Received: xxx **Accepted:** xxx **Published:** xxx.
DOI. xxxxxxx
xxx Volume xxx, Issue xxx, (1-23)
Research Article
Open Access
Control and Optimization in Applied Mathematics - COAM

Solution Techniques for Fuzzy Graph Partitioning Based on Heuristic Optimization

Mohammed Alsaeedi[1], **Mostafa Tavakoli**[1]✉, **Ahmad Abouyee**[1], **Khatere Ghorbani Moghadam**[2], **Reza Ghanbari**[1]

[1] Faculty of Mathematical Sciences, Department of Applied Mathematics, Ferdowsi University of Mashhad, Mashhad, Iran.
[2]Mosaheb Institute of Mathematics, Kharazmi University, Tehran, Iran.

✉ **Correspondence:**
Mostafa Tavakoli
E-mail:
m_tavakoli@um.ac.ir

How to Cite
Alsaeedi, M., Tavakoli, M., Abouyee, A., Ghorbani Moghadam, Kh., Ghanbari R. (2025). "Solution techniques for fuzzy graph partitioning based on heuristic optimization", Control and Optimization in Applied Mathematics, 10(): 1-23, doi: 10.30473/coam.2025.74009.1296.

Abstract. In this study, we propose a novel graph partitioning problem where the edges are characterized by trapezoidal fuzzy numbers. A linear ranking function is employed to establish an order among these fuzzy numbers. We derive the necessary conditions for the existence of an optimal solution to this problem. To address the fuzzy graph partitioning problem, we implement and compare the performance of three algorithms: Genetic Algorithm, Tabu Search, and Sequential Least Squares Programming. The algorithms are evaluated based on objective values, computational time, and the number of iterations across multiple numerical examples. Utilizing Dolan-Moré performance profiles, we demonstrate the superiority of our proposed approach relative to existing methods. The findings highlight the robustness and computational efficiency of our methodology, making a meaningful contribution to the advancement of fuzzy graph algorithms and their practical applications.

Keywords. Fuzzy, Graph-partitioning, Fuzzy graph, Heuristic optimization, Fuzzy edge representation.

MSC. 90C59; 03E72.

https://mathco.journals.pnu.ac.ir

1 Introduction

Graph partitioning involves dividing a graph into two or more disjoint sets of vertices such that the total weight of the edges between these sets is minimized. When the weights of the edges are fuzzy, the resulting structure is known as a fuzzy graph. The concept of fuzzy graphs was first introduced by Kaufman et al. [24], building upon the fuzzy relations described by Zadeh [40]. Subsequently, Rosenfeld [36] formally defined fuzzy graphs as a generalization of classical, crisp graphs, where both vertices and edges are associated with membership degrees. Rosenfeld explored various properties of fuzzy graphs, including connectivity, paths, and cycles, based on these membership degrees. Since then, numerous studies have examined fuzzy graphs and applied them to various optimization problems [10, 26, 34].

In the context of graph partitioning, the classical problem of graph coloring is considered a subproblem, historically studied by Pieter and Mouton [32]. Since the early 1970s, graph partitioning has attracted extensive research interest. An innovative approach was introduced by Kernighan and Lin [25], who proposed a heuristic method to partition graph nodes optimally, minimizing the sum of the edge costs between partitions. Their method is computationally efficient and scalable, making it suitable for large-scale applications such as electronic circuit design. Given that graph partitioning problem (GPP) is an NP-hard problem, solutions are typically categorized into exact and heuristic methods. Exact methods, like branch-and-cut algorithms [7], column generation [22], and exact algorithms for weighted edges [20], aim to find optimal solutions but are computationally intensive. In contrast, heuristic approaches offer approximate solutions within reasonable timeframes, which are often more practical for real-world problems. Recent methods have included multi-level parallel algorithms [1] and hybrid approaches [37] incorporating fuzzy logic to predict points of sewer networks.

Heuristic algorithms such as Genetic Algorithms (GAs) have been widely employed for the GPP. Bui and Moon [8] introduced a hybrid GA for GPP, while Shazely et al. [38] developed GA-based techniques to generate multiple high-quality solutions, comparing different strategies like fitness sharing and deterministic crowding. Other studies have applied GAs to multi-objective partitioning problems [13] and integrated encoding schemes tailored for the GPP [5], with applications extending to clustering and machine learning [9, 29].

The Tabu Search (TS) algorithm is also a prominent method for the GPP. Kadluczka and Wala [23] utilized TS, along with GA, to solve a generalized version of GPP involving weighted graphs. Similarly, Lim and Chee [30] applied TS for balanced minimum cut problems, whereas Benlic and Hao [4] proposed a multilevel TS-based algorithm for balanced graph partitioning. Bruglieri and Cordone [6] introduced the minimum gap GPP (MGGPP), which aims to partition into connected subgraphs with equal or nearly equal weights, using a two-level TS and large neighborhood search for large-scale instances (up to 23,000 vertices).

