Memetic Computing (2022) 14:253–285
https://doi.org/10.1007/s12293-022-00365-w

REGULAR RESEARCH PAPER

Metaheuristic approaches for ratio cut and normalized cut graph partitioning

Gintaras Palubeckis[1]

Received: 5 October 2021 / Accepted: 3 April 2022 / Published online: 29 April 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
Partitioning a set of graph vertices into two or more subsets constitutes an important class of problems in combinatorial optimization. Two well-known members of this class are the minimum ratio cut and the minimum normalized cut problems. Our focus is on developing metaheuristic-based approaches for ratio cut and normalized cut graph partitioning. We present three techniques in this category: multistart simulated annealing (MSA), iterated tabu search (ITS), and the memetic algorithm (MA). The latter two use a local search procedure. To speed up this procedure, we apply a technique that reduces the effort required for neighborhood examination. We carried out computational experiments on both random graphs and benchmark graphs from the literature. The numerical results indicate that the MA is a clear winner among the tested methods. Using rigorous statistical tests, we show that MA is unequivocally superior to MSA and ITS in terms of both the best and average solution values. Additionally, we compare the performances of MA and the variable neighborhood search (VNS) heuristic from the literature, which is the state-of-the-art algorithm for the normalized cut model. The experimental results demonstrate the superiority of MA over VNS, especially for structured graphs.

Keywords Combinatorial optimization · Graph partitioning · Ratio cut · Normalized cut · Metaheuristics · Memetic algorithm

1 Introduction

Partitioning a set of objects into two or more subsets constitutes an important class of problems in combinatorial optimization. A member of this class can often be modeled by a graph whose vertices represent objects and whose edges link vertices pairs that have some kind of relationship. Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. For an integer $m \geq 2$, a solution to a graph partitioning problem is a set $p = \{V_1, \ldots, V_m\}$ in which V_i, $i \in M = \{1, \ldots, m\}$, are nonempty and mutually disjoint subsets of V such that $\cup_{i \in M} V_i = V$. The various graph partitioning problems differ in the objective function and/or in the constraints. In many cases, the objective function incorporates the total edge weights connecting vertices from different subsets in partition p. Let the edge weights of graph G be denoted by c_{uv}, $(u, v) \in E$. For $V_k, V_l \subset V$, $V_k \cap V_l = \emptyset$, the sum $C(V_k, V_l) = \sum_{u \in V_k, v \in V_l} c_{uv}$ is called the cut between subsets V_k and V_l. There are several graph partitioning problems that require minimizing the cut between partition subsets (or clusters). One of them is the ratio cut graph partitioning problem. For a graph G and fixed integer $m \geq 2$, it is stated as

$$\min_{p \in \Pi} F_r(p) = \sum_{k=1}^{m} C(V_k)/|V_k| \tag{1}$$

where $C(V_k)$ is the shortcut to $C(V_k, V \setminus V_k)$ and Π is the set of all partitions of V into m nonempty subsets. Thus, the contribution of a partition subset V_k to (1) is represented by the ratio of the cut between V_k and the rest of the graph to the cardinality of V_k. Problem (1) was first considered by Wei and Cheng [69] and Hagen and Kahng [25] for $m = 2$ and then generalized to a multiway ratio cut (i.e., for $m > 2$) by Chan et al. [11]. Later, Shi and Malik [58] introduced another objective function called normalized cut. The corresponding graph partitioning problem is expressed as follows:

✉ Gintaras Palubeckis
gintaras.palubeckis@ktu.lt;
gintaras.palubeckis77@gmail.com

[1] Faculty of Informatics, Kaunas University of Technology, Studentu 50-408, 51368 Kaunas, Lithuania

$$\min_{p\in\Pi} F_n(p) = \sum_{k=1}^{m} C(V_k)/d(V_k) \tag{2}$$

where $d(V_k) = \sum_{v\in V_k} d_v$ and the sum $d_v = \sum_{u\in V} c_{vu}$ is referred to as the weighted degree of the vertex $v \in V$. In (2), the term for a partition subset V_k is the ratio of the cut between V_k and the rest of the graph to the sum of the weighted degrees of the vertices in V_k. Yu and Shi [70] extended the normalized cut model to $m > 2$ partition subsets.

A salient feature of both the ratio cut and normalized cut models is that no constraints are imposed on the partition subset size. This makes a significant difference versus graph partitioning problems in which the size of each subset is bounded from above and below. A well-studied problem of this sort is the maximally diverse grouping problem (MDGP). Many algorithms for solving the MDGP have been proposed, including variable neighborhood search [5,51], hybrid genetic algorithms [51,60], multistart simulated annealing [51], tabu search with strategic oscillation [22], the artificial bee colony algorithm [55], iterated tabu search [52], iterated maxima search [36], and neighborhood decomposition-based variable neighborhood search and tabu search [37]. Other related graph partitioning problems include maximum k-cut [15], ratio association [16], minimum conductance graph partitioning [10,43,44], overlapping normalized cut [63], cohesive clustering [7], partition size constrained minimum cut [1], and edge-ratio clustering [6].

Many applications of the ratio cut and normalized cut graph partitioning problems have been identified in the literature. Perhaps the main application of these graph partitioning models can be observed in clustering. Ratio cut and normalized cut-based clustering methods can be used in a variety of domains, such as image data analysis [14], pattern recognition [33,61], web search [63], data mining [3], image segmentation [28,58], and gene network analysis [16]. Other applications include community detection [32,39,48], data classification [47,73], VLSI design [25], tree segmentation [35], salient object detection [21], robot swarm dynamic regrouping [53], Bayesian-type statistics [59], water distribution network partitioning [71], and detecting similar groups in heterogeneous networks [40].

It is well known that both the ratio cut and normalized cut graph partitioning problems are NP-hard [25,30]. Therefore, it is necessary to develop heuristic algorithms for these problems. Exact methods can be used to solve problem instances with very small sizes only. Fan and Pardalos [17] presented quadratically constrained programs that can be used to find optimal ratio and normalized cuts. They reported computational results for graphs with 10 vertices. For large graphs, one must resort to heuristic algorithms, which provide good but not necessarily optimal solutions. Most of the efforts in this direction have been focused on developing spectral methods for graph partitioning. The basic idea of these methods is to relax (1) and (2) into continuous optimization problems. The latter can be solved by first computing the eigenvectors of the Laplacian matrix of the graph and then finding the final partition using k-means or other suitable algorithms. Many studies have been devoted to the design and performance evaluation of spectral algorithms for ratio cut and normalized cut, including those by Hagen and Kahng [25], Chan et al. [11], Fan and Pardalos [17], Hochbaum [30], Merkurjev et al. [47], Zhang et al. [72], and Han et al. [26]. Many authors have proposed various improvements to the standard spectral clustering method. Lu et al. [42] presented nonnegative and sparse spectral clustering algorithms based on the ratio cut and normalized cut criteria. Experiments have shown that these algorithms outperform the standard spectral clustering technique. Chen et al. [12] proposed a direct normalized cut algorithm exploiting the idea of directly optimizing the normalized cut model. The algorithm has only a quadratic time complexity. Chen et al. [13] considered a new normalized cut model with balance regularization to avoid a trivial solution. They developed an iterative method to solve the new model without using eigendecomposition. More comprehensive discussions of the spectral method for graph clustering and partitioning can be found in the tutorial by von Luxburg [64] and survey papers by Nascimento and de Carvalho [49] and Gallier [23].

Although the traditional approach to ratio cut and normalized cut graph partitioning relies on using the spectral method, there are also algorithms that are based on different principles. Jia et al. [33] proposed an approximate weighted kernel k-means algorithm for the normalized cut. The algorithm avoids the direct eigendecomposition of the Laplacian matrix and is suitable for handling very large graphs. Lorente-Leyva et al. [41] outlined two alternative approaches for the normalized cut partitioning problem. One approach is a heuristic search procedure, and the other is a quadratic formulation-based method. Dhillon et al. [16] designed a fast multilevel algorithm that can be tuned to minimize specific objectives, including ratio cut and normalized cut. The refinement step of the algorithm employs the weighted kernel k-means technique. Fan and Pardalos [17] presented several semidefinite programming relaxations for ratio and normalized cut models. They developed a graph partitioning algorithm whose main step involves solving one of these relaxations. Compared with the spectral clustering method, this algorithm can obtain improved solutions.

However, the literature lacks metaheuristic-based approaches for finding the minimum ratio cut or normalized cut in the graph. One such approach was proposed by Hansen et al. [27]. They developed a variable neighborhood search (VNS) algorithm for the normalized cut model. Their local search (LS) procedure within the VNS framework analyzes all the possibilities of moving a vertex from one partition subset to

another. The algorithm also employs a fast LS technique that considers only moves between connected subsets (two vertex subsets V_k and V_l are connected if there is an edge whose one vertex belongs to V_k and the other vertex belongs to V_l). During VNS execution, this technique is combined with the complete LS strategy. The experimental results show that the VNS method of Hansen et al. [27] is an efficient approach for solving the normalized cut graph partitioning problem. Metaheuristic-based approaches have also been developed for several related problems. Cafieri et al. [6] proposed a VNS algorithm for graph bipartitioning with the edge-ratio criterion. The algorithm is embedded into a hierarchical divisive heuristic to obtain a partition of the vertex set of the graph into a larger number of subsets. Mu et al. [48] and Ji et al. [32] presented ant colony optimization algorithms for community detection in complex networks. In their approach, the ratio cut is included as a linear term in the objective function (modularity density) of the problem. Lu et al. [43] developed a hybrid evolutionary algorithm for finding minimum conductance in the graph. The algorithm employs a tabu search technique as a local optimization procedure.

Literature analysis shows that the ratio cut and normalized cut graph partitioning problems have been mostly addressed by using various heuristic techniques. Most often, they are obtained through continuous relaxations of ratio or normalized cut models. Very little research has been devoted to metaheuristic optimization approaches although graph partitioning and clustering have many applications in a wide range of areas, as outlined earlier in this section. Considering these observations, our motivation is to implement and investigate the performance of metaheuristic-based algorithms for graph partitioning with ratio cut and normalized cut criteria. The main significance of this work lies in the development and experimental comparison of several metaheuristic algorithms for these graph partitioning problems. In the case of the normalized cut criterion, the comparison includes both the new algorithms and the best existing method. The significance of comparative analysis of various approaches is that it can help identify promising directions for designing better algorithms. The results of our computational study underscore the potential for evolutionary methods for solving graph partitioning problems. The main contribution of our work consists of three diverse algorithms for ratio cut and normalized cut graph partitioning. A simulated annealing (SA) algorithm is selected because of its popularity and success in solving complex combinatorial optimization problems. To be able to apply a CPU-based termination rule, we implemented SA as a multistart procedure. The iterated tabu search (ITS) algorithm is chosen because the tabu search (TS) is one of the most widely used local search methods. To potentially achieve better performance, we apply TS iteratively. Population-based evolutionary approaches in our study are represented by the memetic algorithm (MA). We prefer MA over the genetic algorithm (GA), because MA usually demonstrates faster convergence and better optimization than GA. The algorithm choice was guided by two reasons. First, our focus is on metaheuristics that exist for a long time and have shown excellent performance in solving numerous combinatorial optimization problems. Second, we consider the diversity factor and do not limit ourselves to considering only evolutionary techniques. It seems that some of the new population-based methods can perform equally well or even outperform the ITS and SA algorithms. In recent years, many effective strategies have been proposed to address optimization problems. They include a memetic algorithm with competition [68], iterated local search with tabu search [50], a multistart iterated tabu search algorithm [4], brain storm optimization with an orthogonal learning mechanism [45], and a decomposition-based algorithm using a localized control variable analysis approach [46].

We experimentally compare the proposed algorithms on two different sets of graphs: random graphs and benchmark graphs from the literature. We report results for both ratio cut and normalized cut graph partitioning scenarios. We also present comparison results between our best algorithm and the state-of-the-art VNS-based algorithm of Hansen et al. [27].

The remainder of the paper is organized as follows. In Sects. 2, 3, and 4, we present the multistart simulated annealing (MSA), iterated tabu search, and memetic algorithms, respectively. Section 5 is devoted to experimental analysis and comparisons of algorithms. Concluding remarks are given in Sect. 6.

2 Multistart simulated annealing

In this section, we present an implementation of the SA method for ratio cut and normalized cut graph partitioning. This method is based on an analogy between the metallurgical process of annealing in thermodynamics and the process of searching for the global extremum of a function. This analogy has been efficiently exploited by Kirkpatrick et al. [34] and Černý [8]. The guiding principle of the SA method is to escape a local optimum by accepting worsening moves with a certain probability. A recent overview of different SA variants can be found in [20].

The idea of simulated annealing is to generate trial solutions in the neighborhood of the current solution and either accept or reject them. A new solution is always accepted if it improves the current solution. Otherwise, a decision on acceptance or rejection is made at random with a probability depending on the difference between objective function values of two solutions and the current temperature of the cooling process. Because the search starts with a high initial temperature, this probability is higher at the initial steps of

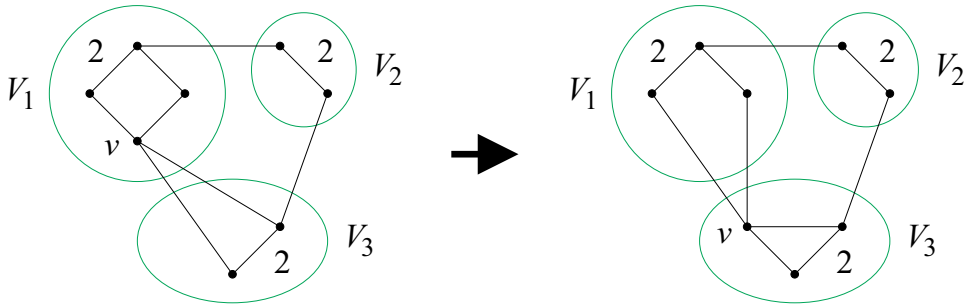


Fig. 1 Illustration of the relocation move: vertex v is transferred to subset V_3

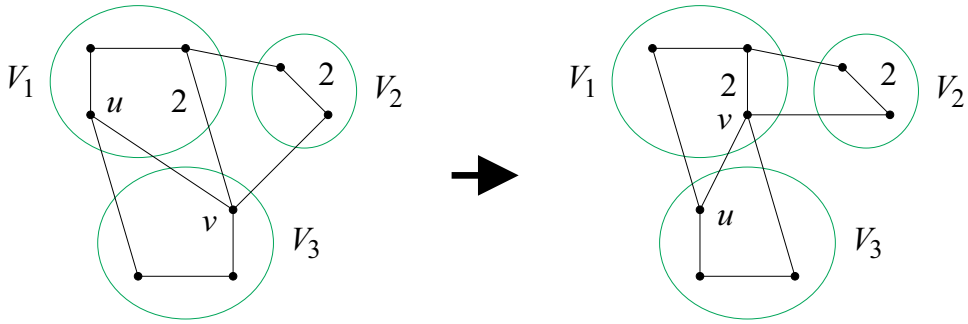


Fig. 2 Illustration of the swap move: the vertices u and v are interchanged

the algorithm. As the algorithm proceeds, the temperature is reduced after each iteration. The algorithm stops whenever the temperature becomes very close to zero. At each temperature level, a certain number of trial solutions are evaluated. In our implementation of SA, the initial temperature is obtained by generating a random solution and calculating the absolute difference between objective function values of the random solution and a solution in its neighborhood many times. The temperature is initialized with the largest of these differences. Another important design feature of our algorithm is that SA is executed multiple times.

Without loss of generality, we present our multistart simulated annealing algorithm in terms of the ratio cut metric. The adaptation of the algorithm to the case of normalized cut graph partitioning requires only very minor changes that come from using a different objective function. The algorithm employs two move types, relocating a vertex from its current subset to the other subset in a partition and swapping two vertices located in two distinct subsets. The relocation move is illustrated in Fig. 1, where vertex $v \in V_1$ is moved to partition subset V_3. An example of the swap move is shown in Fig. 2, where the solution on the right is obtained by swapping vertex $v \in V_3$ with vertex $u \in V_1$. The choice of move type at each iteration is controlled by a parameter $Q \in [0, 1]$, which defines the probability of selecting a swap move. Trivially, the relocation move is performed with the probability $1 - Q$. Assume that a vertex $v \in V_k$, $k \in M$, is relocated from its current subset V_k to subset V_l, $l \neq k$, of the partition $p = \{V_1, \ldots, V_m\} \in \Pi$. Let the resulting partition be denoted by p'. Naturally, it is assumed that v belongs to the set $V'(p)$, composed of taking the union of all subsets in p having cardinality greater than one. Thus, $v \in V_k \subseteq V'(p) = \cup_{i=1,|V_i|>1}^{m} V_i$. For $p \in \Pi$, we can define the relocation neighborhood $N_1(p)$ of p as the set of all partitions that can be obtained from p by relocating a single vertex. The change in cost incurred by applying a move is called the move gain. In the case of the above-defined relocation move, it is denoted by $\delta(p, p')$. Formally, $\delta(p, p') = F_r(p') - F_r(p)$. To provide an expression for δ, we rewrite the objective function of (1) as $F_r(p) = \sum_{i=1}^{m} R(V_i)$ where

$$R(V_i) = C(V_i)/|V_i|. \tag{3}$$

For a vertex $v \in V$ and a subset $V_i \in p$, let $c_v(V_i) = \sum_{u \in V_i} c_{vu}$. This number represents the sum of weights of edges between v and vertices in subset V_i. Of course, the sum $c_v(V_i)$ is also defined for $v \in V_i$. The sums $c_v(V_i)$ along with the cut weights $C(V_i)$ and ratios $R(V_i)$ can be used to compute δ as follows.

Proposition 1 *For a vertex $v \in V_k$ and a subset V_l, $l \neq k$, in the partition p,*

$$\begin{aligned}\delta(p, p') &= (C(V_k) - d_v + 2c_v(V_k))/(|V_k| - 1) \\ &+ (C(V_l) + d_v - 2c_v(V_l))/(|V_l| + 1) - R(V_k) - R(V_l).\end{aligned} \tag{4}$$

Proof Let us denote the cut weights for partition p' by $C'(V_i)$, $V_i \in p'$. Imagine that in the first step of the relocation operation, vertex v is temporarily removed from graph G (and thus out of V_k). Then, the weight of the cut between V_k and $V \setminus V_k$ decreases by $d_v - c_v(V_k)$ and that between V_l and $V \setminus V_l$ decreases by $c_v(V_l)$. In the second step, the vertex v is inserted into V_l. This increases the weight of the cut between

V_k and $V \setminus V_k$ by $c_v(V_k)$ and that between V_l and $V \setminus V_l$ by $d_v - c_v(V_l)$. Thus

$$C'(V_k) = C(V_k) - d_v + 2c_v(V_k) \tag{5}$$

and

$$C'(V_l) = C(V_l) + d_v - 2c_v(V_l). \tag{6}$$

Considering the fact that $C'(V_i) = C(V_i)$ for all $i \neq k, l$, we can write

$$\begin{aligned}\delta(p, p') = {} & C'(V_k)/(|V_k| - 1) + C'(V_l)/(|V_l| + 1) \\ & - R(V_k) - R(V_l).\end{aligned} \tag{7}$$

Substituting (5) and (6) into (7), we obtain (4). □

To illustrate the computation of δ by (4), we use Fig. 1 in which edge weights greater than one are indicated close to the edges. For $k = 1$ and $l = 3$, we have $C(V_k) = 3$, $C(V_l) = 3$, $c_v(V_k) = 2$, $c_v(V_l) = 2$, $d_v = 4$, $R(V_k) = 0.75$, and $R(V_l) = 1.5$. Putting these values in (4), we obtain $\delta(p, p') = (3-4+4)/3+(3+4-4)/3-0.75-1.5 = -0.25$.

