47.73 %

Finally, in Table 3, we report the execution time (in seconds) of our size reduction algorithms (t) and the total time to solve the ILP problems for the initial and reduced networks (T, T', and T^*, respectively). The execution times T and T' are taken from [22]. Provided values show that the time spent on the size reduction is negligible compared to the total time required to solve the problem. For almost all networks, size reduction leads to a significant decrease in the execution time. For some networks, observed improvements reach more than 98% compared to the initial network and up to 84% compared to the method from [22]. For two networks, the time to solve the ILP problem was higher than the time reported in [22]. The reason could be the solver's different version or parameter settings. This possibility is supported by the fact that in our experiment, the time to solve the full GD00-A network was 6.98 s, and for the power-494-bus network, the solver did not find a solution after 200,000 s. We also note here that it is not guaranteed that the smaller size of the problem

Table 4. Results for the networks in the second benchmark.

Network	n	n^*	n^*/n	Var	Var.*	Constr	Constr.*	t	T	T^*	T^*/T	Q
Wild cats	30	**25**	83%	435	**300**	10043	**5531**	0.00004	0.14	**0.08**	57%	1 304
Cars	33	**31**	93%	528	**465**	14708	**12027**	0.00003	0.22	**0.18**	82%	1 501
Workers	34	34	100%	561	561	14605	14605	0.00001	0.24	0.24	100%	964
Cetacea	36	**35**	97%	630	**595**	9798	**9207**	0.00002	0.13	**0.12**	92%	967
Micro	40	40	100%	780	780	22201	22201	0.00002	0.32	0.32	100%	1 034
UNO	54	**16**	29%	1431	**120**	45756	**606**	0.00019	0.71	**0.01**	1%	798
UNO 1a	158	**19**	12%	12403	**171**	1161623	**999**	0.00228	40.78	**0.01**	0%	12 197
UNO 1b	139	**10**	7%	9591	**45**	910908	**172**	0.00165	28.93	**0.00**	0%	11 775
UNO 2a	158	**75**	47%	12403	**2775**	1542583	**160952**	0.00195	35.50	**2.12**	6%	72 820
UNO 2b	145	**73**	50%	10440	**2628**	1310497	**152675**	0.00151	27.63	**1.98**	7%	71 818

makes it easier to solve, and in some cases, a smaller problem could be much harder than a larger one.

After applying our method in the setting of modularity maximization, we tested it in the setting of the CPP. To that end, we used a well-known set of real-world networks representing similarities between different objects. It was compiled by Grötschel and Wakabayashi [15] and later reused in several studies [11,19]. Table 4 presents the networks in this benchmark and summarizes the results. From it, one can see that the two networks were not reduced while the size of some other networks was reduced by more than ten times, and the computational time decreased by more than 90%, from seconds to milliseconds. All the networks in this set are relatively easy to solve even without reduction, although a recent study by Du et al. [11] reported the execution time of two minutes for some of the networks using CPLEX. Here we showed that applying a simple and fast preprocessing reduces the total time needed to solve the CPP by orders of magnitude.

6 Conclusion

Network clustering is an important and well-studied problem in network science. It gives rise to the clique partitioning problem (CPP) that can be used as a stand-alone clustering method or appear in community detection via modularity maximization. However, the computational complexity of the CPP is in the non-deterministic polynomial-time hard (NP-hard) class. Therefore, finding the exact solution of the CPP is only possible for networks of a small to moderate size. Usually, it is done by solving a corresponding integer linear programming (ILP) problem using one of the specialized solvers.

In this work, we proposed a new method for reducing the size of the network while preserving its optimal partition into clusters according to the CPP criterion. Our work extends two previous efforts in this direction applied to the modularity maximization problem [3,22], but we generalized our approach for the broader class of the CPP. Furthermore, by theoretically proving the correctness of our methods, we also proved the correct part of the approach proposed by Lorena et al. [22], which was validated only empirically. Moreover, we also pointed out that their approach was incorrect in the general case and showed how to address it. Even though their method removed some of the nodes that could not have been removed, our approach decreased the number of nodes by another 6% on average.

We proposed two algorithms implementing our method to identify and merge the appropriate vertices. Our experiments showed that for the networks of the size where an exact solution is possible, the total preprocessing time required by the algorithms is negligible compared to the expected time needed to solve the resulting ILP problem.