In this paper, we extend the classical GPP to incorporate fuzzy edge weights represented by trapezoidal fuzzy numbers. To handle the fuzzy data, we employ a linear ranking function for ordering fuzzy numbers. We apply three optimization algorithms, the GA, the TS and Sequential Least Squares Programming (SLSQP), to solve this fuzzy GPP.

The remainder of this paper is organized as follows: Section 2 details the standard GPP with weighted edges, its formulation, and conditions for optimal solutions. Section 3 introduces the fuzzy GPP, the ranking function and the conditions for optimality. Section 4 describes the GA, including population initialization, crossover, and stopping criteria. Section 5 presents the TS algorithm is presented, focusing on local search procedures to improve solutions. Section 6 discusses the SLSQP algorithm and the rationale for its inclusion. Section 7 provides numerical results demonstrating the effectiveness of our proposed methods. Finally, Section 8 concludes the paper

2 Graph Partitioning

Consider a weighted graph, wherein edges are assigned weights. The objective of graph partitioning is to divide the vertex set into two or more disjoint subsets, subject to specific constraints, such that the total weight of edges crossing between different subsets is minimized. The problem is commonly referred to as the *minimum cut* problem. Let $G = (V, E)$ be a graph with vertex set V and edge set E, where V is partitioned into two disjoint subsets V_1 and V_2, satisfying $V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$. To solve the GPP, we introduce a vector $x = (x_1, x_2, \ldots, x_n)^T$ be a whose elements are binary variables, taking values in $0, 1$. Each component x_i corresponds to vertex v_i, such that:

$$V_1 = \{i : x_i = 1\}, \quad V_2 = \{i : x_i = 0\}. \tag{1}$$

Define the vector $e = (1, 1, \ldots, 1)^T$ of length n. Then, the scalar product $e^T x = m$ indicates that exactly m vertices belong to V_1, with the remaining $n - m$ vertices in V_2. Let $A = (a_{ij})$ be the adjacency matrix, where a_{ij} denotes the weight of the edge between vertices v_i and v_j. The GPP can then be expressed as a continuous optimization:

$$\begin{aligned} \min \quad & (e - x)^T (A + D) x \\ \text{s.t.} \quad & \\ & e^T x = m, \\ & 0 \le x \le e, \end{aligned} \tag{2}$$

where D is a diagonal matrix with entries determined based on A. The objective function $(e - x)^T Ax$ quantifies the total weight of edges between the two partitions, for any feasible solution of (2). An extension of the formulation of (2) incorporates bounds $[l, u]$ on the size of

V_1, i.e., $l \leq e^T x \leq u$, to constrain the cardinality of the subset V_1. The generalized continuous problem becomes:

$$\begin{aligned} \min \quad & (e-x)^T(A+D)x \\ \text{s.t.} \quad & \\ & l \leq e^T x \leq u, \\ & 0 \leq x \leq e. \end{aligned} \tag{3}$$

The subsequent theorem provides conditions under which the solution to this problem (3) is binary.

Theorem 1. [19]: If the diagonal matrix $D = \text{diag}(d_{11}, d_{22}, \ldots, d_{nn})$ satisfies the inequalities

$$d_{ii} + d_{jj} \geq 2a_{ij}, \quad d_{ii} \geq 0, \qquad i, j = 1, 2, \ldots, n, \tag{4}$$

then, the problem (3) admits a binary solution. Furthermore, if the inequalities in (4) hold strictly, i.e.,

$$d_{ii} + d_{jj} > 2a_{ij}, \quad d_{ii} > 0, \qquad i, j = 1, 2, \ldots, n, \tag{5}$$

then, every local optimal solution of the problem (3) is binary.

3 Fuzzy Graph Partitioning

In this section, we examine the GPP in the fuzzy context (Fuzzy GPP). Specifically, we consider the scenario where the edges of the graph are weighted with trapezoidal fuzzy numbers. To compare these fuzzy weights, we utilize a linear ranking function. Therefore, we begin by reviewing some fundamental concepts related to trapezoidal fuzzy numbers and their ranking methods, referencing [14, 15, 31]).

Definition 1. (Trapezoidal fuzzy numbers and ranking functions)
A fuzzy set $\tilde{A}$ on $\mathbb{R}$ is called a fuzzy number if it satisfies the following conditions:

1. $\tilde{A}$ is convex,
2. Its membership function $\mu_{\tilde{A}}$ is piecewise continuous,
3. There exist real numbers $a < b < c < d$ such that $\mu_{\tilde{A}}$ is increasing on $[a, b]$, equals to 1 on $[b, c]$, decreasing on $[c, d]$, and is zero elsewhere.