Upon acceptance of the move, the cut weights of the subsets V_k and V_l are updated according to Equations (5) and (6): $C(V_k)$ is decreased by $d_v - 2c_v(V_k)$ and $C(V_l)$ is increased by $d_v - 2c_v(V_l)$. The values of $c_w(V_k)$ and $c_w(V_l)$, $w \in V \setminus \{v\}$, are updated by setting

$$\begin{aligned}c_w(V_k) &:= c_w(V_k) - c_{wv}, \\ c_w(V_l) &:= c_w(V_l) + c_{wv}.\end{aligned} \tag{8}$$

In the case of normalized cut graph partitioning, Equation (4) is slightly modified as follows:

$$\begin{aligned}\delta(p, p') = {} & (C(V_k) - d_v + 2c_v(V_k))/(d(V_k) - d_v) \\ & + (C(V_l) + d_v - 2c_v(V_l))/(d(V_l) + d_v) \\ & - C(V_k)/d(V_k) - C(V_l)/d(V_l).\end{aligned} \tag{9}$$

If the move is accepted, then d_v is subtracted from $d(V_k)$ and added to $d(V_l)$. Certainly, the operations defined by Equations (5), (6), and (8) are also applied.

Next we consider the swap move. Assume that a vertex $v \in V_k$ is interchanged with a vertex $u \in V_l, l \neq k$. As before, we denote the initial partition by p and the resulting partition by p'. All the partitions that can be obtained from p in this way form the swap neighborhood $N_2(p)$ of p. The gain of swap move is denoted by $\Delta(p, p')$. By definition, $\Delta(p, p') = F_r(p') - F_r(p)$. The following statement presents a formula for Δ.

Proposition 2 *For a vertex $v \in V_k$ and a vertex $u \in V_l$, $l \neq k$,*

$$\begin{aligned}\Delta(p, p') = {} & (C(V_k) + d_u - d_v + 2(c_v(V_k) \\ & - c_u(V_k) + c_{uv}))/|V_k| \\ & + (C(V_l) + d_v - d_u + 2(c_u(V_l) \\ & - c_v(V_l) + c_{uv}))/|V_l| \\ & - R(V_k) - R(V_l).\end{aligned} \tag{10}$$

Proof We split the swap operation into four steps. Assume first that vertex v is temporarily removed from graph G. This action decreases $C(V_k)$ by $d_v - c_v(V_k)$ and $C(V_l)$ by $c_v(V_l)$. In the second step, vertex u is removed from graph G. As a result, $C(V_k)$ is further decreased by $c_u(V_k) - c_{uv}$ and $C(V_l)$ by $d_u - c_{uv} - c_u(V_l)$. The graph is restored by first adding vertex v to subset V_l. This step increases $C(V_k)$ by $c_v(V_k)$. Since vertex u is removed from graph G, the current weighted degree of vertex v is $d_v - c_{uv}$, and the current sum of edge weights between v and V_l is $c_v(V_l) - c_{uv}$. Therefore, $C(V_l)$ increases by $d_v - c_{uv} - (c_v(V_l) - c_{uv}) = d_v - c_v(V_l)$. In the last step, the vertex u is inserted into V_k. Now, the sum of edge weights between u and V_k is $c_u(V_k) - c_{uv}$. It follows that $C(V_k)$ is increased by $d_u - (c_u(V_k) - c_{uv}) = d_u - c_u(V_k) + c_{uv}$. Since vertex v has been moved to subset V_l, it also follows that $C(V_l)$ is increased by $c_u(V_l) + c_{uv}$. Now, let $C'(V_i)$, $V_i \in p'$, stand for the cut weights for partition p'. Summing the changes in $C(V_k)$ for each step of the above procedure, we obtain

$$\begin{aligned}C'(V_k) = {} & C(V_k) - (d_v - c_v(V_k)) \\ & - (c_u(V_k) - c_{uv}) + c_v(V_k) \\ & + (d_u - c_u(V_k) + c_{uv}) = C(V_k) \\ & + d_u - d_v + 2(c_v(V_k) - c_u(V_k) + c_{uv}).\end{aligned} \tag{11}$$

Analogously

$$\begin{aligned}C'(V_l) = {} & C(V_l) - c_v(V_l) \\ & - (d_u - c_{uv} - c_u(V_l)) + (d_v - c_v(V_l)) + \\ & (c_u(V_l) + c_{uv}) = C(V_l) + d_v - d_u \\ & + 2(c_u(V_l) - c_v(V_l) + c_{uv}).\end{aligned} \tag{12}$$

Clearly, $C'(V_i) = C(V_i)$ for all $i \neq k, l$. Therefore, substituting (11) and (12) into equation $\Delta(p, p') = C'(V_k)/|V_k| + C'(V_l)/|V_l| - R(V_k) - R(V_l)$ gives (10). □

Figure 2 provides an example of the swap move. As before, only edge weights greater than 1 are shown. For $k = 3$ and $l = 1$, we have $C(V_k) = 5, C(V_l) = 5, c_v(V_k) = 1, c_v(V_l) = 3, c_u(V_k) = 2, c_u(V_l) = 1, d_v = 5, d_u = 3$, and $R(V_k) = R(V_l) = 5/3$. Then, using (10), we can calculate the swap move gain $\Delta(p, p')$, which is equal to $(5+3-5+2(1-2+1))/3 + (5+5-3+2(1-3+1))/3 - 5/3 - 5/3 = -2/3$.

If the solution p' is accepted, then first, the cut weights of the subsets V_k and V_l are updated using Equations (11) and (12) (assuming that $C'(V_k)$ and $C'(V_l)$ represent new values of $C(V_k)$ and $C(V_l)$, respectively). Then, since vertices v and u are swapped, the following assignments are performed:

$$c_v(V_k) := c_v(V_k) + c_{uv}, c_v(V_l) := c_v(V_l) - c_{uv}, \tag{13}$$

$$c_u(V_k) := c_u(V_k) - c_{uv}, c_u(V_l) := c_u(V_l) + c_{uv}, \tag{14}$$

$$\begin{aligned} &c_w(V_k) := c_w(V_k) + c_{uw} - c_{vw}, \\ &c_w(V_l) := c_w(V_l) + c_{vw} - c_{uw}, \\ &w \in V \setminus \{u, v\}. \end{aligned} \tag{15}$$

For a normalized cut, the swap move gain is computed as follows:

$$\begin{aligned} \Delta(p, p') = &(C(V_k) + d_u - d_v + 2(c_v(V_k) - c_u(V_k) \\ &+ c_{uv}))/(d(V_k) - d_v + d_u) \\ &+ (C(V_l) + d_v - d_u + 2(c_u(V_l) - c_v(V_l) \\ &+ c_{uv}))/(d(V_l) + d_v - d_u) \\ &- C(V_k)/d(V_k) - C(V_l)/d(V_l). \end{aligned} \tag{16}$$

Equations (13)–(15) are applicable for both objective functions, F_r and F_n.

Before presenting the MSA algorithm, we need to define a mapping, denoted by q, from vertex set V to set M, which represents the indices of the partition subsets. We will write $q(v) = k$ to indicate that vertex v belongs to subset V_k of partition p. The pseudocode of MSA is given in Algorithm 1. In addition to the aforementioned move type selection probability Q, the parameters of MSA are the cooling factor α, the final temperature $T_{\min}$, and the number of moves, β, to be attempted at each temperature level. The initial temperature $T_{\max}$ is computed using the formula $T_{\max} = \max(\max_{p' \in N_1'} |\delta(p, p')|, \max_{p' \in N_2'} |\Delta(p, p')|)$, where p is a starting partition generated in Line 2 of Algorithm 1 and N_1' (N_2') is a sequence of partitions obtained by applying randomly chosen relocation (respectively, swap) moves to the partition p (thus N_1' and N_2' are extracted from the neighborhoods $N_1(p)$ and $N_2(p)$, respectively). We fixed the length of both sequences at 5, 000. Having the initial and final temperatures of the annealing process, we can calculate the number of temperature levels $\bar{\gamma}$ (Line 5).

As seen in the pseudocode, our SA algorithm is implemented as a multistart procedure. The reason for such a choice is to fairly compare the algorithm with other presented algorithms using a time-based stopping criterion. Obviously, there is no efficient way to control the number of SA restarts. This number decreases with increasing value of the parameter β. Usually, β is assumed to be dependent on the size of the problem instance. Our multistart mechanism is simple-minded. At the start of each iteration, a random partition of the graph is generated. This is done by first randomly generating a permutation of the vertices of G and then splitting its entries (vertices) into m subsets of approximately equal sizes. The latter operation assigns the first $\lceil n/m \rceil$ vertices in the permutation to the first subset, the next $\lceil n/m \rceil$ (or $\lfloor n/m \rfloor$) vertices to the second subset and so on. The obtained partition of the graph is passed to the SA method. Additional operations must be performed before entering SA for the first time. They include initialization of the best solution p^* and calculation of the initial temperature $T_{\max}$. This temperature is reused in subsequent SA restarts. It can be noted that SA with multiple starts requires no additional parameters beyond those of classical SA or the probability parameter Q.

From the pseudocode, we see that SA is wrapped in the "while" statement, which loops until a stop condition is met. We use a stopping criterion, where the search is terminated when a maximum time limit is reached. At the beginning of the iteration, the values of the sums $c_w(V_i)$, $w \in V$, $i \in M$,

Algorithm 1 Multistart simulated annealing.

```
    MSA
1:  while stop condition is not met do
2:    Generate a partition p ∈ Π randomly and set f := F_r(p)
3:    if first execution of the "while" loop then
4:      Assign p to p* and f to f*
5:      Compute T_max and γ̄ = ⌊(log(T_min) − log(T_max))/ log α⌋
6:    end if
7:    T := T_max
8:    for γ = 1, ..., γ̄ do
9:      for i = 1, ..., β do
10:       if random(0, 1)> Q then // relocation move
11:         Randomly select a vertex v ∈ V'(p) and a subset V_l, l ≠ k = q(v)
12:         Compute δ := δ(p, p') by (4) where p' is the solution defined by
    v and l
13:       else // swap move
14:         Randomly select vertices v, u ∈ V such that k = q(v) ≠ l = q(u)
15:         Compute δ := Δ(p, p') by (10) where p' is the solution defined
    by v and u
16:       end if
17:       if δ ≤ 0 or exp(−δ/T) ≥ random(0, 1) then
18:         f := f + δ
19:         if relocation move has been selected then
20:           Update C(V_k), C(V_l), and c_w(V_k), c_w(V_l) for all w ≠ v by (5),
    (6), and (8)
21:           Move v from V_k to V_l // partition p and mapping q are updated
22:         else // swap move
23:           Update C(V_k) and C(V_l) by (11) and (12)
24:           Update c_w(V_k) and c_w(V_l) for all w ∈ V by (13)–(15)
25:           Swap the vertices v and u // partition p and mapping q are
    updated
26:         end if
27:         Compute R(V_k) and R(V_l) by (3)
28:         if f < f* then Assign p to p* and f to f* end if
29:       end if
30:     end for
31:     T := αT
32:   end for
33: end while
34: Stop with the solution p* of value f*
```

and the cut weights $C(V_i)$, $i \in M$, for the generated solution p are initialized. The best solution to the algorithm is denoted as p^* and its value is f^*. This solution is memorized in Line 28. At each iteration, either a relocation move (Lines 11 and 12) or a swap move (Lines 14 and 15) is selected and evaluated. This choice is controlled by the probability parameter Q. If the move is accepted, then the current solution p and related information are updated in Lines 18–27. The temperature is changed after performing β iterations. As is a common practice in SA implementations, the temperature is decreased according to the geometric schedule (Line 31). We note that in the normalized cut case, Equations (4) and (10) should be replaced with Equations (9) and (16), respectively (Lines 12 and 15). Considering only the innermost loop in Algorithm 1, we can obtain the following statement.

Proposition 3 *The computational complexity of the body of loop 9–30 of MSA is* $O(n)$.

Proof The major bottleneck of computations in Lines 10–29 of MSA is updating $c_w(V_k)$, $c_w(V_l)$, $w \in V$ (either in Line 20 or in Line 24). This clearly takes $O(n)$ time. The same upper bound holds on the number of operations required to save the improved solution when such a solution is found (Line 28). Other steps inside the loop are less time-consuming. In particular, only $O(1)$ time is needed to compute the gain of a move (Lines 12 and 15). □

We remark that the parameter β in many SA implementations depends linearly on the size of the problem. This is also the case in our experiments with MSA (see Sect. 5.2). Therefore, all operations inside the loop 8–32 of MSA can be performed in $O(n^2)$ time. However, evaluating the time complexity of all executions of this loop, which implements SA, is difficult because the number of its repetitions depends on the initial temperature $T_{\max}$. In our algorithm, this temperature is a quantity dependent on the character of the problem instance being solved.

3 Iterated tabu search

A possible alternative to the SA technique is the widely used TS method [24]. We apply it to minimize the ratio cut and normalized cut objective functions iteratively.

The main idea of our TS implementation is to repeatedly execute tabu search and solution perturbation procedures. The use of such a strategy implies that it is only the first iteration where TS starts with a fully random solution. The solution perturbation procedure starts with a relatively good solution and proceeds by performing a number of random moves. The resulting solution is passed to the TS component of the approach. The key idea of tabu search is to allow accepting worsening solutions to prevent getting stuck in local minima. To avoid cycling in the search process, for each move performed, the reverse move is forbidden for a specified number of iterations. For this purpose, we use two tabu tables, one for relocation moves and another for swap moves. The TS procedure stops when a predefined number of iterations is reached. This number is an important parameter of the algorithm. A noteworthy feature of our approach is that the TS procedure is enhanced with the integration of an LS technique. Each of them explores both the relocation and swap neighborhoods at each iteration.

As in the case of SA, we present the ITS algorithm for the ratio cut partitioning problem. The pseudocode of the top level of our TS implementation is shown in Algorithm 2. The search starts from a solution generated using the same procedure as for the MSA method. The best solution is denoted as p^* and its value is f^*. The TS algorithm and solution perturbation procedure, named `get_restart_partition`, are executed repeatedly within the loop 3–6. To stop the loop, a CPU time-based termination condition is used.

Algorithm 2 Iterated tabu search.

```
   ITS
1: Randomly generate a partition p ∈ Π and set f := F_r(p)
2: Assign p to p* and f to f*
3: while stop condition is not met do
4:    Apply the tabu search procedure TS(p, f, p*, f*)
5:    p :=get_restart_partition(p, f)
6: end while
7: Stop with the solution p* of value f*
```

Algorithm 3 gives the pseudocode of the TS component of the approach. This component accepts and returns the current partition p, the best-found partition p^*, and their objective function values f and f^*. It starts with the initialization of the tabu tables $t = (t_{vu})$ and $\tilde{t} = (\tilde{t}_{vl})$ by setting their entries to $-\tau$, where τ is the tabu tenure parameter. As the TS algorithm proceeds, the entry t_{vu} of the tabu table t is used to store the last iteration number in which the vertices v and u have been swapped. Similarly, $\tilde{t}_{vl}$ contains the last iteration number in which vertex v has been moved from partition subset V_l to a different subset. Another parameter of the algorithm is the number of iterations I. In each iteration, both neighborhood N_1 and neighborhood N_2 are explored. The selected move is specified by the vertex v^* and partition subset V_{l^*} in the case of relocation and by the vertices v^* and u^* in the case of the swap operation. To distinguish which move type is selected, we use the flag variable ξ, which equals 0 for relocation and 1 for swap move. The gain incurred by the selected move is denoted by $\delta_{\min}$. In the gain calculation, p is the current solution, and p' is either the solution obtained by relocating vertex v from subset V_k to subset V_l (Line 7) or the solution obtained by swapping vertices v and u

Algorithm 3 Tabu search.

TS(p, f, p^*, f^*)

1: Set $t_{vu} := -\tau$ for all $v, u \in V$, $v \neq u$, and $\tilde{t}_{vl} := -\tau$ for all $v \in V$ and $l \in M$
2: **for** $i = 1, \ldots, I$ **do**
3: Set $\delta_{\min} := \infty$, $\rho := 0$, and $U := V$
4: **for** each vertex v **do**
5: $U := U \setminus \{v\}$
6: **for** each l, $l \neq k = q(v)$, in the case of $v \in V'(p)$ **do**
7: Compute $\delta := \delta(p, p')$ by (4) where p' is the solution defined by v and l
8: **if** $f + \delta < f^*$ **then**
9: $\rho := \rho + 1$
10: With probability $1/\rho$ set $\delta_{\min} := \delta$, $v^* := v$, $l^* := l$, and $\xi := 0$
11: **else if** $i - \tilde{t}_{vl} > \tau$ and $\delta < \delta_{\min}$ **then**
12: Set $\delta_{\min} := \delta$, $v^* := v$, $l^* := l$, and $\xi := 0$
13: **end if**
14: **end for**
15: **for** each vertex $u \in U$ such that $l = q(u) \neq k = q(v)$ **do**
16: Compute $\delta := \Delta(p, p')$ by (10) where p' is the solution defined by v and u
17: **if** $f + \delta < f^*$ **then**
18: $\rho := \rho + 1$
19: With probability $1/\rho$ set $\delta_{\min} := \delta$, $v^* := v$, $u^* := u$, and $\xi := 1$
20: **else if** $i - t_{vu} > \tau$ and $\delta < \delta_{\min}$ **then**
21: Set $\delta_{\min} := \delta$, $v^* := v$, $u^* := u$, and $\xi := 1$
22: **end if**
23: **end for**
24: **end for**
25: **if** $\xi = 0$ **then** $\tilde{t}_{v^* q(v^*)} := i$ **else** $t_{v^* u^*} := i$ **end if**
26: Perform relocation (if $\xi = 0$) or swap (if $\xi = 1$) move // as in Lines 19–27 of MSA
27: $f := f + \delta_{\min}$
28: **if** $\rho > 0$ **then**
29: $p :=$`local_search`(p, f)
30: Assign p to p^* and f to f^*
31: **end if**
32: **end for**
33: **return** current partition p, its value f, and possibly updated p^* and f^*

the current partition p is updated. If an improved solution has been found ($\rho > 0$), then the TS algorithm attempts to further improve this solution by applying a local search procedure (Line 29). A description of this procedure is given later in this section.

Looking at the TS pseudocode, we can see that the statements inside the outermost loop can be split into two sequences: those in Lines 3–27 and those in Lines 28–31. The first of these sequences has a worst-case runtime $O(n^2)$. The second of these sequences includes a call to the LS procedure. However, its worst-case time complexity is unknown. A remark on the computational complexity of the basic parts of `local_search` is given at the end of this section. To periodically direct the search toward unexplored regions

Algorithm 4 Solution perturbation.