To evaluate our methods, we chose two sets of real-world networks previously used in the literature to evaluate algorithms for solving the CPP and modularity maximization exactly. Our experiments showed that our method considerably

decreases the size of many real-world networks. On average, the network size is reduced by about 40% for two datasets, and in some cases, we observe a reduction of more than 90%. Correspondingly, the size of the ILP problem was reduced substantially, which significantly reduced the time and memory consumption required to solve the problem exactly. The average time reduction was about 55% in both datasets we used. Such improvements in time efficiency allow the application of the exact solution methods to more extensive networks. One of the possible directions for future work is to apply and evaluate the utility of proposed methods to improve the performance of heuristics that do not find the exact solution but still could benefit from the smaller size of the problem.

Acknowledgement. This research was supported by the MUNI Award in Science and Humanities (MASH) of the Grant Agency of Masaryk University under the Digital City project (MUNI/J/0008/2021).

References

1. Agarwal, G., Kempe, D.: Modularity-maximizing graph communities via mathematical programming. Eur. Phys. J. B **66**(3), 409–418 (2008). https://doi.org/10.1140/epjb/e2008-00425-1
2. Aloise, D., Cafieri, S., Caporossi, G., Hansen, P., Perron, S., Liberti, L.: Column generation algorithms for exact modularity maximization in networks. Phys. Rev. E **82**(4), 46112 (2010). https://doi.org/10.1103/PhysRevE.82.046112
3. Arenas, A., Duch, J., Fernández, A., Gómez, S.: Size reduction of complex networks preserving modularity. New J. Phys. **9**(6), 176–176 (2007). https://doi.org/10.1088/1367-2630/9/6/176
4. Belyi, A., Bojic, I., Sobolevsky, S., Sitko, I., Hawelka, B., Rudikova, L., Kurbatski, A., Ratti, C.: Global multi-layer network of human mobility. Int. J. Geogr. Inf. Sci. **31**(7), 1381–1402 (2017). https://doi.org/10.1080/13658816.2017.1301455
5. Belyi, A., Sobolevsky, S., Kurbatski, A., Ratti, C.: Subnetwork constraints for tighter upper bounds and exact solution of the clique partitioning problem. arXiv preprint arXiv:2110.05627 (2021)
6. Belyi, A.B., Sobolevsky, S.L., Kurbatski, A.N., Ratti, C.: Improved upper bounds in clique partitioning problem. J. Belarusian State Univ. Math. Informatics **2019**(3), 93–104 (2019). https://doi.org/10.33581/2520-6508-2019-3-93-104
7. Benati, S., Puerto, J., Rodríguez-Chía, A.M.: Clustering data that are graph connected. Eur. J. Oper. Res. **261**(1), 43–53 (2017). https://doi.org/10.1016/j.ejor.2017.02.009
8. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. **2008**(10), P10008 (2008). https://doi.org/10.1088/1742-5468/2008/10/p10008
9. Brandes, U., Delling, D., Gaertler, M., Gorke, R., Hoefer, M., Nikoloski, Z., Wagner, D.: On modularity clustering. IEEE Trans. Knowl. Data Eng. **20**(2), 172–188 (2008). https://doi.org/10.1109/TKDE.2007.190689
10. Dorndorf, U., Jaehn, F., Pesch, E.: Modelling robust flight-gate scheduling as a clique partitioning problem. Transp. Sci. **42**(3), 292–301 (2008). https://doi.org/10.1287/trsc.1070.0211