Notice 1. A trapezoidal fuzzy number $\tilde{A}$ is denoted by $(a_L, a_U, \alpha, \beta)$, where $[a_L, a_U]$ corresponds to its core, and $(a_L - \alpha, a_U + \beta)$ represents its support (see Figure 1).

Notice 2. We denote $\mathcal{F}(\mathbb{R})$ as the set of all fuzzy numbers.

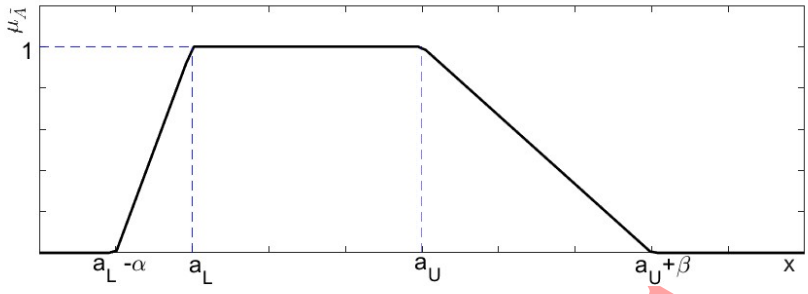


Figure 1: Trapezoidal fuzzy number $\tilde{A} = (a_L, a_U, \alpha, \beta)$.

Definition 2. (Operations on trapezoidal fuzzy numbers)
Let $\tilde{a} = (a_L, a_U, \alpha_1, \alpha_2)$ and $\tilde{b} = (b_L, b_U, \beta_1, \beta_2)$ be two trapezoidal fuzzy numbers. For a real number r, the scalar multiplication is defined as:

- If $r > 0$, then $r\tilde{a} = (ra_L, ra_U, r\alpha_1, r\alpha_2)$.
- if $r < 0$, then $r\tilde{a} = (ra_U, ra_L, -r\alpha_2, -r\alpha_1)$.

The addition of two fuzzy numbers is given by:

$$\tilde{a} + \tilde{b} = (a_L + b_L, a_U + b_U, \alpha_1 + \beta_1, \alpha_2 + \beta_2).$$

To facilitate comparison between fuzzy numbers, a crucial step in fuzzy partitioning, we employ ranking functions.

Definition 3. A ranking function $\mathcal{R} : F(\mathbb{R}) \to \mathbb{R}$ assigns a real value to each fuzzy number for comparison purposes. For any two fuzzy numbers $\tilde{a}$, $\tilde{b}$ in $F(\mathbb{R})$, the ordering relations are defined as:

$$\begin{aligned} \tilde{a} \geq_{\mathcal{R}} \tilde{b} &\iff \mathcal{R}(\tilde{a}) \geq \mathcal{R}(\tilde{b}), \\ \tilde{a} >_{\mathcal{R}} \tilde{b} &\iff \mathcal{R}(\tilde{a}) > \mathcal{R}(\tilde{b}), \\ \tilde{a} =_{\mathcal{R}} \tilde{b} &\iff \mathcal{R}(\tilde{a}) = \mathcal{R}(\tilde{b}). \end{aligned}$$

Similarly, the relations $\tilde{a} <_{\mathcal{R}} \tilde{b}$ and $\tilde{a} \leq_{\mathcal{R}} \tilde{b}$ are defined accordingly.

Remark 1. The zero trapezoidal fuzzy number $\tilde{0}$ is represented by $(0, 0, 0, 0)$.

Remark 2. For constants $c_L, c_U, c_\alpha, c_\beta$, at least one non-zero. For $\tilde{a} = (a_L, a_U, \alpha, \beta)$, the linear ranking function on $\mathcal{F}(\mathbb{R})$ is often defined as:

$$\mathcal{R}(\tilde{a}) = c_L a_L + c_U a_U + c_\alpha \alpha + c_\beta \beta.$$

A special case of the preceding linear ranking function has been proposed by Yager ([39]) as follows:

$$\mathcal{R}(\tilde{a}) = \frac{1}{2}\int_0^1 (\inf \tilde{a}_\mu + \sup \tilde{a}_\mu) d\mu,$$

which simplifies to (see [10, 1]):

$$\mathcal{R}(\tilde{a}) = \frac{a_L + a_U}{2} + \frac{\beta - \alpha}{4}. \tag{6}$$