`get_restart_partition`(p, f)

1: Generate the values of the solution perturbation parameters K and L
2: $U := \emptyset$ // initially U is an empty set
3: **repeat** K times
4: Select L pairs (v, l), $v \in V'(p) \setminus U$, $l \in M$, $l \neq q(v)$, with smallest δ computed by (4)
5: Randomly choose a pair, say (v^*, l^*), from the list constructed in Step 4
6: $f := f + \delta(p, p')$, where p' is the solution defined by v^* and l^*
7: Update p by performing relocation move // as in Lines 20, 21, and 27 of MSA
8: $U := U \cup \{v^*\}$
9: **end repeat**
10: **return** partition p and its value f

(Line 16). The variable ρ is used to know whether an improving solution has been found. If this is the case, ρ is positive. If two or more improving solutions are discovered, then one of them (or, more precisely, the corresponding move) is selected probabilistically (Lines 10 and 19). The role of the set U in the algorithm (Lines 3, 5, and 15) is to guarantee that each pair of vertices (v, u) is considered only once. The algorithm also uses the set V' and mapping q, which we defined in the previous section.

Once neighborhoods N_1 and N_2 of the current solution p have been explored, the algorithm proceeds by updating the tabu tables t and $\tilde{t}$ (Line 25) and performing either the relocation or swap operation, depending on the value of ξ (Line 26). These operations are implemented in the same way as those in MSA (Lines 19–27 of Algorithm 1). Specifically, Equations (5), (6), and (8) are used for $v = v^*$, $k = q(v^*)$, and $l = l^*$ in the case of the relocation move, and Equations (11)–(15) are used for $v = v^*$, $u = u^*$, $k = q(v^*)$, and $l = q(u^*)$ if the swap move has been selected. In the same step, of the solution space, the ITS algorithm applies a solution perturbation procedure `get_restart_partition`. Its pseudocode is presented in Algorithm 4. The input to the procedure includes a partition p, which is the last solution recorded by the TS component of the approach. At the initialization step, `get_restart_partition` first draws uniformly at random an integer $K_{\max}$ from the interval $[n\kappa_1, n\kappa_2]$ and then integers K from the interval $[K_{\min}, K_{\max}]$ and L from the interval $[L_{\min}, L_{\max}]$. In these calculations, κ_1, κ_2, $K_{\min}$, $L_{\min}$, and $L_{\max}$ are ITS parameters. The first two parameters, that is, κ_1 and κ_2 define the range for the maximum number of relocation moves to be performed. To bound this number from below, the parameter $K_{\min}$ is used. At each iteration of the solution perturbation procedure, a list of the best relocation moves is built. The parameters $L_{\min}$ and $L_{\max}$ restrict the length of this list to be bounded from below and above, respectively. The role of all these ITS parameters is to guide the diversification of the search. Their appropriate values

are selected experimentally. The parameters K and L of the `get_restart_partition` procedure, when used together, control the level of degradation of the objective function value due to the perturbation of the current partition p. The quality of the generated partition deteriorates with the increase in K and L. In contrast, small values of K and L may lead to producing a partition that is too close to the input solution p. From the pseudocode, it should be clear that the use of the set U guarantees that each vertex is moved from one subset to another at most once. Provided L is a constant and K is proportional to n (such choices are made in Sect. 5.2), the computational complexity of the procedure `get_restart_partition` is $O(n^2m)$.

Algorithm 5 Local search.

```
    local_search(p, f)
1:  Set λ :=true and s_i := 1 for each i ∈ M
2:  while λ =true do
3:    λ :=false
4:    for each pair k ∈ M'(p) and l ∈ M \ {k} such that s_k > 0 or s_l > 0 do
5:      μ :=true
6:      while μ =true and |V_k| > 1 do
7:        μ :=false
8:        for each v ∈ V_k do
9:          Compute δ := δ(p, p') by (4) for p' defined by v and l
10:         if δ < 0 then Set μ :=true, v* := v and break from the loop end if
11:       end for
12:       if μ =true then
13:         λ :=true
14:         Perform relocation move for v* and l // as in Lines 20, 21, and 27 of
    MSA
15:         f := f + δ
16:         Set s_k := 2 and s_l := 2
17:       end if
18:     end while
19:   end for
20:   for each pair k, l ∈ M, l > k, such that s_k > 0 or s_l > 0 do
21:     μ :=true
22:     while μ =true do
23:       μ :=false
24:       for each v ∈ V_k and each u ∈ V_l do
25:         Compute δ := Δ(p, p') by (10) for p' defined by v and u
26:         if δ < 0 then Set μ :=true, v* := v, u* := u and break from the
    loop end if
27:       end for
28:       if μ =true then
29:         λ :=true
30:         Perform swap move for v* and u* // as in Lines 23–25 and 27 of
    MSA
31:         f := f + δ
32:         Set s_k := 2 and s_l := 2
33:       end if
34:     end while
35:   end for
36:   if λ =true then Set s_i := s_i − 1 for each s_i > 0, i ∈ M end if
37: end while
38: return partition p and its value f
```

We end this section by describing our implementation of the LS algorithm for the considered graph partitioning problems. As seen before, this algorithm is employed within the TS framework. However, perhaps more importantly, the LS procedure is the key ingredient of an MA, which is presented in the next section. The pseudocode of the LS procedure is given in Algorithm 5. As is typical in most LS implementations, the procedure executes in a number of iterations. Each iteration performs all the operations contained in the outer "while" loop (Lines 3–36). For a partition p at the beginning of the loop, let us consider a partition subset V_k, $k \in M$. Assume that the content of V_k does not change during the execution of the iteration. More precisely, no vertex $v \in V_k$ moves to another subset (Line 14), and no vertex $u \in V \setminus V_k$ is added to V_k (Line 14 when $l = k$). Moreover, no vertex $v \in V_k$ is interchanged with a vertex $u \in V \setminus V_k$ (Line 30). In this case, the entry s_k of the vector $S = (s_1, \ldots, s_m)$ at the end of the iteration (Line 36) is equal to 0. If at least one of the above conditions is not satisfied, then $s_k = 1$. Within the loop, some entries of S may temporarily be set to 2. Assume now that $s_k = 0$ and $s_l = 0$ for subsets V_k and V_l. Then, in the next iteration, all the gains δ and Δ expressed by (4) and (10) for V_k and V_l, respectively, have nonnegative values. It follows that, in the next iteration, there is no need to examine moves in which both subsets, V_k and V_l, are involved. This strategy implemented in the `local_search` procedure reduces the computational time needed to reach a locally optimal solution. The same technique oriented at accelerating neighborhood examinations was proposed by Lai et al. [37] for the MDGP. These authors, however, considered relocation moves and swap moves separately. They used two $0-1$ matrices of size $m \times m$. The (i, j)-entry of the first matrix takes value 0 if and only if no relocation of a vertex from the ith group to the jth group resulted in obtaining an improving solution. Similarly, the second matrix is defined with respect to the swap operation. Lai et al. [37] provided computational results that demonstrated the unequivocal efficiency of the proposed neighborhood exploration strategy.

As can be seen in Algorithm 5, each iteration of our LS procedure is composed of two phases. In the first of them (Lines 4–19), the neighborhood N_1 of the current partition p is explored. In Line 4, $M'(p)$ refers to the set of subsets in partition p whose size is greater than 1. Formally, $M'(p) = \{k \mid k \in M, V_k \in p, |V_k| > 1\}$, where $M = \{1, \ldots, m\}$ as before. If an improving solution is found, then flag μ is set to true (Line 10). As a result, the current solution p and its value f are updated (Lines 14 and 15). In the second phase (Lines 20–35), a better-quality solution is searched for in neighborhood $N_2(p)$. If such a solution is found among the neighbors of p, then it is used to replace p (Lines 30 and 31).

It can be observed from the pseudocode that the time complexity of the body (Lines 7–17) of the first inner "while" loop is $O(n)$ and that of the body (Lines 23–33) of the second inner "while" loop is $O(n^2)$. In obtaining these estimates, it is assumed that in the worst case, the size of partition subsets

can be proportional to the graph order n. The number of times the "while" loops are executed depends on the graph and on the starting solution p. This number, however, is difficult to estimate.

4 A memetic algorithm

Evolutionary algorithms are an important class of approaches for solving various optimization problems. Among them, the memetic algorithm is one of the most successful techniques. The fundamental concept of the MA is to apply the genetic operators in combination with an LS procedure. In this section, we present an MA for ratio cut and normalized cut graph partitioning. As before, a description is given in terms of the first of these problems.

Like a GA, the MA manipulates a population of individuals where each individual represents a solution to the optimization problem being solved. To generate new members of the population, a crossover operator comes into play. Usually, it is a binary operator that combines the genes of two parents in some manner to produce an offspring. In the context of ratio cut and normalized cut, individuals in the population correspond to partitions of vertex set V. A convenient method for coding an individual is to use a mapping q introduced in Sect. 2. We remind that for a partition $p = \{V_1, \ldots, V_m\}$ and a vertex $v \in V_i$, the value $q(v) = i$ points to the partition subset to which vertex v belongs.

One of the key ideas of our proposed algorithm is to employ the crossover operator that is used in grouping genetic algorithms (GGAs) (see [31,54]). In a GGA, the chromosomes represent the allocation of certain objects (e.g., vertices of the graph) to groups. When generating offspring from two parents, a GGA manipulates groups instead of group members. Our crossover operator randomly selects two individuals as parents and repeatedly transfers a subset of vertices from one of the selected parents to offspring. If the resulting collection of subsets does not cover all the vertices of the graph, then a repair mechanism is triggered to obtain a feasible partition. The offspring partition replaces the worst individual in the current population if it is not worse than the worst individual and of course differs from all individuals in the population. Another component of our memetic algorithm is the LS procedure, which is precisely the same as that used in the ITS algorithm. Certainly, this procedure is applied to each generated offspring. Moreover, LS is used to improve randomly generated graph partitions when constructing an initial population of solutions.

Let p' and p'' be two partitions in Π that are submitted as an input to the crossover operator. The pseudocode of our GGA crossover implementation is given in Algorithm 6, where r, r', and r'' are the current number of subsets in the offspring partition p and parent partitions p' and p'', respectively, and where the resulting offspring is represented by the mapping q. The most important part of the procedure (the "while" loop spanning Lines 3–13) serves to transfer genetic information from the parents to the offspring. First, it equiprobably selects one of the parents and then finds a partition subset of this parent with the smallest value of the ratio R (given by (3)). Ties among subsets are broken by random selection. The chosen subset, V_k, is moved from the parent partition p' (or p'') to the offspring partition (Lines 7 and 8 in the case of p'). Assume that the parent p' is involved in this operation. Then, each vertex of V_k is removed from the corresponding subset of the partition p'' (Line 9). The iteration terminates by deleting empty subsets of p'', if any, and updating their number r'' accordingly. If the parent p'' is selected, then similar operations with respect to p'' are performed (Line 11). Upon emptying both p' and p'' (then

Algorithm 6 Crossover procedure for ratio cut.

```
   get_offspring(p', p'')
// p' and p'' are parent partitions
1:  Initialize q(v) with 0 for each v ∈ V
2:  Set r := 0, r' := m, and r'' := m
3:  while r < m and r' + r'' > 0 do
4:    Increment r by 1
5:    if random(0, 1)< 0.5 or r'' = 0 and, in addition, r' > 0 then
6:      Select a subset V_k ∈ p' with R(V_k) = min_{V_i ∈ p'} R(V_i)
7:      Set q(v) := r for each v ∈ V_k
8:      Remove V_k from p' and decrement r' by 1
9:      Remove all vertices of V_k from p'', and possibly update r''
10:   else
11:     Perform operations in Lines 6–9 with roles of p' and p'' (and r' and r'')
  reversed
12:   end if
13: end while
14: if U := {v ∈ V | q(v) = 0} is nonempty then
15:   while U ≠ ∅ do
16:     Select a vertex v from the set U and remove it from U
17:     if r < m then
18:       Increment r by 1 and set q(v) := r
19:     else
20:       Randomly select a subset index k ∈ {1, ..., m}
21:       Set q(v) := k
22:     end if
23:   end while
24: end if
// Suppose that the mapping q defines a partition p = {V_1, ..., V_r}
25: if r = m then return p
26: Set M̃ := {1, ..., r} and compute Z(M̃) = (Σ_{i∈M̃} |V_i|)/|M̃| = n/r
27: while r < m do
28:   Select a subset V_k, k ∈ M̃, at random from those satisfying
  |V_i| ≥ max(2, Z(M̃))
29:   if V_k is selected in Line 28 then
30:     Increment r by 1
31:     Remove k from M̃ and update Z(M̃) accordingly
32:     Randomly split V_k into two subsets of nearly equal size: V_r and new V_k
33:   else // No subset V_i, i ∈ M̃, with |V_i| ≥ max(2, Z(M̃)) exists
34:     Reset M̃ to {1, ..., r} and compute Z(M̃) as in Line 26
35:   end if
36: end while
37: return p
```

$r' = r'' = 0$), there may still be some vertices uncovered by offspring subsets. The set of such vertices in the pseudocode is denoted by U. If $U \neq \emptyset$ and $r < m$, then the priority is to create single-vertex subsets (Lines 17, 18). When r reaches m, every remaining vertex of U is assigned to a randomly chosen partition subset (Lines 20, 21). The resulting partition p completely covers the vertices of the graph. However, it may occur that p consists of an insufficient number of subsets. In this case, to repair the partition p, an additional "while" loop is used (Lines 27–36). In this part of the pseudocode of the algorithm, $\tilde{M}$ is the set of indices of the partition subsets that are used as potential candidates for splitting, and $Z(\tilde{M})$ denotes the average size of these subsets. Before entering the loop, $Z(\tilde{M}) = n/r$ because initially $\cup_{i \in \tilde{M}} V_i = V$ (Line 26). Assume that $r < m$ at this point. Then, the partition p is refined by applying the subset splitting operation $m - r$ times. Only subsets of size at least $Z(\tilde{M})$ (and trivially at least 2) are candidates for splitting. Each iteration starts by randomly selecting one of the subsets (Line 28). The chosen subset is randomly split into two subsets of nearly equal size (Line 32). Its index is removed from $\tilde{M}$, and the new value of $Z(\tilde{M})$ is calculated (Line 31). If no suitable subset is found, then a new round of splitting begins with an enlarged set $\tilde{M}$ (Line 34). Concluding the description of the crossover operator, we remark that an offspring can be generated efficiently.

Proposition 4 *The computational complexity of the procedure* `get_offspring` *is* $O(n^2)$.

Proof Observe that `get_offspring` performs $O(n^2)$ operations to compute the ratios $R(V_i)$, $i \in M$. This can be performed at the initialization stage of the procedure. Other parts of the crossover have less complexity. In particular, all iterations of the "while" loops 3–13 and 27–36 take $O(nm)$ time. □

Algorithm 7 Memetic algorithm.

```
    MA
 1: f* := ∞
 2: P := init_population(p*, f*)
 3: while stop condition is not met do
 4:    Randomly choose two individuals, p' and p'', from the population P
 5:    p :=get_offspring(p', p'')
 6:    f := F_r(p)
 7:    p :=local_search(p, f)
 8:    P :=evaluate_offspring(P, p, f, p*, f*)
 9: end while
10: Stop with the solution p* of value f*
```

The pseudocode of the top level of the MA for ratio cut graph partitioning is presented in Algorithm 7. After constructing an initial population P (Line 2), the algorithm iterates over the following four steps until a termination condition is satisfied: selection of a pair of individuals in the current generation as parents (Line 4), crossover of the parents, generation of an offspring p (Line 5), execution of an LS algorithm on the offspring p (Line 7), and evaluation of p (Line 8). Similar to MSA and ITS, the iterations stop when a maximum time limit is reached. The best solution in MA is denoted as p^* and its value is f^*. MA

Algorithm 8 Generating the initial population.

```
    init_population(p*, f*)
// P is population and z is its size
 1: P := ∅
 2: while |P| < z do
 3:    Generate a partition p̃ ∈ Π at random and set f := F_r(p̃)
 4:    p :=local_search(p̃, f)
 5:    if f < f* then Assign p to p* and f to f* end if
 6:    if p differs from each p' ∈ P then
 7:       Include p into P
 8:    else if p̃ differs from each p' ∈ P then
 9:       Include p̃ into P
10:    end if
11: end while
12: return P
```

is based on the application of four procedures. Two were previously described: `get_offspring` previously in this section and `local_search` in Sect. 3. The pseudocode of `init_population` is given in Algorithm 8. This procedure creates an initial population P of size z, where z is a parameter of MA. In the pseudocode, p^* is the best solution in the population, and f^* is its objective function value. At each iteration, the candidates for inclusion in P are a partition $\tilde{p} \in \Pi$ generated at random using the same routine as for MSA and a locally optimal solution p produced by `local_search` applied to $\tilde{p}$. A priority is assigned to partition p. If p differs from all solutions currently in P, then it is appended to the population. Otherwise, an attempt is made to add the partition $\tilde{p}$ to P.

The condition in Line 6 of Algorithm 8 is checked using an $m \times m$ matrix, $H = (h_{ij})$, whose entry h_{ij} is the number of vertices $v \in V$ such that $q(v) = i$ and $q'(v) = j$, where q and q' are the mappings corresponding to partitions p and p', respectively. The following statement demonstrates that this condition as well as that in Line 8 can be checked rather easily.

Proposition 5 *The computational complexity of checking whether partitions $p \in \Pi$ and $p' \in \Pi$ are different is $O(n)$.*

Proof Clearly, constructing the matrix H takes only $O(n)$ operations. The partitions p and p' are different if and only if the matrix H constructed for them has more than m nonzero entries. Taken together, these two observations prove the claim. □

Algorithm 9 gives the pseudocode of the procedure `evaluate_offspring`. It attempts to replace the worst individual in population P with generated offspring p. This

Algorithm 9 Updating the population.

```
evaluate_offspring(P, p, f, p*, f*)
if f < f* then
  Assign p to p* and f to f*
  η :=true
else
  if p differs from each p' ∈ P and f ≤ max_{p'∈P} F_r(p') then η :=true
  else η :=false end if
end if
if η =true then
  Replace the worst individual in P by p
end if
return P
```

procedure is also responsible for memorizing the best solution found.

5 Computational experiments

The purpose of this section is to examine the computational performance of the described algorithms for ratio cut and normalized cut graph partitioning. We present comparative experiments on both randomly generated graphs and well-known benchmark graphs taken from the literature.

5.1 Experimental setup

All the algorithms were coded in the C++ programming language, and the tests were carried out on a laptop with an Intel Core i5-6200U CPU running at 2.30 GHz. We remark that the code is designed to be applicable for graphs of arbitrary density. To achieve this requirement, the graph is represented in a computer's memory by the edge-weight matrix. This puts a limit on the size of graphs the code can deal with. Our computer can handle graphs with up to approximately 10^4 vertices.