11. Du, Y., Kochenberger, G., Glover, F., Wang, H., Lewis, M., Xie, W., Tsuyuguchi, T.: Solving clique partitioning problems: a comparison of models and commercial solvers. Int. J. Inf. Technol. Decis. Mak. **21**(01), 59–81 (2022). https://doi.org/10.1142/S0219622021500504
12. Fortunato, S.: Community detection in graphs. Phys. Rep. **486**, 75–174 (2010). https://doi.org/10.1016/j.physrep.2009.11.002
13. Fortunato, S., Hric, D.: Community detection in networks: a user guide. Phys. Rep. **659**, 1–44 (2016). https://doi.org/10.1016/j.physrep.2016.09.002
14. Girvan, M., Newman, M.E.J.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. **99**(12), 7821–7826 (2002). https://doi.org/10.1073/pnas.122653799
15. Grötschel, M., Wakabayashi, Y.: A cutting plane algorithm for a clustering problem. Math. Program. **45**(1), 59–96 (1989). https://doi.org/10.1007/BF01589097
16. Grötschel, M., Wakabayashi, Y.: Facets of the clique partitioning polytope. Math. Program. **47**(1), 367–387 (1990). https://doi.org/10.1007/BF01580870
17. Guimerà, R., Nunes Amaral, L.A.: Functional cartography of complex metabolic networks. Nature **433**(7028), 895–900 (2005). https://doi.org/10.1038/nature03288
18. Hu, S., Wu, X., Liu, H., Li, R., Yin, M.: A novel two-model local search algorithm with a self-adaptive parameter for clique partitioning problem. Neural Comput. Appl. **33**(10), 4929–4944 (2020). https://doi.org/10.1007/s00521-020-05289-5
19. Jaehn, F., Pesch, E.: New bounds and constraint propagation techniques for the clique partitioning. Discret. Appl. Math. **161**(13), 2025–2037 (2013). https://doi.org/10.1016/j.dam.2013.02.011
20. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: a review. ACM Comput. Surv. **31**(3), 264–323 (1999). https://doi.org/10.1145/331499.331504
21. Javed, M.A., Younis, M.S., Latif, S., Qadir, J., Baig, A.: Community detection in networks: a multidisciplinary review. J. Netw. Comput. Appl. **108**, 87–111 (2018). https://doi.org/10.1016/j.jnca.2018.02.011
22. Lorena, L.H.N., Quiles, M.G., Lorena, L.A.N.: Improving the performance of an integer linear programming community detection algorithm through clique filtering. In: Misra, S., Gervasi, O., Murgante, B., Stankova, E., Korkhov, V., Torre, C., Rocha, A.M.A.C., Taniar, D., Apduhan, B.O., Tarantino, E. (eds.) Improving the Performance of an Integer Linear Programming Community Detection Algorithm Through Clique Filtering. LNCS, vol. 11619, pp. 757–769. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24289-3_56
23. Lu, Z., Zhou, Y., Hao, J.K.: A hybrid evolutionary algorithm for the clique partitioning problem. IEEE Trans. Cybern., 1–13 (2021). https://doi.org/10.1109/TCYB.2021.3051243
24. Miyauchi, A., Sonobe, T., Sukegawa, N.: Exact Clustering via Integer Programming and Maximum Satisfiability. Proc. AAAI Conf. Artif. Intell. **32**(1) (2018)
25. Miyauchi, A., Sukegawa, N.: Redundant constraints in the standard formulation for the clique partitioning problem. Optim. Lett. **9**(1), 199–207 (2014). https://doi.org/10.1007/s11590-014-0754-6
26. Newman, M.E.J.: The structure and function of complex networks. SIAM Rev. **45**(2), 167–256 (2003). https://doi.org/10.1137/S003614450342480
27. Oosten, M., Rutten, J.H.G.C., Spieksma, F.C.R.: The clique partitioning problem: facets and patching facets. Networks **38**(4), 209–226 (2001). https://doi.org/10.1002/net.10004
28. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: AAAI (2015). http://networkrepository.com

29. Sobolevsky, S., Campari, R., Belyi, A., Ratti, C.: General optimization technique for high-quality community detection in complex networks. Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys. **90**(1) (2014). https://doi.org/10.1103/PhysRevE.90.012811
30. Wakabayashi, Y.: Aggregation of binary relations: algorithmic and polyhedral investigations. Ph.D. thesis, Doctoral Dissertation. University of Augsburg (1986)
31. Wang, H., Alidaee, B., Glover, F., Kochenberger, G.: Solving group technology problems via clique partitioning. Int. J. Flex. Manuf. Syst. **18**(2), 77–97 (2006). https://doi.org/10.1007/s10696-006-9011-3
32. Xu, Y., Li, J., Belyi, A., Park, S.: Characterizing destination networks through mobility traces of international tourists - a case study using a nationwide mobile positioning dataset. Tour. Manag. **82** (2021). https://doi.org/10.1016/j.tourman.2020.104195