For two trapezoidal fuzzy numbers $\tilde{a} = (a_L, a_U, \alpha_1, \alpha_2)$ and $\tilde{b} = (b_L, b_U, \beta_1, \beta_2)$, the relation $\tilde{a} >_{\mathcal{R}} \tilde{b}$ holds if and only if :

$$2(a_L + a_U) + \alpha_2 - \alpha_1 > 2(b_L + b_U) + \beta_2 - \beta_1.$$

The GPP with Trapezoidal Fuzzy Numbers

When the edges of the graph $G = (V, E)$ are represented with fuzzy weighted, the partitioning problem can be formulated as follows:

$$\begin{aligned} \min \quad & \tilde{f}(x) =_{\mathcal{R}} (e - x)^T (\tilde{A} + \tilde{D}) x \\ \text{s.t.} \quad & \\ & l \leq e^T x \leq u, \\ & 0 \leq x \leq e, \end{aligned} \tag{7}$$

where $\tilde{A} = (\tilde{a}_{ij})$ is the fuzzy adjacency matrix with trapezoidal fuzzy numbers $\tilde{a}_{ij}$ for $i, j = 1, 2, \ldots, n$. Additionally, $\tilde{D} = \text{diag}(\tilde{d}_{11}, \tilde{d}_{22}, \ldots, \tilde{d}_{nn})$ is a diagonal fuzzy matrix, with entries defined as trapezoidal fuzzy numbers depending on $\tilde{A}$.

Next, we present a theorem that converts the FPP described in (7) into an equivalent crisp formulation, utilizing the ranking function introduced in Equation (6).

Theorem 2. The fuzzy problem (7) is equivalent to the following crisp optimization problem:

$$\begin{aligned} \min \quad & f(x) = (e - x)^T (\mathcal{R}(\tilde{A}) + \mathcal{R}(\tilde{D})) x \\ \text{s.t.} \quad & \\ & l \leq e^T x \leq u, \\ & 0 \leq x \leq e, \end{aligned} \tag{8}$$

where $\mathcal{R}$ is a linear ranking function as defined in (6), and

$$\mathcal{R}(\tilde{A}) = (\mathcal{R}(\tilde{a}_{ij}))_{(i,j)}, \quad i, j = 1, 2, \ldots, n.$$

Proof. By the definition of the ranking operation, $\tilde{f}(x) =_{\mathcal{R}} (e - x)^T (\tilde{A} + \tilde{D}) x$ is equivalent to

$$f(x) := \mathcal{R}(\tilde{f}(x)) = \mathcal{R}((e-x)^T(\tilde{A}+\tilde{D})x).$$

Since $\mathcal{R}$ is a linear operator, f simplifies as:

$$f(x) = (e-x)^T\mathcal{R}(\tilde{A}+\tilde{D})x = (e-x)^T(\mathcal{R}(\tilde{A})+\mathcal{R}(\tilde{D}))x,$$

which completes the proof. □

Building on this, Theorem 1 can be extended for the FPP as follows:

Theorem 3. Suppose the diagonal matrix $\tilde{D} = \text{diag}(\tilde{d}_{11}, \tilde{d}_{22}, \ldots, \tilde{d}_{nn})$ is chosen such that the following inequalities hold:

$$\tilde{d}_{ii} + \tilde{d}_{jj} \geq_{\mathcal{R}} 2\tilde{a}_{ij}, \quad \tilde{d}_{ii} \geq_{\mathcal{R}} \tilde{0}, \quad i,j = 1,2,\ldots,n, \tag{9}$$

then, the problem described in (7) admits a binary solution. Furthermore, if these inequalities in (9) are strict, i.e.,

$$\tilde{d}_{ii} + \tilde{d}_{jj} >_{\mathcal{R}} 2\tilde{a}_{ij}, \quad \tilde{d}_{ii} >_{\mathcal{R}} \tilde{0}, \quad i,j = 1,2,\ldots,n, \tag{10}$$

then each local optimal solution of (7) will be a binary.

Proof. By Theorem 2, the problem (8)(its crisp equivalent) is equivalent to the fuzzy problem (7). Consequently, we can replace the original matrices A and D in the problem (3) with their ranked counterparts $\mathcal{R}(\tilde{A})$ and $\mathcal{R}(\tilde{D})$, respectively. This results in the conditions (3):

$$\mathcal{R}(\tilde{d}_{ii}) + \mathcal{R}(\tilde{d}_{jj}) \geq 2R(\tilde{a}_{ij}), \quad \mathcal{R}(\tilde{d}_{ii}) \geq 0, \quad i,j = 1,2,\ldots,n. \tag{11}$$

which, due to the linearity of $\mathcal{R}$, the conditions in (11) can be translate into

$$\mathcal{R}(\tilde{d}_{ii} + \tilde{d}_{jj}) \geq \mathcal{R}(2\tilde{a}_{ij}), \quad \mathcal{R}(\tilde{d}_{ii}) \geq 0, \quad i,j = 1,2,\ldots,n,$$