We performed our experiments on the following two sets of graphs:

(a) complete undirected graphs with edge weights generated in the following two steps. First, each vertex is assigned a point whose coordinates are sampled randomly and uniformly from a rectangle. Then, for each pair of vertices $u, v \in V$, the weight of the edge $(u, v) \in E$ is computed as $c_{uv} = \min(1/d_{uv}, 100)$, where d_{uv} is the Euclidean distance between points corresponding to the vertices u and v. The ensemble of graphs generated in this way is split into five subsets according to the number of vertices, which is 200, 500, 1000, 2000, and 3000, respectively. Each subset consists of five graphs.

(b) A collection of 36 benchmark graphs. It consists of the following three sets: the 10th DIMACS Implementation Challenge Benchmark [2] (first 18 graphs in the tables to follow, that is, from `karate` to `add32`, inclusively), 12 graphs from the Network Data Repository [56] (from `Trefethen-200` to `bio-dmela` in the tables), and 6 social network samples [9] (last 6 graphs in the tables, that is, from `soc52` to `pokec_2000`). We remark that the graph named `netscience` has more than 100 isolated vertices. We deleted them, reducing the size of `netscience` from 1589 to 1461.

The dataset (a) and the source codes of the presented algorithms are publicly available at http://www.personalas.ktu.lt/~ginpalu/grpart.html.

In our computational experiments, we run each algorithm 10 times on each graph in sets (a) and (b). Maximum CPU time limits for a run of an algorithm were as follows: 30 s for $n \leq 200$, 200 s for $200 < n \leq 500$, 500 s for $500 < n \leq 1000$, 1000 s for $1000 < n \leq 2000$, 2000 s for $2000 < n \leq 4000$, 4000 s for $4000 < n \leq 6000$, and 8,000 s for $n > 6000$. To assess the performance of the algorithms, we use the following measures: the objective function value of the best solution out of 10 runs, the average objective function value of 10 solutions, and the average time taken to find the best solution in a run.

5.2 Parameter settings

The main control parameters of the MSA algorithm are the cooling factor α, the final temperature $T_{\min}$, and the number of iterations at each temperature level, β. Following recommendations from the SA literature [57,62], we set α to 0.95, $T_{\min}$ to 0.0001, and β to $100n$. The specific parameter of our implementation of SA is the move type selection probability Q. We tested values of Q from 0 to 1 in increments of 0.1. To this end, we run MSA on a training sample consisting of 10 complete graphs whose edge weights are generated using the same procedure as for set (a). The five graphs in this sample are of order $n = 200$ and the other of order $n = 500$. The number of partition subsets is 5 for $n = 200$ and 10 for $n = 500$. The sample is disjoint from set (a), which is reserved for the testing stage. The performance of the MSA configurations was measured in terms of the average objective function value over 10 runs. The experiment in the case of ratio cut has shown that our MSA algorithm is quite robust to variation in the parameter Q in the interval [0, 0.9]. We also found that its solutions are substantially worse for $Q = 1$. This implies that using only swap operations is not a good strategy. Restricting to the interval [0, 0.9], a marginally better performance was observed for $Q = 0.1$. Therefore, we set Q to 0.1. A similar experiment was conducted with MSA for the case of normalized cut graph partitioning. Analogous conclusions as in the previous experiment were made. Based on the results obtained, we decided, for the normalized cut case, to set Q to 0.2.

The parameters of our ITS algorithm are τ, κ_1, κ_2, $K_{\min}$, $L_{\min}$, $L_{\max}$, and I (see Sect. 3). To determine good values for these parameters, we experimented with the same training set of graphs as used for the MSA algorithm. We relied on a simple parameter setting procedure. Its idea consists of allowing one parameter to take a number of predefined values while keeping the other parameters fixed at reasonable values chosen during preliminary tests. We ran ITS for both problems, ratio cut and normalized cut minimization. Because of their close similarity, it was possible to identify a common set of good parameter values for both problems. First, we varied κ_1 from 0.1 to 0.9 in increments of 0.1. The related parameter κ_2 ($> \kappa_1$) was fixed at 1, which is the maximum possible value. The range of acceptable values for κ_1 was found to be 0.5 to 0.8. We fixed κ_1 at 0.7. Then, we attempted to decrease the value of κ_2. Respecting condition $\kappa_2 > \kappa_1$, we tested two values of κ_2: $\kappa_2 = 0.8$ and $\kappa_2 = 0.9$. However, none of them led to better results. Therefore, we fixed κ_2 at 1 for all further experiments with ITS. We continued by examining the following 6 values of the parameter $K_{\min}$: 1, 5, 10, 15, 20, and 50. The results showed little sensitivity of ITS to this parameter. We arbitrarily set $K_{\min}$ to 15. The next step was to analyze the effect of the parameter $L_{\min}$. We ran ITS with $L_{\min} \in \{3, 5, 10, 25, 50, 100, 200\}$. We observed that the algorithm was fairly robust to the choice of $L_{\min}$. Slightly better results were obtained with smaller values of $L_{\min}$. Based on this finding, we set $L_{\min} = 10$. In the next experiment, we tried the following values of the parameter $L_{\max}$: 10, 25, 50, 100, 200, 300, 500, 700, 1000, 1500, 2000, and 3000. The ITS algorithm showed consistently good performance when $L_{\max}$ varied from 200 to 2000. Its performance deteriorated for $L_{\max} \leq 100$ and $L_{\max} = 3000$. We decided to fix $L_{\max}$ at 500. Further, we investigated how the performance of ITS depends on the tabu tenure parameter τ. We varied τ from 5 to 30 with a step of 5. Additionally, we tested $\tau = 3$. The range of acceptable values for τ was found to be quite wide. The results of very similar quality were obtained for $\tau \in \{10, \ldots, 30\}$. We fixed τ at 20, which was the middle value in this set. Perhaps a more important parameter of TS in the ITS framework is the number of iterations I. We ran ITS for the following values of I: 20, 50, 100, 200, 400, and 600. The best performance of ITS was observed when using $I \in \{100, 200\}$, with a slight edge to $I = 100$. The algorithm showed the worst performance for $I = 600$, especially for $I = 20$. Considering these findings, we elected to set I to 100.

The only parameter of our memetic algorithm is the population size z. We performed trial runs of MA with values of z up to 500 individuals. Based on the results obtained, we fixed z at 100. The performance of MA decreased when using smaller values of z. By increasing z above 100, the results on the training set of graphs appeared to be equally good or even marginally better than in the case of $z = 100$. However, MA with such z values becomes unduly time-consuming when used to partition large graphs. This comes from the fact that each individual of the initial population is submitted to an LS procedure that, for large graphs, takes a significant amount of time. Thus, the population size in our memetic algorithm should not be too large, and $z = 100$ is a good choice.

Table 1 Number of partition subsets (m) and time limit (in seconds) for random graphs

Graph series	n	m	Time limit (s)
g-200	200	5	30
g-500	500	10	200
g-1000	1000	20	500
g-2000	2000	30	1000
g-3000	3000	50	2000

Table 2 Best results for ratio cut when running MSA, ITS, and MA on random graphs

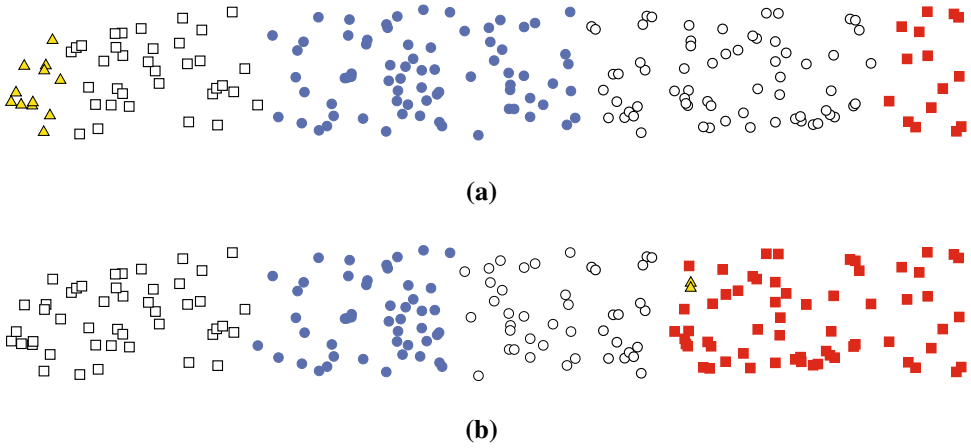
Graph	F_r^* MSA	ITS	MA
g-200-1	29.470787	**28.465644**	**28.465644**
g-200-2	**28.517328**	29.273550	**28.517328**
g-200-3	30.987977	**28.973014**	**28.973014**
g-200-4	28.018165	**27.991020**	**27.991020**
g-200-5	**27.843409**	**27.843409**	**27.843409**
g-500-1	198.782258	199.255728	**198.422620**
g-500-2	205.177224	205.732897	**204.587033**
g-500-3	**203.013723**	205.030860	**203.013723**
g-500-4	211.324003	210.961690	**209.366998**
g-500-5	205.164948	205.659258	**205.024331**
g-1000-1	1066.858248	1084.433932	**1061.061249**
g-1000-2	1066.365263	1065.563404	**1055.293605**
g-1000-3	**1074.052434**	1098.746424	1074.549342
g-1000-4	1088.758073	1097.070798	**1082.427574**
g-1000-5	1052.218926	1070.969257	**1047.958047**
g-2000-1	3555.965694	3568.355139	**3542.648687**
g-2000-2	3427.247307	3448.666324	**3414.023989**
g-2000-3	3575.194193	3606.018014	**3566.690424**
g-2000-4	3503.978136	3503.188732	**3480.601163**
g-2000-5	**3574.354453**	3627.354310	3582.175691
g-3000-1	10,177.017043	10,236.556683	**10,160.933638**
g-3000-2	9644.329837	9688.840004	**9617.682643**
g-3000-3	9654.802322	9712.334919	**9643.131826**
g-3000-4	9792.708644	9844.506723	**9786.830270**
g-3000-5	9509.276970	9557.378026	**9499.811186**

5.3 Numerical results for ratio cut

In this section, we present computational results obtained by the proposed algorithms for ratio cut graph partitioning. To

Table 3 Average results for ratio cut when running MSA, ITS, and MA on random graphs (the time is in seconds)

Graph	MSA		ITS		MA	
	$\overline{F_r}$	Time	$\overline{F_r}$	Time	$\overline{F_r}$	Time
g-200-1	29.568885	19	**28.465644**	3	**28.465644**	1
g-200-2	28.676491	15	29.338723	9	**28.517328**	1
g-200-3	31.036475	14	**28.991713**	11	29.009659	2
g-200-4	28.202671	16	28.097177	15	**28.018686**	6
g-200-5	28.526665	14	28.222911	14	**27.843409**	1
g-500-1	199.687609	88	200.862029	88	**198.491499**	56
g-500-2	206.432303	97	206.764527	56	**204.587033**	17
g-500-3	203.429266	97	207.039165	54	**203.087972**	40
g-500-4	212.618641	116	215.678167	59	**209.858584**	83
g-500-5	205.901594	99	207.778088	50	**205.328684**	101
g-1000-1	1069.421050	213	1097.856368	70	**1064.103136**	373
g-1000-2	1069.617529	295	1093.238921	108	**1057.419530**	346
g-1000-3	1077.223346	325	1116.316648	42	**1076.574194**	368
g-1000-4	1092.684641	342	1118.042031	101	**1084.293881**	397
g-1000-5	1062.036247	252	1084.078647	81	**1049.871139**	362
g-2000-1	3576.523899	547	3609.580853	143	**3545.146297**	959
g-2000-2	3462.714848	628	3504.891449	369	**3417.816873**	878
g-2000-3	3594.326927	559	3712.926018	172	**3571.157759**	915
g-2000-4	3513.369869	629	3552.292048	211	**3485.098691**	911
g-2000-5	3604.991310	602	3679.960011	187	**3587.270757**	869
g-3000-1	10,238.240934	1278	10,528.516808	666	**10,184.505301**	1823
g-3000-2	9671.068240	1153	9885.932732	979	**9628.423598**	1829
g-3000-3	9697.201817	1457	9864.712575	1106	**9654.822446**	1872
g-3000-4	9828.842172	1446	10,066.155128	849	**9799.888439**	1857
g-3000-5	9539.164250	1180	9720.000781	978	**9510.454537**	1899



(a)

(b)

Fig. 3 Best solutions for `g-200-5`. **a** Ratio cut ($F_r = 27.843409$, $F_n = 1.460579$). **b** Normalized cut ($F_r = 48.911503$, $F_n = 1.266439$)

perform pairwise comparisons between two algorithms, we apply the Wilcoxon signed-rank test.

We began our comparison by testing the performance of each of the algorithms on random graphs (set (a) in Sect. 5.1). The size of graphs, n, and the number of partition subsets, m, are shown in Table 1. Time limits (last column of the table) were provided in Sect. 5.1. Tables 2 and 3 summarize the results of the experiment, where the first column identifies the graph and the rest of the tables report the performance of MSA, ITS, and MA. Columns 2–4 of Table 2 contain the objective function value of the best solution out of 10 runs, denoted as F_r^*. The best value of F_r^* for each instance is highlighted in boldface. The two columns of Table 3 for each algorithm give the average objective function value of

Table 4 Summary of the results obtained by the tested algorithms for ratio cut on random graphs

Algorithm pair	OFV	#wins	#ties	#losses	p-value	Statistical significance
MA versus MSA	Best	20	3	2	< 0.001	Yes
	Average	25	0	0	< 0.001	Yes
MA versus ITS	Best	21	4	0	< 0.001	Yes
	Average	23	1	1	< 0.001	Yes
MSA versus ITS	Best	18	1	6	< 0.001	Yes
	Average	21	0	4	< 0.001	Yes

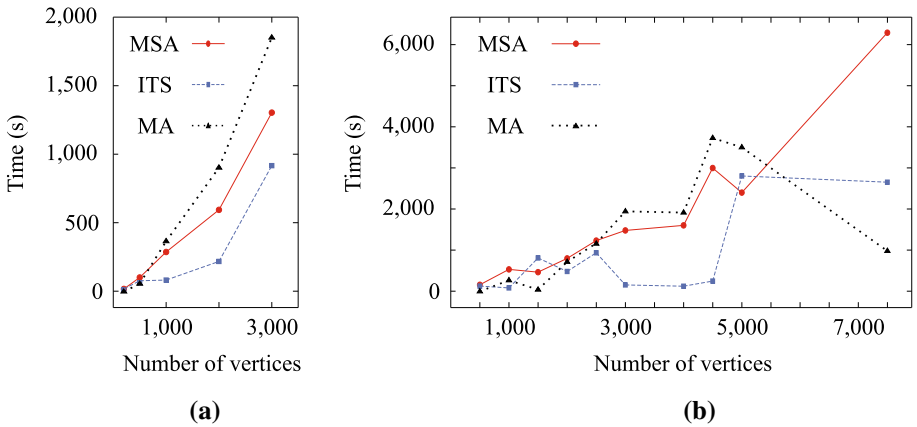


Fig. 4 Time taken by MSA, ITS, and MA to find the best solution in a run (the case of ratio cut). **a** Random graphs. **b** Benchmark graphs

10 solutions, denoted as $\bar{F}_r$, and the average time (in seconds) taken to reach the last improvement in solution quality. The best value of $\bar{F}_r$ among all the algorithms is indicated in boldface.

An example solution for a random graph is presented in Fig. 3a. It is the best solution for `g-200-5` obtained by the tested algorithms. It was found in two runs (out of 10) of MSA, two runs of ITS, and all 10 runs of MA.

Table 2 shows that MA performs considerably better than the other two algorithms. The MA obtained the best solution for more than 90% of the graphs (23 out of 25), whereas MSA and ITS produced the best result for 5 and 4 graphs, respectively. Comparing average values (Table 3), we observe even more prominent dominance of MA. This algorithm yielded the best results for 24 graphs. The average performance of the other two algorithms was much worse: ITS obtained the best average values for two graphs only, whereas MSA failed to do so for the entire set of graphs.

The results of the pairwise comparison of algorithms are shown in Table 4, where the names of the algorithms are given in the first column. The second column identifies the values being compared: "Best" denotes the values displayed in the columns in Table 2 labeled F_r^* and "Average" denotes those in the columns in Table 3 labeled $\bar{F}_r$. Columns 3 to 5 of Table 4 show comparison results: #wins, #ties, and #losses count the number of graphs on which the first algorithm in the pair finds a better, an equally good or an inferior solution than the second algorithm. Additionally, we applied the Wilcoxon signed-rank test for each pair of algorithms. The p-values from this test are estimated in the penultimate column. Using a standard significance level of 0.05, we found that the results of the first algorithm in the pair are better than those of the second algorithm (by placing "Yes" in the last column). Table 4 clearly shows that the MA is superior to the MSA technique and ITS method. Comparing the latter two approaches, an obvious advantage of MSA over ITS is observed.

Figure 4a shows the performance comparison of the tested algorithms in terms of computational time. Each point in the plot represents the average value taken over 5 graphs. The results indicate that the algorithms can be ranked, from fastest to slowest, in the following order: ITS, MSA, and MA.

In Tables 5 and 6, we computationally compare the algorithms on graphs in the (b) dataset. Their second and third columns give the size of the graph and the number of partition subsets, respectively. The labels of the other columns have the same meaning as in Tables 2 and 3. By analyzing the results in Tables 5 and 6, we find that MA performs much better than the other two algorithms. We observe that MA produced the best solution for 35 graphs (out of 36). The ITS and MSA algorithms obtained the best solutions for 12 and 9 benchmark graphs, respectively. Moreover, MA achieved the best average result for 35 graphs, whereas each of MSA and ITS did this only for 7 graphs. The only graph on which MA was defeated (by MSA) was `add20`. We also notice that Table 5 includes 7 graphs (`karate`, `dolphins`, `polbooks`, `football`, `can-292`, `ia-infect-dublin`, and `soc52`) for which

Table 5 Best results for ratio cut when running MSA, ITS, and MA on benchmark graphs from the literature

Graph	n	m	F_r^*		
			MSA	ITS	MA
karate	34	2	**0.937931**	**0.937931**	**0.937931**
dolphins	62	2	**0.432056**	**0.432056**	**0.432056**
polbooks	105	3	**2.295626**	**2.295626**	**2.295626**
football	115	12	**41.956854**	**41.956854**	**41.956854**
jazz	198	2	3.498814	**1.005076**	**1.005076**
celegansneural	297	20	37.192437	**31.111511**	**31.111511**
celegans_metabolic	453	3	0.918349	1.292873	**0.906726**
delaunay_n10	1024	20	18.019499	19.243098	**15.329365**
email	1133	2	0.550802	0.670216	**0.500884**
netscience	1461	12	0.138986	0.091430	**0.000000**
delaunay_n11	2048	30	28.493743	28.351407	**20.785773**
add20	2395	5	**0.768463**	0.827959	0.801684
data	2851	25	26.980359	28.331474	**18.579787**
delaunay_n12	4096	50	57.933976	44.455957	**33.371430**
3elt	4720	20	13.037060	9.615413	**5.407438**
uk	4824	25	11.161544	9.593982	**2.981080**
power	4941	20	5.585462	5.840868	**2.929816**
add32	4960	20	4.176777	4.714050	**3.336302**
Trefethen-200	200	6	29.302787	**29.177737**	**29.177737**
ash292	292	5	2.160565	2.128728	**2.108259**
can-292	292	4	**2.701870**	**2.701870**	**2.701870**
ia-infect-dublin	410	2	**0.515001**	**0.515001**	**0.515001**
dwt-503	503	3	**1.434769**	1.456562	**1.434769**
Trefethen-700	700	8	46.778263	46.178871	**46.178633**
can-715	715	5	3.104721	3.464374	**2.940939**
L	956	6	1.483827	1.771585	**1.077067**
dwt-1005	1005	10	6.053665	6.664872	**5.095268**
road-minnesota	2642	10	1.789967	2.247812	**0.845962**
socfb-Brandeis99	3898	2	6.554323	**0.000000**	**0.000000**
bio-dmela	7393	3	0.854338	1.072443	**0.366938**
soc52	52	3	**2.437500**	**2.437500**	**2.437500**
gplus_200	200	5	0.872655	**0.851482**	**0.851482**
gplus_500	500	5	0.847377	1.055230	**0.668285**
pokec_500	500	5	0.866088	0.583463	**0.485606**
gplus_2000	2000	10	3.104040	3.366344	**2.343085**
pokec_2000	2000	10	5.311986	5.690171	**3.605891**

all three heuristics were capable of finding the best solution. A summary of the pairwise comparison results between the algorithms is given in Table 7. The entry "No" in the last column means that there is no significant difference between the results of the two compared algorithms. From the table, it can be concluded that MA is by far the best performing algorithm for the (b) set of graphs. The number of losses is at most 1. We can also see that there is no marked difference between the results of MSA and ITS at the selected significance level of 0.05.