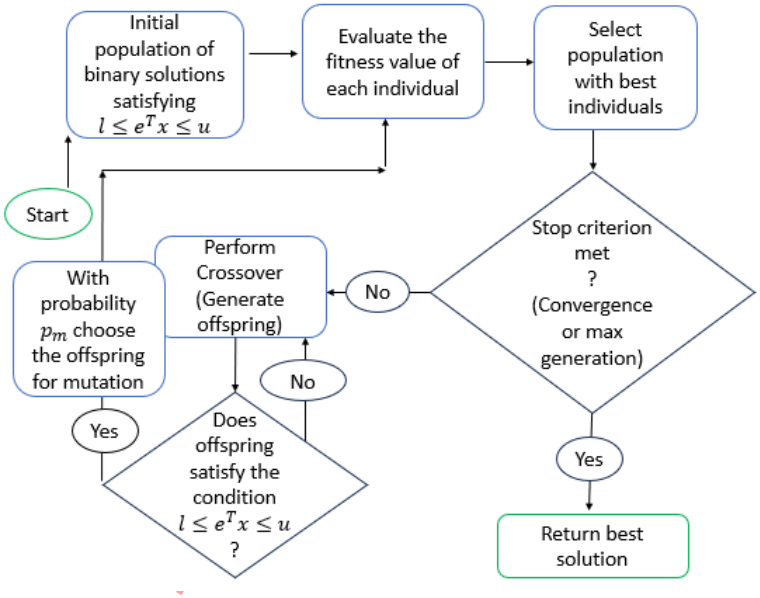
inducing conditions (9). Similarly, conditions (9) are proved. □

4 Genetic Algorithm (GA) Overview

The Genetic Algorithm (GA) is an optimization technique inspired by the principles of natural selection and evolution. It effectively addresses complex optimization problems by iteratively generating and assessing candidate solutions, mimicking the evolutionary process observed in nature. In a GA, a population of candidate solutions, referred to as individuals, is initialized randomly. Each individual is evaluated using a fitness function, which quantifies its quality. The most fit individuals are then selected to produce new offspring through genetic operations

such as crossover and mutation. These offspring inherit traits from their parents, and the process repeats over multiple generations, allowing the population to progressively improve and converge toward an optimal solution [18, 33].

This approach is particularly well-suited for our problem, as it excels in optimizing combinatorial problems with constraints such as $e^T x = m$, which are central to our formulation. The search space, of size $\binom{n}{m}$, is vast, making deterministic or gradient-based methods less effective due to the discrete binary nature of variables and the constraints involved. GA naturally respects these constraints through careful initialization and crossover operations that maintain feasibility. Moreover, the quadratic nature of the objective function, $(e - x)^T Ax$, can exhibit multiple local minima. The stochastic nature of GA allows it to explore diverse solutions in parallel, helping to avoid entrapment in local minima. A customized crossover mechanism ensures that offspring remain valid solutions, while the population-based search systematically explores the solution space for improvements. Overall, GA's ability to handle problem complexity, discrete variables, and constraints makes it a robust and efficient optimization choice. A flowchart illustrating the GA process is shown in Flowchart 2.



Flowchart 2: A flowchart of the GA.

4.1 Problem Formulation

Now consider the following optimization problem:

$$\begin{aligned} \min \quad & f(x) = (e-x)^T(\mathcal{R}(\tilde{A}) + \mathcal{R}(\tilde{D}))x \\ \text{s.t.} \quad & \\ & e^T x = m, \\ & 0 \le x \le e. \end{aligned} \tag{12}$$

where $\bar{A} = \mathcal{R}(\tilde{A}) + \mathcal{R}(\tilde{D})$. The fuzzy diagonal matrix $\tilde{D}$ can be designed so that $\mathcal{R}(\tilde{D})$ satisfies the relevant conditions (10) and ensures $\bar{A}$ is diagonally dominant matrix. Recall that a square matrix is diagonally dominant if the magnitude of each diagonal element is at least equal to the sum of the magnitudes of the other (non-diagonal) elements in the same row.

4.2 GA Implementation for the Problem

To apply GA, we start by creating an initial population $G = x^0, x^1, \ldots, x^N$ consisting of potential solutions with binary elements, generated randomly. To satisfy the constraints $\sum x_i = m$ and $0 \le x_i \le 1$, we set exactly m elements to 1 and the remaining n–m to 0 in each individual. The fitness of each candidate x is evaluated using the objective function:

$$f(x) = (e-x)^T \bar{A} x.$$

Individuals with the lowest fitness values are selected for the next generation. Crossover is performed by randomly choosing two parent solutions $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$. The sum $z = x + y$ is computed, and offspring are generated by averaging and applying random modifications:

i. Set $w = \lceil z/2 \rceil$,

ii. Randomly select s–m components of w where the value is 1 and set them to 0 in which $s = \|w\|_1$,

iii. Repeat this process to generate multiple offspring until the offspring meet $\left\lceil \frac{s}{s-m} \right\rceil$.

Each offspring's fitness is evaluated, and the best solutions are combined with selected individuals from the previous population to form the new generation. The process continues until the improvement falls below a threshold ϵ.

Algorithm 1 represents the GA structure for solving the problem (12).

4.3 Adaptation for Problem (8)

By modifying some steps slightly, this GA can also be used to solve the alternative formulation (8). Algorithm 2 describes this adapted process.

Algorithm 1 Solving the FPP (12) by Genetic Algorithm (GA)

Input: Matrix $\bar{A} = \mathcal{R}(\tilde{A}) + \mathcal{R}(\tilde{D})$.

Parameters:
- Population size: N,
- Maximum generations: T,
- Mutation probability: p_m,
- Crossover probability: p_c,
- Convergence threshold: ϵ.

Output: Near-optimal solution $x = (x_1, x_2, \ldots, x_n)$ of (12) and objective value $f(x) = (e - x)^T \bar{A} x$.

1. **Initialization:** Randomly Generate initial population $G = \{x^0, x^1, \ldots, x^N\}$ with binary entries with satisfying constraints in (12).
2. **Fitness Evaluation and Selection:**
   - 2-1: For each $x \in G$, evaluate $f(x) = (e - x)^T \bar{A} x$.
   - 2-2: Select $\lfloor p_c \cdot N \rfloor$ chromosomes for crossover based on their fitness using tournament or roulette wheel selection.
3. **Crossover:** For each pair of selected parents $x, y \in G$:
   - 3-1: Compute $z = x + y$. Set $t_i = \lceil z_i/2 \rceil$ for $i = 1, \ldots, n$.
   - 3-2: Calculate $s = \sum_{i=1}^{n} t_i$.
   - 3-3: Generate offspring z^i by randomly turning $(s - m)$ selected 1s into zeros.
   - 3-4: Add feasible offspring z^i to the new population G.
   - 3-5: Repeat Steps 3 and 4 to generate at most $\left\lceil \frac{s}{s-m} \right\rceil$ offspring.
4. **Mutation:**
   - 4-1: For each chromosome in G, with probability p_m:
     - 4-1-1: Randomly select two indices i and j such that $x_i = 1, x_j = 0$
     - 4-1-2: Swap a randomly chosen 1 with a 0.
   - 4-2: Ensure constraint satisfaction (12) after mutation.
5. **Replacement:** Form new population of size N by combining top offspring and elite individuals.
6. **Termination:** Stop if the difference between consecutive best fitness values is less than ϵ or if T maximum generations are reached.

Algorithm 2 Solving the FPP (8) via the GA

Input: Matrix $\bar{A} = \mathcal{R}(\tilde{A}) + \mathcal{R}(\tilde{D})$.

Output: Optimal solution $x = (x_1, x_2, \ldots, x_n)$ with objective $f(x) = (e - x)^T \bar{A} x$.

1. **Initial Population** G**:**

   1-1: Find the $u - l + 1$ feasible binary solutions $G^0 = \{x^{(1)}, \ldots, x^{(u-l+1)}\}$.

   1-2: Generate additional solutions by swapping components of solutions in G^0, ensuring feasibility.

   1-3: Combine to form $G = G^0 \cup G^1$.

2. Apply selection, crossover, mutation, replacement, and termination procedures similar to Algorithm 1 with necessary adjustments as per problem specifics.

5 Tabu Search (TS)

Tabu Search (TS) is a metaheuristic optimization algorithm tailored for solving complex combinatorial problems. It iteratively explores the solution space by moving to the best neighboring solutions, even if such moves temporarily worsen the current solution. This strategic acceptance helps avoid entrapment in local minima. A key feature of the TS is its memory-based tabu list, which records recently visited solutions or moves to prevent revisiting them, thereby promoting diversification of the search. The TS is renowned for its high flexibility, effective constraint handling, and strong performance on large-scale and intricate problems.

The TS is particularly appropriate for our minimization problem because it efficiently navigates the vast solution space of the graph, actively avoiding cycles through its tabu list. It manages the combinatorial constraints $l \leq \sum x_i \leq u$ effectively by concentrating the search on feasible neighboring solutions. Its capability to optimize nonlinear objective functions, specifically, $(e - x)^T Ax$, while escaping local optima makes it highly suitable. Additionally, its scalability and customizable stopping criteria make it an excellent choice for large, graph-based optimization challenges. A flowchart illustrating the TS process is provided in Flowchart 3.

Over time, the TS has been extensively adapted, and numerous variants and improvements have been developed. The following Algorithm 3 outlines the core steps of the classical TS approach.

A visual representation of the TS process is shown in Flowchart 3.

Algorithm 3 Tabu Search Algorithm for solving the FPP (8)

Input: Matrix $\bar{A} \in \mathbb{R}^{n\times n}$ (symmetric), lower bound l, upper bound u, maximum iterations max_iter, tabu list size $tabu\_size$.

Output: A near-optimal solution x^{best} and its objective function value $f(x^{best}) = (e - x^{best})^T \bar{A} x^{best}$.

1. **Initialization:**

   1-1: Set $e \leftarrow \mathbf{1}_n$ (vector of ones with length n).