The computational time taken by each of the tested algorithms is depicted in Fig. 4b. To construct the plots shown there, we split the interval [250, 7750] into subintervals of length 500 and then, for each subinterval, identified all graphs whose number of vertices falls within that subinterval. In this way, we obtained 10 nonempty subsets of graphs. For example, the subset corresponding to the subinterval [250, 750) consists of 10 graphs. Each point in the plot represents the average value of the time taken over all graphs in the corresponding subset and is placed at the x coordinate being

Table 6 Average results for ratio cut when running MSA, ITS, and MA on benchmark graphs from the literature (the time is in seconds)

Graph	n	m	MSA		ITS		MA	
			F_r	Time	F_r	Time	F_r	Time
karate	34	2	**0.937931**	23	**0.937931**	0	**0.937931**	0
dolphins	62	2	**0.432056**	15	**0.432056**	0	**0.432056**	0
polbooks	105	3	**2.295626**	18	**2.295626**	0	**2.295626**	0
football	115	12	**41.956854**	14	**41.956854**	0	**41.956854**	0
jazz	198	2	3.498814	14	1.010230	0	**1.005076**	0
celegansneural	297	20	39.537592	126	34.427923	42	**31.111511**	0
celegans_metabolic	453	3	0.918349	138	1.454931	62	**0.906726**	1
delaunay_n10	1024	20	18.777441	759	20.083913	84	**15.515484**	644
email	1133	2	0.565147	598	0.892314	147	**0.500884**	1
netscience	1461	12	0.210992	462	0.199925	807	**0.000000**	54
delaunay_n11	2048	30	30.569323	1241	31.267510	74	**20.979736**	1816
add20	2395	5	**0.783234**	1323	0.930753	833	0.801684	459
data	2851	25	27.978858	1477	31.938601	151	**19.016062**	1959
delaunay_n12	4096	50	59.404881	2065	48.470260	240	**33.677608**	3858
3elt	4720	20	13.628198	2996	11.701970	245	**5.464363**	3744
uk	4824	25	11.426019	2263	10.800167	2975	**4.020491**	3863
power	4941	20	5.846557	2418	6.380431	2936	**3.260920**	3787
add32	4960	20	4.816276	2513	5.591480	2496	**3.611136**	2900
Trefethen-200	200	6	29.428377	16	**29.177737**	1	**29.177737**	0
ash292	292	5	2.173243	107	2.138945	58	**2.110577**	27
can-292	292	4	**2.701870**	101	2.810330	40	**2.701870**	1
ia-infect-dublin	410	2	0.630187	82	0.629642	77	**0.515001**	1
dwt-503	503	3	1.434890	209	1.486739	338	**1.434769**	7
Trefethen-700	700	8	46.949315	250	46.256942	218	**46.178633**	26
can-715	715	5	3.358126	304	3.613356	207	**2.940939**	25
L	956	6	1.683417	262	2.031320	54	**1.083021**	160
dwt-1005	1005	10	6.756896	499	7.723954	29	**5.138586**	320
road-minnesota	2642	10	2.099074	1137	2.446506	1033	**0.929364**	1885
socfb-Brandeis99	3898	2	6.554323	1136	**0.000000**	0	**0.000000**	1
bio-dmela	7393	3	0.889243	6422	1.157065	2651	**0.448604**	996
soc52	52	3	**2.437500**	15	**2.437500**	0	**2.437500**	0
gplus_200	200	5	0.872655	15	0.941756	16	**0.851482**	0
gplus_500	500	5	0.966212	77	1.148792	129	**0.675894**	25
pokec_500	500	5	0.992090	152	0.716869	62	**0.485606**	3
gplus_2000	2000	10	3.470316	632	4.034365	632	**2.435137**	147
pokec_2000	2000	10	6.351653	517	6.713176	731	**4.194163**	226

Table 7 Summary of the results obtained by the tested algorithms for ratio cut on benchmark graphs from the literature

Algorithm pair	OFV	#wins	#ties	#losses	p-value	Statistical significance
MA versus MSA	Best	27	8	1	< 0.001	Yes
	Average	29	6	1	< 0.001	Yes
MA versus ITS	Best	24	12	0	< 0.001	Yes
	Average	29	7	0	< 0.001	Yes
MSA versus ITS	Best	16	7	13	> 0.2	No
	Average	19	5	12	> 0.2	No

Table 8 Best results for a normalized cut when running MSA, ITS, and MA on random graphs

Graph	F_n^*		
	MSA	ITS	MA
g-200-1	1.329776	**1.315066**	**1.315066**
g-200-2	1.401752	**1.363153**	**1.363153**
g-200-3	1.219671	**1.205851**	**1.205851**
g-200-4	**1.207305**	1.215972	**1.207305**
g-200-5	**1.266439**	**1.266439**	**1.266439**
g-500-1	3.468584	3.476683	**3.455826**
g-500-2	3.841118	**3.835036**	**3.835036**
g-500-3	3.792776	3.794346	**3.790331**
g-500-4	3.484301	3.540494	**3.468899**
g-500-5	**3.682650**	**3.682650**	3.685251
g-1000-1	9.834501	10.013168	**9.724178**
g-1000-2	9.937449	9.987719	**9.630570**
g-1000-3	10.082908	9.995309	**9.955058**
g-1000-4	9.947898	10.043799	**9.676365**
g-1000-5	9.986396	9.890707	**9.599150**
g-2000-1	16.900033	17.037897	**16.863759**
g-2000-2	17.302464	17.290662	**16.875586**
g-2000-3	17.261999	17.283946	**16.919533**
g-2000-4	17.286398	17.056313	**16.992265**
g-2000-5	17.166575	17.051547	**16.949080**
g-3000-1	32.868900	32.749950	**32.245723**
g-3000-2	32.914623	32.825475	**32.158796**
g-3000-3	32.940740	33.094891	**32.469108**
g-3000-4	32.924627	33.059634	**32.490853**
g-3000-5	33.163969	33.066114	**32.455933**

the midpoint of the corresponding subinterval. We excluded from consideration the 8 smallest graphs (of an order less than 250) because the algorithms, especially ITS and MA, require a very small amount of time to partition. The rightmost points in the plots were obtained for a single graph, namely `bio-dmela`. Perhaps they can be marked as outliers when comparing the algorithms. We can see in Fig. 4b that ITS tends to take less CPU time than MSA and MA. These two latter algorithms are comparable in terms of computation time. In fact, MA was slightly faster than MSA for graphs of an order $n < 2700$, and MSA is faster than MA for graphs with $n > 2700$. The running times of the tested algorithms for individual graphs are listed in Table 6.

5.4 Numerical results for a normalized cut

In the second phase of experimentation, we tested the developed algorithms for the graph partitioning problem with the normalized cut objective function. Their performance was evaluated using the same scenario as in the previous section.

First, we tested MSA, ITS, and MA on the set of random graphs. The results are reported in Tables 8 and 9. The objective function value of the best solution (out of 10 runs) is denoted by F_n^* and the average objective function value of 10 solutions is denoted by $\bar{F}_n$. Contrasting the results from Tables 2 and 3, and those from Tables 8 and 9, we see that the MA showed equally strong performance in solving both partitioning problems on random graphs. We can find in Table 8 that MA produced the best solution for 24 graphs in the set, whereas MSA and ITS obtained the best result for 3 and 6 graphs, respectively. We observe from Table 9 that MA provided the best average solutions for 23 random graphs. Each of the other two algorithms achieved the best average value for a single graph. The results of the pairwise comparison of algorithms are shown in Table 10. They statistically support the assertion that MA performs better than MSA and ITS. Looking at the statistics for MSA and ITS, we observe a significant difference in favor of MSA only for the case of average solutions.

Figure 3b shows the best solution found for graph `g-200-5`. This solution is achieved by all three heuristics. In particular, the MA arrived at this solution in each of 10 runs. The solutions for the ratio cut (Fig. 3a) and normalized cut (Fig. 3b) were quite different. The ratio cut objective function value for the second of these solutions was 48.911503, which was much worse than that for the first one. In contrast, the first solution was inferior to the second one in terms of the normalized cut objective function. Its value was 1.460579 against 1.266439 for the second solution.

The execution time of the tested algorithms is plotted in Fig. 5a. The ranking of the algorithms from fastest to slowest was ITS, MSA, and MA. This is exactly the same as in the case of ratio cut minimization. It can also be noted that MA is marginally faster than ITS and MSA for small graphs (with $n \leq 500$).

Tables 11 and 12 summarize the results of MSA, ITS, and MA on the set of benchmark graphs from the literature. Inspecting these tables reveals the clear superiority of the MA over other approaches in our evaluation. This algorithm produced the best partition for each of the 36 graphs. We also see that MSA and ITS yielded the best solution for 8 and 12 graphs, respectively, in the set. Looking at the average values $\bar{F}_n$, we found that MA obtained the best result for all graphs, while MSA and ITS failed in many cases and matched the MA performance for 6 and 8 benchmark graphs, respectively. We also made a pairwise comparison of heuristics. The results are displayed in Table 13, from which it is apparent that MA dominates the other two algorithms. The superiority of MA over other approaches is even more pronounced than in the random graph case. From the last two rows in Table 13, we can conclude that the quality of solutions delivered by MSA and ITS looks comparable.

Table 9 Average results for a normalized cut when running MSA, ITS, and MA on random graphs (the time is in seconds)

Graph	MSA		ITS		MA	
	F_n	Time	F_n	Time	F_n	Time
g-200-1	1.329776	14	**1.315284**	12	1.315828	1
g-200-2	1.434167	17	1.369071	14	**1.363153**	3
g-200-3	1.238273	16	1.206219	8	**1.205851**	2
g-200-4	1.223865	17	1.278333	12	**1.207305**	1
g-200-5	1.272482	15	1.272793	15	**1.266439**	1
g-500-1	3.498204	114	3.583851	132	**3.465290**	66
g-500-2	3.917550	115	3.879862	105	**3.847909**	45
g-500-3	3.869482	62	3.854537	105	**3.798395**	89
g-500-4	3.519804	94	3.653742	124	**3.483869**	36
g-500-5	**3.692783**	101	3.774761	76	3.695054	40
g-1000-1	10.051920	320	10.168825	185	**9.758664**	310
g-1000-2	10.091892	320	10.209080	131	**9.642397**	342
g-1000-3	10.203580	301	10.312122	92	**9.981957**	360
g-1000-4	10.035471	342	10.193633	63	**9.707875**	384
g-1000-5	10.092003	210	10.229109	164	**9.623047**	369
g-2000-1	17.218792	477	17.425192	373	**16.906716**	869
g-2000-2	17.575799	572	17.722551	316	**16.915658**	912
g-2000-3	17.474202	483	17.654850	400	**16.992583**	899
g-2000-4	17.418200	636	17.515490	450	**17.042475**	950
g-2000-5	17.359247	647	17.448308	309	**16.996156**	905
g-3000-1	33.033366	1631	33.091513	781	**32.344338**	1663
g-3000-2	33.066883	1430	33.119156	1372	**32.245668**	1803
g-3000-3	33.190598	1133	33.299338	880	**32.532437**	1770
g-3000-4	33.144343	1442	33.297176	577	**32.576456**	1914
g-3000-5	33.354442	1256	33.478115	1187	**32.575388**	1668

Table 10 Summary of the results obtained by the tested algorithms for a normalized cut on random graphs

Algorithm pair	OFV	#wins	#ties	#losses	p-value	Statistical significance
MA versus MSA	Best	22	2	1	< 0.001	Yes
	Average	24	0	1	< 0.001	Yes
MA versus ITS	Best	19	5	1	< 0.001	Yes
	Average	24	0	1	< 0.001	Yes
MSA versus ITS	Best	11	2	12	> 0.2	No
	Average	20	0	5	< 0.001	Yes

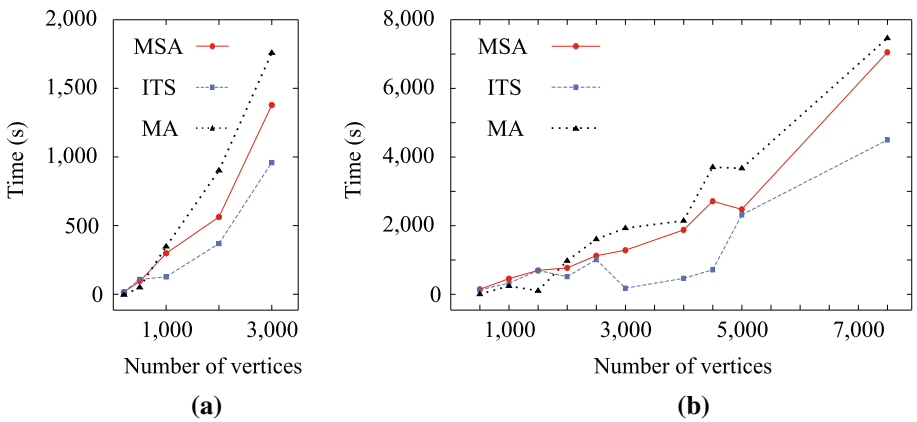


Fig. 5 Time taken by MSA, ITS, and MA to find the best solution in a run (the case of normalized cut). **a** Random graphs. **b** Benchmark graphs

Table 11 Best results for a normalized cut when running MSA, ITS, and MA on benchmark graphs from the literature

Graph	n	m	F_n^*		
			MSA	ITS	MA
karate	34	2	**0.256410**	**0.256410**	**0.256410**
dolphins	62	2	**0.090616**	**0.090616**	**0.090616**
polbooks	105	3	**0.357999**	**0.357999**	**0.357999**
football	115	12	**4.046195**	**4.046195**	**4.046195**
jazz	198	2	**0.157503**	**0.157503**	**0.157503**
celegansneural	297	20	9.942777	10.005737	**9.878295**
celegans_metabolic	453	3	0.443966	0.474388	**0.195767**
delaunay_n10	1024	20	2.879400	2.871209	**2.592740**
email	1133	2	0.232049	0.232504	**0.215629**
netscience	1461	12	0.097262	0.077844	**0.000000**
delaunay_n11	2048	30	4.568587	4.349694	**3.470871**
add20	2395	5	0.600447	0.629620	**0.386418**
data	2851	25	3.286234	2.930128	**2.038039**
delaunay_n12	4096	50	8.691735	7.167249	**5.609273**
3elt	4720	20	2.158368	1.587407	**0.929342**
uk	4824	25	4.057219	3.607386	**1.629709**
power	4941	20	1.625724	2.332730	**1.349342**
add32	4960	20	0.975372	1.555498	**0.326691**
Trefethen-200	200	6	**2.249661**	**2.249661**	**2.249661**
ash292	292	5	0.351585	0.343277	**0.342347**
can-292	292	4	**0.385405**	**0.385405**	**0.385405**
ia-infect-dublin	410	2	0.063591	**0.043793**	**0.043793**
dwt-503	503	3	0.191229	0.189659	**0.184988**
Trefethen-700	700	8	2.773342	**2.772250**	**2.772250**
can-715	715	5	0.441312	**0.435508**	**0.435508**
L	956	6	0.447864	0.454578	**0.283702**
dwt-1005	1005	10	0.927519	0.907562	**0.785061**
road-minnesota	2642	10	0.851094	0.829972	**0.330296**
socfb-Brandeis99	3898	2	0.320622	**0.147692**	**0.147692**
bio-dmela	7393	3	0.876248	0.762828	**0.403828**
soc52	52	3	**0.712968**	**0.712968**	**0.712968**
gplus_200	200	5	0.290692	0.302318	**0.277428**
gplus_500	500	5	0.230335	0.297260	**0.196626**
pokec_500	500	5	0.171848	0.197513	**0.129259**
gplus_2000	2000	10	1.086015	1.053661	**0.647314**
pokec_2000	2000	10	1.511054	1.548266	**0.842486**

Figure 5b illustrates the computational time of the tested algorithms as a function of the graph size. We can see that as in the ratio cut case, ITS took less time to find the best solution in a run than MSA and MA. In particular, ITS dominates MSA over the whole range of graph sizes. We also observe that MA is the fastest approach for small graphs (with $n \leq 1500$). However, with increasing n, it became the slowest algorithm. The timing results for individual graphs are shown in Table 12.

5.5 Comparison with the state of the art

For comparison purposes, we implemented the VNS algorithm of Hansen et al. [27]. Similar to the algorithms in this paper, VNS has been coded in the C++ programming language. The experiments were conducted with the parameter values used in [27]. We restricted ourselves to comparing the best of our algorithms, MA, against VNS. We remind that Hansen et al. [27] developed their algorithm for solving the normalized cut graph partitioning problem.