   1-2: Generate an initial feasible solution $x^{best} = (x_1^{best}, x_2^{best}, \ldots, x_n^{best}) \in \{0,1\}^n$ satisfying $l \leq \sum x_i^{best} \leq u$.

   1-3: Compute initial cost $Best_{cost} \leftarrow f(x^{best})$.

   1-4: Initialize an empty tabu list.

2. **Main Loop:** (repeat for max_iter iterations):

   2-1: For each index $i = 1, \ldots, n$, create a neighbor $x^{new} = (x_1^{new}, x_2^{new}, \ldots, x_n^{new})$ by flipping the component x^{best}, i.e., set $x_i^{new} \leftarrow 1 - x_i^{best}$, keeping other components unchanged.

   2-2: Check if x^{new} respects the constraints $l \leq \sum x_i^{new} \leq u$.

   2-3: Evaluate the objective function $f(x^{new})$.

   2-4: Among all valid neighbors, select the one with the lowest $f(x^{new})$, denoted as $x_{best\_neighbor}$.

3. **Update:**

   3-1: If $x_{best\_neighbor}$ is not in the tabu list or has a better cost than x^{best}:

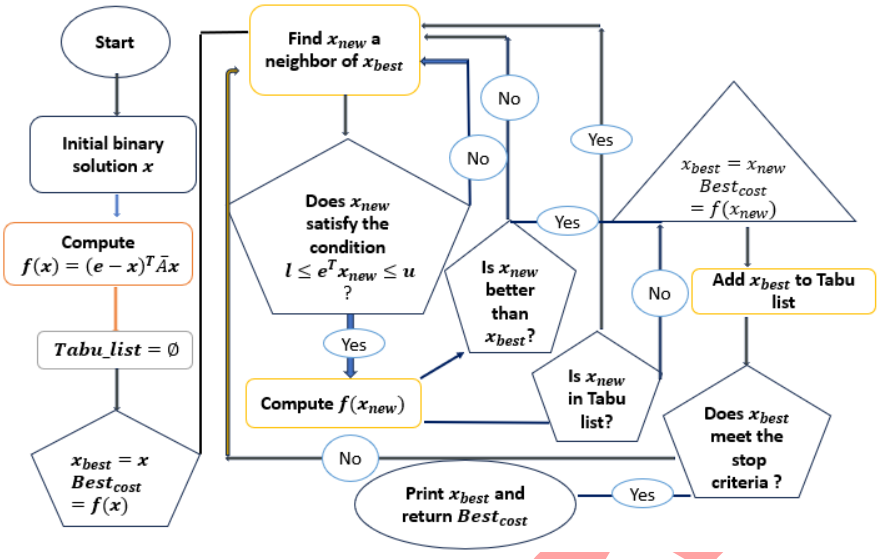
   Update $x^{best} \leftarrow x_{best\_neighbor}$.

   Update $Best_{cost} \leftarrow f(x^{best})$.

   3-2: Add x^{best} to the tabu list.

   3-3: If tabu list exceeds size $tabu\_size$, remove the oldest entry.

4. **Termination:** Return x^{best} and $Best_{cost}$.



Flowchart 3: Flowchart of the TS procedure.

6 Sequential Least Squares Programming (SLSQP) Algorithm

Sequential Least Squares Programming (SLSQP) is a gradient-based optimization method tailored for solving constrained nonlinear optimization problems. It approaches the original problem by iteratively solving a sequence of quadratic programming (QP) subproblems that approximate the nonlinear problem at each step. The key steps are as follows:

- The objective function and constraints are locally approximated using a quadratic models.
- At each iteration, a quadratic subproblem is solved to determine a search direction.
- The solution is updated using a line search method, and the process repeats until convergence criteria, such as gradient norm or constraint satisfaction, are met.

In this study, we employed SLSQP as a benchmark technique for solving the partitioning problem defined in equation (8). The algorithm was implemented using the *"scipy.optimize.minimize"* function in Python. The initial solution x was randomly initialized within $[0, 1]$, and the problem constraints were directly supplied to the SLSQP solver. During optimization, the objective function was minimized iteratively, with each iteration monitored via a callback function to track progress.

Upon convergence, the continuous solution x^* was thresholded such that any $x_i^* > 0.6$ was set to 1, while and the rest were set to 0, yielding a binary partitioning decision. The indices where $x_i^* = 1$ identified the selected subset of nodes. Given that the optimization problem (8) is a quadratic programming problem with linear constraints, SLSQP is particularly suitable for this context for several reasons:

- The objective function in (8) is quadratic, a form efficiently handled by the SLSQP.
- The linear constraints $l \leq e^T x \leq u$ and $x \geq 0$ can be directly incorporated without complex transformations.
- The SLSQP is computationally effective for medium-scale constrained optimization problems, making it a practical choice.
- The non-negativity constraint $x \geq 0$ aligns naturally with SLSQP's capacity to enforce inequality constraints. The constraints $l \leq e^T x \leq u$ and $x \geq 0$.
- The SLSQP maintains feasibility by solving quadratic subproblems that respect the constraints at each iteration, ensuring stable convergence.