Table 12 Average results for a normalized cut when running MSA, ITS, and MA on benchmark graphs from the literature (the time is in seconds)

Graph	n	m	MSA		ITS		MA	
			$\overline{F_n}$	Time	$\overline{F_n}$	Time	$\overline{F_n}$	Time
karate	34	2	**0.256410**	18	**0.256410**	0	**0.256410**	0
dolphins	62	2	**0.090616**	16	**0.090616**	0	**0.090616**	0
polbooks	105	3	**0.357999**	15	**0.357999**	0	**0.357999**	0
football	115	12	**4.046195**	16	**4.046195**	0	**4.046195**	0
jazz	198	2	**0.157503**	12	**0.157503**	0	**0.157503**	0
celegansneural	297	20	9.978573	102	10.043401	129	**9.889473**	91
celegans_metabolic	453	3	0.444504	63	0.535099	119	**0.195767**	5
delaunay_n10	1024	20	2.956622	568	3.082031	281	**2.620307**	733
email	1133	2	0.232101	513	0.236139	546	**0.216769**	62
netscience	1461	12	0.122566	694	0.214127	686	**0.000000**	120
delaunay_n11	2048	30	4.730520	990	4.804968	143	**3.504679**	1815
add20	2395	5	0.637167	893	0.752580	1041	**0.391417**	1309
data	2851	25	3.486326	1282	3.411918	177	**2.067953**	1946
delaunay_n12	4096	50	8.926666	2687	8.007167	465	**5.671613**	3908
3elt	4720	20	2.280803	2710	1.932474	712	**0.943059**	3723
uk	4824	25	4.234634	2318	4.168123	2684	**1.980777**	3882
power	4941	20	1.823150	2573	2.489125	2293	**1.530633**	3791
add32	4960	20	1.201003	2526	1.716454	1955	**0.403704**	3391
Trefethen-200	200	6	2.261075	14	**2.249661**	1	**2.249661**	1
ash292	292	5	0.359794	90	0.344835	102	**0.342347**	2
can-292	292	4	0.388111	111	**0.385405**	51	**0.385405**	1
ia-infect-dublin	410	2	0.063591	69	0.059631	20	**0.043793**	1
dwt-503	503	3	0.193707	304	0.190427	187	**0.188414**	27
Trefethen-700	700	8	2.793341	258	2.772296	248	**2.772250**	38
can-715	715	5	0.460751	264	0.442612	222	**0.436007**	24
L	956	6	0.499008	219	0.515649	56	**0.284154**	112
dwt-1005	1005	10	0.990432	511	1.012138	326	**0.797461**	133
road-minnesota	2642	10	0.920393	1351	0.985167	979	**0.361209**	1939
socfb-Brandeis99	3898	2	0.321812	1061	0.147705	461	**0.147701**	418
bio-dmela	7393	3	0.889283	7049	0.810167	4496	**0.413778**	7481
soc52	52	3	**0.712968**	17	**0.712968**	0	**0.712968**	0
gplus_200	200	5	0.293464	13	0.334596	18	**0.277428**	0
gplus_500	500	5	0.238586	107	0.317876	125	**0.204525**	18
pokec_500	500	5	0.196052	72	0.234238	79	**0.129259**	6
gplus_2000	2000	10	1.256574	599	1.184974	707	**0.700524**	521
pokec_2000	2000	10	1.691226	716	1.717825	687	**0.845040**	648

Table 13 Summary of the results obtained by the tested algorithms for a normalized cut on benchmark graphs from the literature

Algorithm pair	OFV	#wins	#ties	#losses	p-value	Statistical significance
MA versus MSA	Best	28	8	0	< 0.001	Yes
	Average	30	6	0	< 0.001	Yes
MA versus ITS	Best	24	12	0	< 0.001	Yes
	Average	28	8	0	< 0.001	Yes
ITS versus MSA	Best	17	8	11	> 0.2	No
	Average	14	6	16	> 0.2	No

Table 14 Comparison of the memetic algorithm with the state-of-the-art algorithm, VNS, of Hansen et al. [27] for normalized cut partitioning of random graphs (the time is in seconds)

Graph	VNS [27]			MA		
	F_n^*	$\bar{F}_n$	Time	F_n^*	$\bar{F}_n$	Time
g-200-1	**1.315066**	1.331910	1	**1.315066**	*1.315828*	1
g-200-2	**1.363153**	1.391170	7	**1.363153**	*1.363153*	3
g-200-3	1.207078	1.226515	1	**1.205851**	*1.205851*	2
g-200-4	**1.207305**	1.283627	4	**1.207305**	*1.207305*	1
g-200-5	**1.266439**	1.297508	1	**1.266439**	*1.266439*	1
g-500-1	3.565152	3.691304	61	**3.455826**	*3.465290*	66
g-500-2	**3.835036**	3.901362	61	**3.835036**	*3.847909*	45
g-500-3	**3.790331**	3.902471	36	**3.790331**	*3.798395*	89
g-500-4	3.576887	3.703809	69	**3.468899**	*3.483869*	36
g-500-5	3.770853	3.838569	78	**3.685251**	*3.695054*	40
g-1000-1	9.825321	9.944504	206	**9.724178**	*9.758664*	310
g-1000-2	9.796339	9.883550	182	**9.630570**	*9.642397*	342
g-1000-3	9.972615	10.050408	213	**9.955058**	*9.981957*	360
g-1000-4	9.755182	9.885892	289	**9.676365**	*9.707875*	384
g-1000-5	9.694180	9.817429	201	**9.599150**	*9.623047*	369
g-2000-1	16.881007	16.939856	468	**16.863759**	*16.906716*	869
g-2000-2	16.933214	17.147329	553	**16.875586**	*16.915658*	912
g-2000-3	16.975062	17.032259	630	**16.919533**	*16.992583*	899
g-2000-4	**16.969761**	17.075015	515	16.992265	*17.042475*	950
g-2000-5	16.974046	17.054939	419	**16.949080**	*16.996156*	905
g-3000-1	**31.981446**	*32.182476*	1039	32.245723	32.344338	1663
g-3000-2	**32.155155**	32.256756	1421	32.158796	*32.245668*	1803
g-3000-3	**32.257388**	*32.339845*	1473	32.469108	32.532437	1770
g-3000-4	**32.321777**	*32.416463*	1225	32.490853	32.576456	1914
g-3000-5	**32.221956**	*32.351820*	1120	32.455933	32.575388	1668

Table 15 Comparison of MA versus VNS: a summary of the results

Graphs	OFV	#wins	#ties	#losses	p-value	Statistical significance
Random	Best	13	6	6	> 0.2	No
	Average	21	0	4	< 0.025	Yes
Benchmarks	Best	24	11	1	< 0.001	Yes
	Average	30	5	1	< 0.001	Yes

In Table 14, we report the results achieved by VNS and MA for random graphs. The three columns for each algorithm report F_n^*, $\bar{F}_n$, and the average time for reaching the best result. The best value of F_n^* and $\bar{F}_n$ for each instance is shown in boldface and in italics, respectively. It can be seen that MA performs better or equally well with VNS for all graphs of order up to 2000 except for `g-2000-4` in terms of F_n^*. A different conclusion is reached for graphs of order 3000, where VNS produced better partitions than MA. We also observe that, for $n \geq 1000$, the average time taken by VNS to find the best solution in a run is significantly shorter than that of MA. This means that VNS spends more time than MA without improving the quality of the output solution.

We summarize the results of the experiment in the first two rows of Table 15. As before, the Wilcoxon signed-rank test at the significance level of 0.05 was used to compare the performance of the two algorithms. The test demonstrated a statistically significant difference in the average quality of solutions obtained by MA and VNS. We see that MA outperformed VNS on the 21 random graphs (out of 25). However, there was no statistically significant difference between the results of the two algorithms in the case of the best solutions.

Table 16 shows the results of the experiment on the set of benchmark graphs. The last six columns of this table have the same structure as in Table 14. We observe that MA surpassed or matched the VNS algorithm across all graphs except `uk`. We also see that VNS could find the best result for 12 of

Table 16 Comparison of the memetic algorithm with the state-of-the-art algorithm, VNS, of Hansen et al. [27] for normalized cut partitioning of benchmark graphs (the time is in seconds)

Graph	n	m	VNS [27]			MA		
			F_n^*	F_n	Time	F_n^*	F_n	Time
karate	34	2	**0.256410**	*0.256410*	0	**0.256410**	*0.256410*	0
dolphins	62	2	**0.090616**	*0.090616*	0	**0.090616**	*0.090616*	0
polbooks	105	3	**0.357999**	*0.357999*	0	**0.357999**	*0.357999*	0
football	115	12	**4.046195**	*4.046195*	0	**4.046195**	*4.046195*	0
jazz	198	2	**0.157503**	0.249612	0	**0.157503**	*0.157503*	0
celegansneural	297	20	**9.878295**	9.899448	73	**9.878295**	*9.889473*	91
celegans_metabolic	453	3	0.443966	0.661832	67	**0.195767**	*0.195767*	5
delaunay_n10	1024	20	2.601046	2.702850	584	**2.592740**	*2.620307*	733
email	1133	2	0.232049	0.252497	169	**0.215629**	*0.216769*	62
netscience	1461	12	0.073751	0.121331	873	**0.000000**	*0.000000*	120
delaunay_n11	2048	30	3.810993	3.954550	1577	**3.470871**	*3.504679*	1815
add20	2395	5	0.602376	0.672602	1195	**0.386418**	*0.391417*	1309
data	2851	25	2.427628	2.625715	1859	**2.038039**	*2.067953*	1946
delaunay_n12	4096	50	6.508810	6.825981	3818	**5.609273**	*5.671613*	3908
3elt	4720	20	1.335464	1.491187	3799	**0.929342**	*0.943059*	3723
uk	4824	25	**1.562372**	*1.781972*	3899	1.629709	1.980777	3882
power	4941	20	1.641364	1.751849	3972	**1.349342**	*1.530633*	3791
add32	4960	20	1.389029	1.589182	3708	**0.326691**	*0.403704*	3391
Trefethen-200	200	6	**2.249661**	2.253546	1	**2.249661**	*2.249661*	1
ash292	292	5	0.343277	0.388463	21	**0.342347**	*0.342347*	2
can-292	292	4	**0.385405**	0.527403	15	**0.385405**	*0.385405*	1
ia-infect-dublin	410	2	**0.043793**	0.065767	0	**0.043793**	*0.043793*	1
dwt-503	503	3	0.200969	0.227937	147	**0.184988**	*0.188414*	27
Trefethen-700	700	8	2.772310	2.861548	98	**2.772250**	*2.772250*	38
can-715	715	5	0.440794	0.485988	31	**0.435508**	*0.436007*	24
L	956	6	0.301281	0.359839	257	**0.283702**	*0.284154*	112
dwt-1005	1005	10	0.856210	0.952162	350	**0.785061**	*0.797461*	133
road-minnesota	2642	10	0.656730	0.720834	1945	**0.330296**	*0.361209*	1939
socfb-Brandeis99	3898	2	0.298508	0.298508	268	**0.147692**	*0.147701*	418
bio-dmela	7393	3	0.748778	0.769965	7716	**0.403828**	*0.413778*	7481
soc52	52	3	**0.712968**	*0.712968*	0	**0.712968**	*0.712968*	0
gplus_200	200	5	**0.277428**	0.287320	9	**0.277428**	*0.277428*	0
gplus_500	500	5	0.251453	0.303077	136	**0.196626**	*0.204525*	18
pokec_500	500	5	0.268187	0.349631	51	**0.129259**	*0.129259*	6
gplus_2000	2000	10	1.064966	1.257695	904	**0.647314**	*0.700524*	521
pokec_2000	2000	10	1.650285	1.867055	755	**0.842486**	*0.845040*	648

36 graphs. However, almost all of these graphs are of small sizes. To compare the quality of solutions produced by the algorithms more accurately, we calculate the relative difference between their objective function values. To this end, we apply the following formula

$$e = ((F_{\mathrm{VNS}} - F_{\mathrm{MA}})/\min(F_{\mathrm{VNS}}, F_{\mathrm{MA}}))100\%, \tag{17}$$

where e is the relative difference, and F_{VNS} and F_{MA} denote the objective function values achieved by VNS and MA, respectively. Figure 6 shows the relative differences calculated using (17) for F_{VNS} and F_{MA} values contained in the columns in Table 16 labelled F_n^*. In this figure, we omit `netscience` for which e is not defined as well as the graphs for which $e < 2\%$. The results demonstrate that MA significantly outperforms the VNS algorithm. The average relative

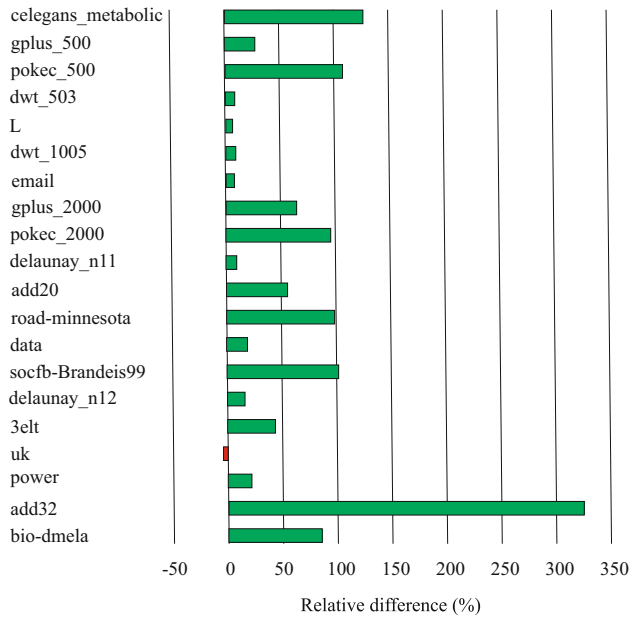


Fig. 6 Relative difference between the best objective function value obtained by the MA and VNS methods

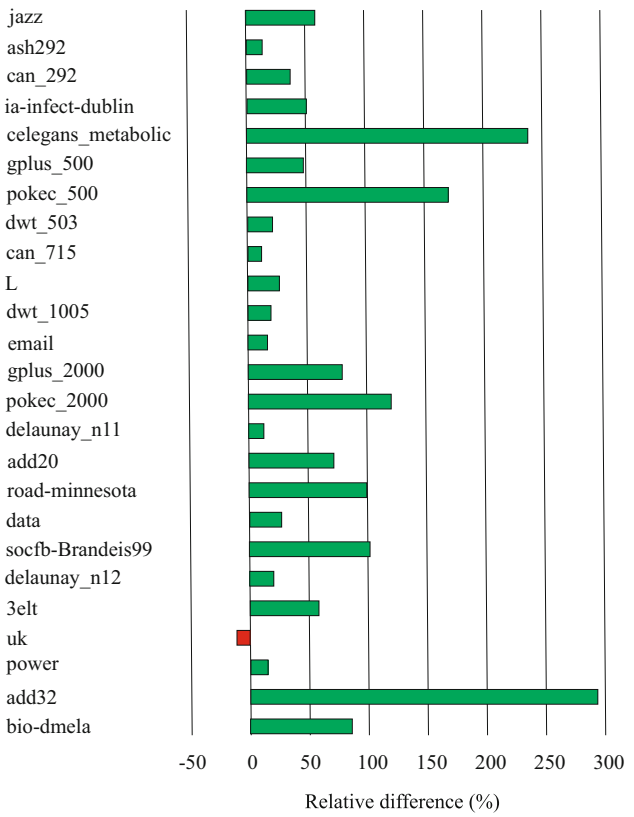


Fig. 7 Relative difference between the average objective function value of solutions found by the MA and VNS methods

difference (calculated for 35 graphs) between the F_n^* values of VNS and MA is 35%.

From Table 16, we find that the advantage of MA over VNS becomes even more prominent when considering the average performance of algorithms. The VNS heuristic achieved the best average result for `uk` and 5 smallest graphs only. There are a number of graphs for which VNS, unlike MA, failed to find the best solution in all 10 runs. Three such graphs (`jazz`, `Trefethen-200`, `gplus_200`) have 200 or fewer vertices. Figure 7 depicts the percentage differences calculated by substituting the $\bar{F}_n$ values of VNS and MA into Equation (17). The figure lists only those graphs for which $e > 4\%$. The results again indicate a very clear superiority of MA over the VNS algorithm. The average relative difference between VNS and MA in terms of the $\bar{F}_n$ value is 48%. Statistically significant differences among the algorithms are confirmed by the Wilcoxon signed-rank test in the last two rows of Table 15.

Evaluating the computational speed of the MA and VNS algorithms reveals that they take a similar amount of time to find the best solution in a run (compare the time columns in Table 16). This especially can be seen when the graph order becomes large. Overall, the average time taken by VNS was 1064 s and that taken by MA was 1004 s.

5.6 Comparison with a different variant of the crossover operation

The numerical results reported in Sects. 5.4 and 5.5 demonstrate the effectiveness and excellent performance of the MA in solving the normalized cut partitioning problem for benchmark graphs as well as for random graphs of order up to 2000. However, MA was less successful for random graphs with 3000 vertices. It is difficult to determine a possible reason for this shortcoming. We investigated the behavior of the offspring generation procedure `get_offspring`. In the first stage of this procedure, all vertices of the graph were assigned to offspring subsets. If the required number of subsets was reached, then the procedure stopped (Line 25 of Algorithm 6). Otherwise, a subset splitting operation was performed until m partition subsets were created (Lines 26–36 of Algorithm 6). We observed a much more frequent use of this operation for random graphs with $n = 3000$ in comparison with benchmark graphs and small random graphs. In the case of random graphs of order 3000, the splitting operation was applied on average in 79.9% of all generated offspring (the average is taken over 5 graphs and 10 runs for each of them). This percentage reduces to 49.1% for random graphs with $n = 2000$ and is less than 1% for all benchmark graphs except `football` and `soc52`, which are very easy to solve by all tested algorithms. One might guess that frequent use of the subset splitting operation may have a negative impact on the performance of the proposed memetic algorithm.

In the process of MA design, we developed a version of the algorithm with a different variant of the crossover operation. This crossover differs from those typically used in GGAs. We restrict ourselves to a brief outline of the alternative version of the MA because this algorithm generally showed inferior performance compared to the MA presented in Sect. 4. The alternative crossover procedure uses the matrix $H = (h_{ij})$ introduced in Sect. 4. Suppose that H is constructed for parent partitions p' and p''. Let q' and q'' be the mappings corresponding to partitions p' and p'', respectively. The procedure first identifies the m largest entries of the matrix H. Their set (denoted by Ψ) is used to construct a one-to-one mapping g from $\{(i, j) \mid h_{ij} \in \Psi\}$ to M. Basically, the mapping g can be simply viewed as arbitrary labeling of the elements of Ψ by the integers in the set $\{1, \ldots, m\}$. Having the mapping g, the procedure can define sets $A_i = \{g(i,k) \mid k \in \{1, \ldots, m\}, h_{ik} \in \Psi\}$, $i \in M$, and $B_j = \{g(k, j) \mid k \in \{1, \ldots, m\}, h_{kj} \in \Psi\}$, $j \in M$. Then, the vertices of the graph are processed one by one. For $v \in V$, let $i = q'(v)$, $j = q''(v)$. If i and j are such that $h_{ij} \in \Psi$, then $q(v)$ is set to $g(i, j)$. Otherwise, $q(v)$ is assigned a partition subset index chosen randomly either from the set $A_i \cup B_j$ if this set is nonempty or from the set M. The resulting offspring partition is represented by the mapping q. We use MA′ to refer to the version of the MA with the just outlined crossover operator.

We experimentally compared MA′ versus MA and concluded that the performance of MA′ was inferior to that of MA for benchmark graphs and for smaller random graphs. However, MA′ is more effective than MA when solving the normalized cut graph partitioning problem for random graphs of order 2000 and 3000. This can clearly be seen in Table 17, where the results for VNS from Table 14 are repeated for comparison reasons. MA′ produced the best solution for 8 graphs, whereas MA and VNS yielded the best partition for only one graph. We also observe that MA′ obtained the best average results for 9 of 10 graphs. By comparing the time columns for MA in Table 14 and MA′ in Table 17, we found that MA′ was more than twice as fast as MA. However, as mentioned before, MA′ could not successfully compete with MA when the comparison between algorithms was performed over all graphs in our test suite. In particular, MA yielded better or equal quality solutions compared to MA′ for almost all (34 out of 36) benchmark graphs. To save space, we, therefore, do not provide detailed experimental results for MA′.

5.7 Comparisons with the genetic algorithm and multistart local search

To further evaluate the MA performance, we compared it with the genetic algorithm for normalized cut graph partitioning. We constructed the latter simply by removing all the calls to the LS procedure from MA. We refer to this algorithm as GA1. We also tested another configuration of the genetic algorithm, which was obtained from MA by removing calls to `local_search` from the offspring generation steps only. In other words, we modified MA by deleting Step 7 of Algorithm 7. However, we kept using `local_search` while generating the initial population. We call this configuration of the genetic algorithm GA2. Additionally, we evaluated the performance of the multistart local search method. This method is very simple and straightforward. It consists of repetitively applying `local_search` to randomly generated starting partitions of the graph. We refer to this algorithm as MLS.

To avoid unnecessarily long computations, we performed the comparison of algorithms on a set of smaller-size graphs from our testbed. This set consists of 10 random graphs with 200 and 500 vertices and all benchmark graphs of order less than 1000. We run each algorithm 10 times on each graph with the same cutoff time as before. To assess the performance of GA1, GA2, and MLS, we used MA as a reference method.

Table 18 summarizes the results of the computational experiments carried out for GA1, GA2, and MLS. We write $F_{\mathrm{GA1}} - F_{\mathrm{MA}}$ to denote the difference in the F_n^* values between GA1 and MA in the column under heading "Best", and the difference in the $\bar{F}_n$ values between GA1 and MA in the column under heading "Average". The differences $F_{\mathrm{GA2}} - F_{\mathrm{MA}}$ and $F_{\mathrm{MLS}} - F_{\mathrm{MA}}$ are defined similarly. The three bottom lines in the table serve as a summary that includes the differences between objective function values averaged over all 29 graphs, the number of graphs for which the two algorithms achieve the same accuracy, and the number of graphs for which the tested algorithm (GA1, GA2 or MLS) produced worse solutions than MA.

The main observation from the table is that MA shows great superiority over both genetic algorithm configurations and the multistart local search approach. We also see that the pure genetic algorithm, GA1, was by far the worst algorithm in the comparison. It was the only one that failed to match the performance of MA for all graphs in the test suite. Another conclusion is that the other two algorithms, GA2 and MLS, demonstrate comparable performance in terms of solution quality. However, each of them was significantly surpassed by the proposed memetic algorithm. Therefore, we conclude that the main reason for the success of MA lies in the integration of the grouping genetic algorithm with the fast LS technique. As Table 18 shows, individually, these components of the approach produce inferior results.

5.8 Analysis of the main parameters

In this section, we study the effect of the main parameters on the performance of the developed algorithms. Again,

Table 17 Comparison of MA and VNS with MA′ for normalized cut partitioning of large random graphs

Graph	VNS [27]		MA		MA′		Time
	F_n^*	$\overline{F}_n$	F_n^*	$\overline{F}_n$	F_n^*	$\overline{F}_n$	
g-2000-1	16.881007	16.939856	16.863759	16.906716	**16.835347**	*16.847496*	235
g-2000-2	16.933214	17.147329	**16.875586**	*16.915658*	16.957265	16.974367	304
g-2000-3	16.975062	17.032259	16.919533	16.992583	**16.826985**	*16.853760*	335
g-2000-4	16.969761	17.075015	16.992265	17.042475	**16.904339**	*16.914954*	304
g-2000-5	16.974046	17.054939	16.949080	16.996156	**16.911500**	*16.939781*	256
g-3000-1	**31.981446**	32.182476	32.245723	32.344338	31.987247	*32.026407*	798
g-3000-2	32.155155	32.256756	32.158796	32.245668	**32.111144**	*32.135333*	824
g-3000-3	32.257388	32.339845	32.469108	32.532437	**32.207694**	*32.237487*	837
g-3000-4	32.321777	32.416463	32.490853	32.576456	**32.188217**	*32.210425*	809
g-3000-5	32.221956	32.351820	32.455933	32.575388	**32.182967**	*32.211187*	899

Table 18 Difference in solution values between each tested algorithm (GA1, GA2, and MLS) and the reference method MA for normalized cut graph partitioning

Graph	$F_{GA1} - F_{MA}$		$F_{GA2} - F_{MA}$		$F_{MLS} - F_{MA}$	
	Best	Average	Best	Average	Best	Average
g-200-1	1.167162	1.357356	0	0.002052	0	0.001690
g-200-2	1.196249	1.341626	0	0.016591	0.009714	0.019265
g-200-3	1.183462	1.405166	0	0.018944	0.007902	0.029355
g-200-4	0.711652	0.936732	0	0.022860	0	0.015943
g-200-5	0.802133	0.995463	0	0.005915	0.000618	0.018303
g-500-1	2.885116	3.647773	0.045725	0.070485	0.075784	0.116486
g-500-2	3.441242	3.802045	0.015730	0.032751	0.021662	0.026262
g-500-3	3.020083	3.585652	0.007052	0.051348	0.051468	0.081077
g-500-4	2.393034	3.038013	0.057843	0.082211	0.121643	0.144298
g-500-5	3.113781	3.487876	0.021505	0.036614	0.059854	0.078331
karate	0.022161	0.085067	0	0	0	0
dolphins	0.151209	0.317575	0	0	0	0
polbooks	0.504993	0.720510	0	0	0	0
football	0.333149	1.406913	0	0	0	0
jazz	0.489774	0.599128	0	0	0	0
celegansneural	2.860721	3.168994	0.258713	0.332509	0.329260	0.358113
celegans_metabolic	1.148899	1.369098	0.252367	0.264751	0.182945	0.243158
Trefethen-200	1.653816	1.797297	0.000138	0.001707	0	0
ash292	1.729106	2.208209	0.008564	0.047468	0.024786	0.042343
can-292	1.484284	1.708798	0	0.001523	0	0
ia-infect-dublin	0.593671	0.654876	0	0	0	0
dwt-503	1.237672	1.386546	0.008630	0.014412	0.006166	0.004479
Trefethen-700	3.429825	3.526682	0.000170	0.000463	0.000023	0.000036
can-715	2.537377	2.764893	0.063423	0.103191	0.043325	0.071006
L	3.339905	3.616975	0.543790	0.613626	0.372426	0.437497
soc52	0.069794	0.222356	0	0	0	0
gplus_200	1.503558	1.720256	0.025582	0.065141	0	0.041644
gplus_500	2.077289	2.467264	0.191239	0.259102	0.181010	0.216877
pokec_500	2.238249	2.504857	0.172179	0.213432	0.137765	0.158294
Average	1.631702	1.925655	0.057678	0.077831	0.056081	0.072567
#ties	0	0	13	7	12	9
#losses	29	29	16	22	17	20

Table 19 Difference in solution values between the MSA algorithm with $\beta' = 100$ and MSA configurations with $\beta' = 500$ and 1000 for normalized cut graph partitioning

Graph	$F_{\text{MSA}(500)} - F_{\text{MSA}(100)}$		$F_{\text{MSA}(1000)} - F_{\text{MSA}(100)}$		#restarts for various β'		
	Best	Average	Best	Average	100	500	1000
g-200-1	0	0.025893	0	0.031799	22.4	5.0	3.0
g-200-2	−0.002139	0.031529	0.054999	0.065779	21.8	4.9	2.9
g-200-3	0.007379	−0.002211	0.016465	0.009080	22.8	5.0	3.0
g-200-4	0	0.007005	0	0.003394	18.2	4.0	2.0
g-200-5	0	0.017600	0.000618	0.022839	18.0	4.0	2.0
g-500-1	−0.005110	0.033308	0.007634	0.035170	23.6	5.0	3.0
g-500-2	0.024382	0.047372	0.054408	0.078468	31.1	6.7	4.0
g-500-3	0.036224	0.011362	0.036224	0.036448	27.7	6.0	3.0
g-500-4	−0.015402	0.028148	0.008865	0.053359	22.8	5.0	3.0
g-500-5	0.001208	0.037273	0.003961	0.039619	24.9	5.0	3.0
celegansneural	−0.005768	−0.016602	−0.029006	−0.005188	57.0	11.9	6.0
celegans_metabolic	0	0.000556	0	0.002066	33.2	7.1	4.0
Trefethen-200	0	0.016525	0.000138	0.027930	24.7	5.2	3.0
ash292	−0.009238	0.007046	−0.000097	0.012073	92.3	18.8	9.6
can-292	0	0.014561	0	0.018618	88.4	18.3	9.5
dwt-503	−0.000116	0.002411	0.000444	0.005273	102.1	20.6	10.9
Trefethen-700	0.040256	0.041165	0.031991	0.074261	27.6	6.0	3.0
can-715	0.015026	0.030757	0.015178	0.051910	24.4	5.3	3.0
L	−0.015007	0.023414	0.059726	1.682849	10.0	2.1	1.1
gplus_200	0	0.023964	−0.013264	0.053604	21.1	4.8	2.8
gplus_500	0.002408	0.007583	0.002408	0.024089	26.1	5.9	3.0
pokec_500	0.001040	0.024265	0.025697	0.045529	22.9	5.2	3.0
Average	0.003416	0.018769	0.012563	0.107680	34.7	7.4	4.0
#wins	7	2	3	1			
#ties	7	0	4	0			
#losses	8	20	15	21			

we focus on the normalized cut graph partitioning problem. We conducted computational experiments on the same set of graphs as used in the previous section. However, we do not report results for graphs for which all variations of the tested algorithm gave the best result. Additionally, we ran our memetic algorithm on several large graphs. The main configurations of the algorithms were used as a basis for comparison.

5.8.1 Usefulness of the restart strategy in simulated annealing

The parameter of MSA that helps to adjust the number of SA restarts is the number of moves attempted at each temperature level. Its value in the main experiment (Sect. 5.4) is set to $\beta = 100n$. Let us denote this parameter as $\beta = \beta' n$. By increasing β' we can decrease the number of SA restarts. We performed an additional experiment with the β' set to 500 and 1000. We denote the MSA configurations for tested β' values by MSA(100), MSA(500), and MSA(1000).

Table 19 compares the results of MSA(500) and MSA (1000) with those of MSA(100). Columns 2–5 of this table were obtained in the same manner as Columns 2–7 of Table 18. We denoted the objective function value achieved by MSA(β') with $\beta' \in \{100, 500, 1000\}$ by $F_{\text{MSA}(\beta')}$. As before, this value is F_n^* for columns labeled "Best" and $\bar{F}_n$ for columns labeled "Average". The $F_{\text{MSA}(100)}$ values were extracted from Tables 8, 9, 11, and 12. The last columns in Table 19 display the number of SA restarts (averaged over 10 runs) in the three scenarios considered. The bottom rows of the table show the wins/ties/losses for MSA(500) or MSA(1000) versus MSA(100).

As observed in Table 19, the difference in solution values between MSA(β') with $\beta' > 100$ and MSA(100) increased with increasing β'. This is especially visible when comparing average results. Thus, performing a larger number of shorter

Table 20 Difference in solution values between the ITS algorithm with $I = 100$ and ITS configurations with $I = 20$, 50, and 500 for normalized cut graph partitioning

Graph	$F_{\mathrm{ITS}(20)} - F_{\mathrm{ITS}(100)}$		$F_{\mathrm{ITS}(50)} - F_{\mathrm{ITS}(100)}$		$F_{\mathrm{ITS}(500)} - F_{\mathrm{ITS}(100)}$	
	Best	Average	Best	Average	Best	Average
g-200-1	0	0.019025	0	−0.000218	0	0.000687
g-200-2	0.026261	0.061670	0	0.002894	0	0.009038
g-200-3	0.001227	0.012612	0	0.001050	0	0.000736
g-200-4	0.021557	0.090407	0.013957	0.000386	0.035923	0.018332
g-200-5	0.000618	0.027548	0	0.001880	0	0.010790
g-500-1	0.032194	0.015965	0.010423	−0.011446	0.035913	0.141521
g-500-2	0.034383	0.041995	0.000790	−0.019134	0.000790	0.008335
g-500-3	0.040497	0.051091	0.014318	0.002553	0.027178	0.099457
g-500-4	0.026220	−0.003907	−0.064589	−0.073240	−0.018709	0.110520
g-500-5	0.027721	0.005420	0.020293	0.020865	0.075962	0.091639
celegansneural	0.004635	0.031424	0.000902	0.008464	−0.024926	0.022521
celegans_metabolic	−0.002320	0.038069	−0.001943	−0.003822	−0.001101	0.087545
Trefethen-200	0	0.002174	0	0	0	0.000014
ash292	0.008382	0.070563	0.001546	0.007580	−0.000930	0.001102
can-292	0	0.092973	0	0.002695	0	0.001986
ia-infect-dublin	0	0.001701	0.017009	0.003681	0	0.001701
dwt-503	0.005862	0.028297	0.001482	0.005425	0	0.001348
Trefethen-700	0.000911	0.019276	0.000606	0.003091	0	−0.000034
can-715	0.005715	0.021471	0	0.006284	0	0.007522
L	0.067159	0.090576	0.040737	0.052957	−0.118570	−0.077637
gplus_200	0.037575	0.038121	−0.006514	0.003038	0.013606	0.017676
gplus_500	0.001468	0.025732	0.001318	0.011869	−0.004480	0.043435
pokec_500	0.011421	−0.002065	0.013676	0.000236	−0.019656	0.002144
Average	0.015282	0.033919	0.002783	0.001178	0.000043	0.026103
#wins	1	2	3	5	7	2
#ties	4	0	7	1	10	0
#losses	18	21	13	17	6	21

SA runs from different starting points was better than performing a smaller number of longer SA runs.

5.8.2 The effect of the number of TS iterations on the ITS performance

As we observed in Sect. 5.2, the ITS algorithm is fairly robust to the choice of the parameters that it depends upon. From preliminary experiments, it was found that the number of TS iterations, I, was one of the most important parameters of the ITS approach. The results given in Sect. 5.4 for ITS were obtained with $I = 100$. To learn more about the influence of parameter I on the performance of the algorithm, we ran ITS with $I = 20$, 50, and 500.

Table 20 shows the comparison results of the four variations of the ITS algorithm. To distinguish between these variations, we use the same naming convention as in the previous section (thus, for example, ITS(20) stands for ITS with $I = 20$). The differences shown in Table 20 were obtained similarly to those in Tables 18 and 19. The last three rows of the table evaluate the tested configurations of the ITS algorithm versus ITS(100).

From the table, we note that replacing $I = 100$ with $I \in \{20, 50, 500\}$ makes the ITS algorithm less efficient in terms of the average solution quality obtained over 10 runs of ITS. Comparing the best results produced by the four versions of ITS, we found that ITS(500) performs equally well as ITS(100). In this respect, the other two variations (with $I = 50$ and especially $I = 20$) were worse than ITS with $I = 100$.

5.8.3 The effect of the population size on the MA performance

The only parameter of our memetic algorithm is the population size (denoted by z in Sect. 4). In our main experiments, we fixed this parameter to 100. To examine the influence of

Table 21 Difference in solution values between the MA algorithm with population size $z = 100$ and MA configurations with $z = 50$ and 500 for normalized cut graph partitioning

Graph	$F_{\text{MA(50)}} - F_{\text{MA(100)}}$		$F_{\text{MA(500)}} - F_{\text{MA(100)}}$	
	Best	Average	Best	Average
g-200-1	0	0.000234	0	−0.000762
g-200-2	0	0.001395	0	0
g-200-3	0	0.000909	0	0
g-200-5	0	0.002585	0	0
g-500-1	0	0.010216	0	−0.008061
g-500-2	0	0.004239	0.003148	−0.003125
g-500-3	0	0.004233	0	−0.007365
g-500-4	0	0.008290	0	−0.012044
g-500-5	0.000017	0.001054	−0.002601	−0.009748
celegansneural	0.020051	0.041856	−0.001311	−0.009233
ash292	0	0.002388	0	0
dwt-503	0	-0.000458	0	−0.003181
can-715	0	0.002027	0	−0.000499
L	−0.000050	0.005322	−0.000671	−0.000877
gplus_500	0	0.003246	0	−0.006613
delaunay_n12	0.007373	0.026521	0.489255	0.563219
3elt	−0.001851	−0.002526	0.573124	0.691849
uk	−0.059800	−0.196686	2.193496	2.124620
power	−0.264570	−0.300354	1.366916	1.267332
add32	0.110039	0.098538	1.028891	1.074933
Average	−0.009440	−0.014349	0.282512	0.283022
#wins	4	4	3	11
#ties	12	0	11	4
#losses	4	16	6	5

this parameter on the performance of the MA, we conducted additional experiments in which MA was run with z set to 50 and to 500. The results are summarized in Table 21. Its structure is similar to that of Table 20. We denote different configurations of MA by MA(z), $z \in \{50, 100, 500\}$.

A close inspection of Table 21 allows us to conclude that the behavior of MA depends on the graph size. For smaller graphs (first 15 rows of the table), the performance of MA increases with increasing z values. In this case, MA(100) is better than MA(50) (there are many positive entries in the second and third columns), and MA(500) is better than MA(100) (all but one entry in the last two columns and the first 15 rows are nonpositive). However, the picture is completely different for large graphs (last 5 graphs in the table). We see that MA(100) drastically outperforms MA(500). This can be explained by the fact that MA(500) takes a significant amount of time to create an initial population of individuals, which is 5 times larger than that of MA(100). The population is composed of solutions obtained by first randomly generating graph partitions and then improving them by applying the LS procedure. However, LS is not sufficiently fast for large graphs and random initial solutions. Since MA(500) initializes a large-sized population, the amount of time left for offspring generation is significantly reduced. Therefore, for large graphs, MA(500) produces a considerably smaller number of offspring than MA(100). For the `uk` graph, for example, on average, this number is 771, 1830 and 2496 for MA(500), MA(100), and MA(50), respectively. Producing fewer offspring leads to a reduction in the quality of solutions found by MA(500).

From the table, we can also see that the performance of MA(50) is comparable to that of MA(100). The latter obtains a smaller average objective function value for a larger number of graphs. On the other hand, the average values of F_n^* and $\bar{F}_n$ achieved by MA(50) are slightly smaller than those for MA(100). This advantage of MA(50) is due to the excellent performance of this version of MA for the `power` graph.

5.9 Analysis of the LS strategy

In this section, we discuss the impact of using vector S in our LS procedure on the performance of the memetic algorithm. The role of the vector S is to accelerate the neighborhood exploration process by evaluating only a subset of all possible relocation and swap moves. The vector S is used to mark the partition subsets that have been changed in the previous

Table 22 Difference in solution values between MA-SD and MA for normalized cut graph partitioning

Graph	$F_{\text{MA-SD}} - F_{\text{MA}}$		#impr
	Best	Average	
g-2000-1	0.024464	0.044930	9
g-2000-2	0.035619	0.043351	9
g-2000-3	0.073531	0.052916	10
g-2000-4	0.021778	0.047195	10
g-2000-5	0.051860	0.049714	9
g-3000-1	0.053193	0.070922	8
g-3000-2	0.116064	0.101844	10
g-3000-3	0.023612	0.064504	9
g-3000-4	0.086269	0.076046	10
g-3000-5	0.066354	0.087438	8
celegansneural	0	0.002400	1
delaunay_n11	0.002657	0.005041	9
data	0.001712	0.010269	10
delaunay_n12	0.016847	0.028090	10
3elt	0.002593	0.003916	9
uk	0.347104	0.169495	10
power	0.092182	0.096161	9
add32	0.008754	0.001819	5
road-minnesota	0.001258	0.009760	10
bio-dmela	0.001455	0.001423	5
Average	0.051365	0.048362	8.5

LS iteration. Then, in the current LS iteration, it is redundant to evaluate moves consisting of relocating a vertex from one unmarked subset to another unmarked subset. A similar observation can be made regarding swap moves.

In order to show the effectiveness of the proposed LS strategy, we experimentally compared the developed memetic algorithm against its version obtained by replacing the LS algorithm of Sect. 3 with a simplified LS procedure. The latter does not use the vector S. Its pseudocode is obtained from that in Algorithm 5 by removing initialization of S (Line 1), conditions imposed on S (Lines 4 and 20), and statements in Lines 16, 32, and 36. It is important to note that both LS implementations (i.e., that given in Algorithm 5 and the simplified procedure) return the same locally optimal solution. However, it is reasonable to expect that the former would be faster than the latter. We refer to the version of MA with S usage disabled in LS as MA-SD.

Table 22 compares the performance of MA and MA-SD on 10 random graphs and 10 benchmark graphs from our test suite. We provide results obtained for large graphs. For small graphs, the performance difference between MA and MA-SD is less significant. The second and third columns of the table provide the same kind of statistics as Table 21. The column labeled "#impr" gives the number of runs (out of 10) when MA yielded a better solution than MA-SD. In the remaining runs (if #impr < 10), MA and MA-SD produced the same solution. Figure 8 compares the objective function values achieved by MA and MA-SD for the two graphs, `g-3000-4` and `road-minnesota`.

Table 22 and Fig. 8 show that the proposed LS strategy leads to a significantly better performance of MA compared to the use of a traditional LS procedure. The reason behind this observation is that with a faster LS technique, the memetic algorithm is able to generate more offspring, which in many cases allows obtaining better results. As can be seen in the last column of Table 22, MA improved MA-SD solutions in all 10 runs for 8 of the 20 graphs. The average improvement rate is 85%. Thus, the proposed LS acceleration technique has a positive impact on the performance of the memetic algorithm.

6 Concluding remarks

The main focus of the paper was to investigate the capabilities of the most popular metaheuristic approaches when applied to ratio cut and normalized cut graph partitioning problems. Three specific algorithms were developed: multi-start simulated annealing, iterated tabu search, and memetic algorithms. In each of the algorithms, some optimization strategies were adopted. By calculating and updating the cut weights for the partitions efficiently, the time complexity of the inner loop of SA was significantly reduced. This allows more SA restarts to be performed. Both ITS and MA used a local search procedure. To speed up this procedure, we applied a technique that reduces the effort required for neighborhood examination. In the MA, we implemented a version of the crossover operator that was used in GGAs.

We carried out computational experiments on two sets of graphs. The results show that the MA is unequivocally superior to both MSA and ITS with respect to solution quality. This conclusion is valid for both models (ratio cut and normalized cut) and graph types (random and benchmark graphs from the literature). In some experiments, MA could find better or equally good solutions than ITS or MSA for all graphs in a given set. We also concluded that, in terms of the solution quality, MSA showed better performance than ITS for random graphs. However, there was no statistically significant difference between the results of these two algorithms for the considered set of benchmark graphs. Finally, from the results of our study, we concluded that ITS was the fastest algorithm in the comparison. It tends to find reasonable solutions more quickly than other tested algorithms. However, the MA is not unacceptably slow compared to ITS. For smaller graphs, MA is even faster than ITS and MSA. Additionally, we experimentally compared MA, which is our

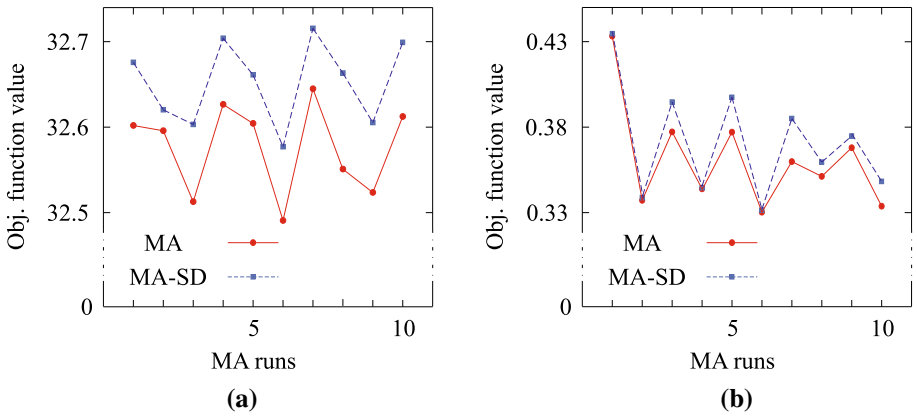


Fig. 8 Comparison between MA and MA-SD in terms of the objective function value. **a** `g-3000-4`. **b** `road-minnesota`

best algorithm, with the VNS algorithm of Hansen et al. [27] (which is the state-of-the-art algorithm for the normalized cut model). We found that MA provided better overall performance than VNS. Based on the obtained results, it is believed that the MA is an excellent approach to solving the ratio cut and normalized cut models.

There are promising directions for further research. One area of interest can be to develop innovative evolutionary algorithms for minimum cut problems. Some of the recent population-based metaheuristic algorithms can be used to solve these problems, such as monarch butterfly optimization [18,19], earthworm optimization algorithm [67], elephant herding optimization [66], moth search algorithm [65], slime mold algorithm [38], and Harris hawks optimization [29]. The proposed LS procedure could be embedded in such algorithms and used as a powerful technique for search intensification. Another promising way is to construct hybrid algorithms by combining two (or even more) metaheuristic methods to benefit from the strengths of each of them. Finally, an important avenue for further work is to apply the ideas of the ratio and normalized cut algorithms for developing metaheuristic-based techniques for solving other graph partitioning problems.

References

1. Aksoylar C, Qian J, Saligrama V (2017) Clustering and community detection with imbalanced clusters. IEEE Trans Signal Inf Process Netw 3(1):61–76
2. Bader DA, Kappes A, Meyerhenke H, Sanders P, Schulz C, Wagner D (2017) Benchmarking for graph clustering and partitioning. In: Alhajj R, Rokne J (eds) Encyclopedia of social network analysis and mining. Springer, New York. https://doi.org/10.1007/978-1-4939-7131-2_23
3. Banerjee S, Kayal D (2016) Detection of hard exudates using mean shift and normalized cut method. Biocybern Biomed Eng 36(4):679–685
4. Bektur G (2020) A multi-start iterated tabu search algorithm for the multi-resource agent bottleneck generalized assignment problem. Int J Optim Control Theor Appl 10(1):37–46
5. Brimberg J, Mladenović N, Urošević D (2015) Solving the maximally diverse grouping problem by skewed general variable neighborhood search. Inf Sci 295:650–675
6. Cafieri S, Hansen P, Mladenović N (2014) Edge-ratio network clustering by variable neighborhood search. Eur Phys J B 87:116
7. Cao B, Glover F, Rego C (2015) A tabu search algorithm for cohesive clustering problems. J Heuristics 21(4):457–477
8. Černý V (1985) Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm. J Optim Theory Appl 45(1):41–51
9. Chalupa D (2017) A memetic algorithm for the minimum conductance graph partitioning problem. arXiv preprint arXiv:1704.02854
10. Chalupa D, Hawick KA, Walker JA (2018) Hybrid bridge-based memetic algorithms for finding bottlenecks in complex networks. Big Data Res 14:68–80
11. Chan PK, Schlag MDF, Zien JY (1994) Spectral k-way ratio-cut partitioning and clustering. IEEE Trans Comput Aided Des Integr Circuits Syst 13(9):1088–1096
12. Chen X, Hong W, Nie F, He D, Yang M, Huang JZ (2018) Spectral clustering of large-scale data by directly solving normalized cut. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining (KDD 2018). ACM, London, pp 1206–1215
13. Chen X, Hong W, Nie F, Huang JZ, Shen L (2020) Enhanced balanced min cut. Int J Comput Vis 128:1982–1995
14. Chen X, Huang JZ, Nie F, Chen R, Wu Q (2017) A self-balanced min-cut algorithm for image clustering. In: IEEE international conference on computer vision (ICCV 2017). IEEE, Venice, Italy, pp 2080–2088. https://doi.org/10.1109/ICCV.2017.227
15. de Sousa VJR, Anjos MF, Le Digabel S (2019) Improving the linear relaxation of maximum k-cut with semidefinite-based constraints. EURO J Comput Optim 7(2):123–151
16. Dhillon IS, Guan Y, Kulis B (2007) Weighted graph cuts without eigenvectors: a multilevel approach. IEEE Trans Pattern Anal Mach Intell 29(11):1944–1957
17. Fan N, Pardalos PM (2012) Multi-way clustering and biclustering by the ratio cut and normalized cut in graphs. J Comb Optim 23(2):224–251
18. Faris H, Aljarah I, Mirjalili S (2018) Improved monarch butterfly optimization for unconstrained global search and neural network training. Appl Intell 48(2):445–464
19. Feng Y, Deb S, Wang GG, Alavi AH (2021) Monarch butterfly optimization: a comprehensive review. Expert Syst Appl 168:114418
20. Franzin A, Stützle T (2019) Revisiting simulated annealing: a component-based analysis. Comput Oper Res 104:191–206
21. Fu K, Gong C, Gu IYH, Yang J (2015) Normalized cut-based saliency detection by adaptive multi-level region merging. IEEE Trans Image Process 24(12):5671–5683

22. Gallego M, Laguna M, Martí R, Duarte A (2013) Tabu search with strategic oscillation for the maximally diverse grouping problem. J Oper Res Soc 64(5):724–734
23. Gallier J (2016) Spectral theory of unsigned and signed graphs. Applications to graph clustering: a survey. arXiv preprint arXiv:1601.04692
24. Glover F, Laguna M (1997) Tabu search. Kluwer Publisher
25. Hagen L, Kahng AB (1992) New spectral methods for ratio cut partitioning and clustering. IEEE Trans Comput Aided Des 11(9):1074–1085
26. Han J, Xiong K, Nie F (2017) Orthogonal and nonnegative graph reconstruction for large scale clustering. In: Proceedings of the twenty-sixth international joint conference on artificial intelligence (IJCAI 2017). Melbourne, pp 1809–1815
27. Hansen P, Ruiz M, Aloise D (2012) A VNS heuristic for escaping local extrema entrapment in normalized cut clustering. Pattern Recognit 45(12):4337–4345
28. He Y, Gong H, Xiong B, Xu X, Li A, Jiang T, Sun Q, Wang S, Luo Q, Chen S (2015) iCut: an integrative cut algorithm enables accurate segmentation of touching cells. Sci Rep 5:12089. https://doi.org/10.1038/srep12089
29. Heidari AA, Mirjalili S, Faris H, Aljarah I, Mafarja M, Chen H (2019) Harris hawks optimization: algorithm and applications. Future Gener Comput Syst 97:849–872
30. Hochbaum DS (2013) A polynomial time algorithm for Rayleigh ratio on discrete variables: replacing spectral techniques for expander ratio, normalized cut, and Cheeger constant. Oper Res 61(1):184–198
31. James TL, Brown EC, Keeling KB (2007) A hybrid grouping genetic algorithm for the cell formation problem. Comput Oper Res 34(7):2059–2079
32. Ji P, Zhang S, Zhou Z (2020) A decomposition-based ant colony optimization algorithm for the multi-objective community detection. J Ambient Intell Humaniz Comput 11(1):173–188
33. Jia H, Ding S, Du M, Xue Y (2016) Approximate normalized cuts without eigen-decomposition. Inf Sci 374:135–150
34. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
35. Krzystek P, Serebryanyk A, Schnörr C, Červenka J, Heurich M (2020) Large-scale mapping of tree species and dead trees in Šumava national park and Bavarian forest national park using lidar and multispectral imagery. Remote Sens 12:661. https://doi.org/10.3390/rs12040661
36. Lai X, Hao J-K (2016) Iterated maxima search for the maximally diverse grouping problem. Eur J Oper Res 254(3):780–800
37. Lai X, Hao J-K, Fu ZH, Yue D (2021) Neighborhood decomposition based variable neighborhood search and tabu search for maximally diverse grouping. Eur J Oper Res 289(3):1067–1086
38. Li S, Chen H, Wang M, Heidari AA, Mirjalili S (2020) Slime mould algorithm: a new method for stochastic optimization. Future Gener Comput Syst 111:300–323
39. Liu C, Liu Q (2018) Community detection based on differential evolution using modularity density. Information 9:218. https://doi.org/10.3390/info9090218
40. Liu X, Shen C, Guan X, Zhou Y (2019) Digger: detect similar groups in heterogeneous social networks. ACM Trans Knowl Discov Data 13(1):2. https://doi.org/10.1145/3267106
41. Lorente-Leyva LL, Herrera-Granda ID, Rosero-Montalvo PD, Ponce-Guevara KL, Castro-Ospina AE, Becerra MA, Peluffo-Ordóñez DH, Rodríguez-Sotelo JL (2018) Developments on solutions of the normalized-cut-clustering problem without eigenvectors. In: Huang T, Lv J, Sun C, Tuzikov AV (eds) Advances in neural networks-ISNN 2018, vol 10878. Lect Notes Comput Sci. Springer, pp 318–328
42. Lu H, Fu Z, Shu X (2014) Non-negative and sparse spectral clustering. Pattern Recognit 47(1):418–426
43. Lu Z, Hao J-K, Wu Q (2020) A hybrid evolutionary algorithm for finding low conductance of large graphs. Future Gener Comput Syst 106:105–120
44. Lu Z, Hao J-K, Zhou Y (2019) Stagnation-aware breakout tabu search for the minimum conductance graph partitioning problem. Comput Oper Resh 111:43–57
45. Ma L, Cheng S, Shi Y (2021) Enhancing learning efficiency of brain storm optimization via orthogonal learning design. IEEE Trans Syst Man Cybern Syst 51(11):6723–6742
46. Ma L, Huang M, Yang S, Wang R, Wang X (2021) An adaptive localized decision variable analysis approach to large-scale multiobjective and many-objective optimization. IEEE Trans Cybern. https://doi.org/10.1109/TCYB.2020.3041212
47. Merkurjev E, Bertozzi A, Yan X, Lerman K (2017) Modified Cheeger and ratio cut methods using the Ginzburg–Landau functional for classification of high-dimensional data. Inverse Probl. https://doi.org/10.1088/1361-6420/33/7/074003
48. Mu C, Zhang J, Liu Y, Qu R, Huang T (2019) Multi-objective ant colony optimization algorithm based on decomposition for community detection in complex networks. Soft Comput 23(23):12683–12709
49. Nascimento MCV, de Carvalho ACPLF (2011) Spectral methods for graph clustering—a survey. Eur J Oper Res 211(2):221–231
50. Nogueira B, Tavares E, Maciel P (2021) Iterated local search with tabu search for the weighted vertex coloring problem. Comput Oper Res 125:105087. https://doi.org/10.1016/j.cor.2020.105087
51. Palubeckis G, Karčiauskas E, Riškus A (2011) Comparative performance of three metaheuristic approaches for the maximally diverse grouping problem. Inf Technol Control 40(4):277–285
52. Palubeckis G, Ostreika A, Rubliauskas D (2015) Maximally diverse grouping: an iterated tabu search approach. J Oper Res Soc 66(4):579–592
53. Qiao Z, Zhang J, Qu X, Xiong J (2020) Dynamic self-organizing leader-follower control in a swarm mobile robots system under limited communication. IEEE Access 8:53850–53856
54. Ramos-Figueroa O, Quiroz-Castellanos M, Mezura-Montes E, Schütze O (2020) Metaheuristics to solve grouping problems: a review and a case study. Swarm Evol Comput 53:100643
55. Rodriguez FJ, Lozano M, García-Martínez C, González-Barrera JD (2013) An artificial bee colony algorithm for the maximally diverse grouping problem. Inf Sci 230:183–196
56. Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: Proceedings of the twenty-ninth AAAI conference on artificial intelligence. AAAI Press, Austin, pp 4292–4293
57. Rutenbar RA (1989) Simulated annealing algorithms: an overview. IEEE Circuits Devices Mag 5(1):19–26
58. Shi J, Malik J (2000) Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell 22(8):888–905
59. Shi X, Wu Y, Rao CR (2017) Consistent and powerful graph-based change-point test for high-dimensional data. Proc Natl Acad Sci U S A 114(15):3873–3878
60. Singh K, Sundar S (2019) A new hybrid genetic algorithm for the maximally diverse grouping problem. Int J Mach Learn Cybern 10(10):2921–2940
61. Tolić D, Antulov-Fantulin N, Kopriva I (2018) A nonlinear orthogonal non-negative matrix factorization approach to subspace clustering. Pattern Recognit 82:40–55
62. van Laarhoven PJM (1988) Theoretical and computational aspects of simulated annealing. Erasmus Universiteit Rotterdam, Rotterdam
63. Van Lierde H, Chow TWS, Chen G (2020) Scalable spectral clustering for overlapping community detection in large-scale networks. IEEE Trans Knowl Data Eng 32(4):754–767
64. von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416

65. Wang GG (2018) Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. Memet Comput 10(2):151–164
66. Wang GG, Deb S, Coelho LS (2015) Elephant herding optimization. In: Proceedings of the 2015 3rd international symposium on computational and business intelligence (ISCBI). IEEE, Bali, pp 1–5. https://doi.org/10.1109/ISCBI.2015.8
67. Wang GG, Deb S, Coelho LS (2018) Earthworm optimisation algorithm: a bio-inspired metaheuristic algorithm for global optimisation problems. Int J Bio-Inspired Comput 12(1):1–22
68. Wang L, Lu J (2019) A memetic algorithm with competition for the capacitated green vehicle routing problem. IEEE/CAA J Autom Sin 6(2):516–526
69. Wei Y-C, Cheng C-K (1991) Ratio cut partitioning for hierarchical designs. IEEE Trans Comput Aided Des Integr Circuits Syst 10(7):911–921
70. Yu SX, Shi J (2003) Multiclass spectral clustering. In: Proceedings of the ninth IEEE international conference on computer vision (ICCV'03), vol 1. IEEE, Nice, pp 313–319. https://doi.org/10.1109/ICCV.2003.1238361
71. Zevnik J, Kramar Fijavž M, Kozelj D (2019) Generalized normalized cut and spanning trees for water distribution network partitioning. J Water Resour Plan Manag 145(10):04019041. https://doi.org/10.1061/(ASCE)WR.1943-5452.0001100
72. Zhang R, Nie F, Li X (2017) Self-weighted spectral clustering with parameter-free constraint. Neurocomputing 241:164–170
73. Zheng S, Xu Z, Yang H, Song J, Pan Z (2019) Comparisons of different methods for balanced data classification under the discrete non-local total variational framework. Math Found Comput 2(1):11–28

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.