However, it is important to note that the SLSQP may become trapped in a local optimum, especially when the problem's objective function (8) is non-convex. The convexity of the quadratic objective depends on the properties of the matrix $\bar{A}$. Specifically:

- If $\bar{A}$ is positive semidefinite (all eigenvalues are non-negative), the objective function is convex, and the SLSQP is guaranteed to converge to the global optimum.
- Conversely, if $\bar{A}$ is indefinite (has both positive and negative eigenvalues), the function is non-convex. In such cases, multiple local minima may exist, and the SLSQP might converge to a local, rather than a global, minimum.

7 Numerical Experiments

In this section, we present numerical experiments to validate the theoretical results discussed earlier. All computations were performed using R version 4.4.2 on a system equipped with an Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz (up to 1.80GHz). The Irace package was employed alongside R to optimize the algorithm parameters.

Example 1. We consider the graph with 20 nodes ([19]), illustrated in Figure 4 . Details of the graph structure are provided in Table 1.

The fuzzy adjacency matrix $\tilde{A} = (\tilde{a}ij)$ assigns a fuzzy number $(2, 4, 1, 1)$ to all connected edges, indicating the strength of the connection. For non-connected pairs, the entry is $(0, 0, 0, 0)$. The fuzzy diagonal entries are set as $\tilde{d}ii = [2, 12, 1, 1]$ for each $i = 1, 2, \ldots, 20$. Using the ranking function defined in equation (6), the conditions in (10) are satisfied. We set the lower and upper bounds as $l = 3$ and $u = 5$.

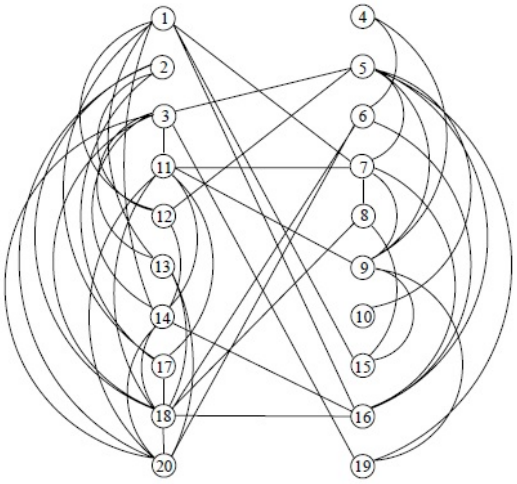


Figure 4: Sample graph for Example 1 [19].

Table 1: Graph structure for Example 1.

Node	Connects to	Node	Connects to
1	7, 12, 13, 14, 15, 16, 17	9	11, 15, 19
2	12, 17, 18, 20	11	14, 17, 18, 20
3	5, 11, 13, 14, 18, 19, 20	12	14
4	6,9	13	18,20
5	7, 9, 10, 12, 16, 19	14	16,18,20
6	16,18,20	16	18
7	8, 9, 11, 16	17	18
8	15, 18	18	20

Both the GA and TS produced the following partition:

$$x = [0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0],$$

with an objective value of 15. Here, the subset $V_1 = [8, 10, 15]$, while V_2 contains the remaining nodes:

$$V_2 = [1, 2, 3, 4, 5, 6, 7, 9, 11, 12, 13, 14, 16, 17, 18, 19, 20].$$

Results are summarized in Table 2. Using Irace for parameter tuning, the initial population for the GA comprised 20 chromosomes over 5 generations. For the TS, the tabu list stored the last 10 moves.

If we set the lower and upper bounds as $l = u = 10$, this means the condition $l \leq e^T x \leq u$ reduces to $e^T x = 10$. The results under this setting are presented in Table 3. Using the Irace package for parameter tuning, the initial population for the GA consisted of 50 chromosomes over 30 generations. For the TS, the tabu list stored the last 5 moves or solutions visited.

Table 2: Graph partitioning results achieved by the GA and the TS algorithms for Example 1.

Method	Objective Value	V_1	CPU Time (s)	Iterations	Population
GA	15	[8,10,15]	0.0156	2	20
TS	15	[8,10,15]	0.014	11	——

Table 3: Graph partitioning results for Example 1 with fixed bounds $l = u = 10$.

Method	Objective Value	V_1	CPU Time (s)	Iterations	Population
GA	39	[1, 2, 3, 5, 10, 12, 13, 16, 17, 19]	0.078	2	50
TS	39	[1,5,7,8,9,10,12,15,16,19]	0.008	6	—

Example 2. We consider a graph with 1,000 vertices, where 20% of the edges are randomly assigned nonzero weights. In the fuzzy adjacency matrix $\tilde{A} = (\tilde{a}ij)$, if there is a connection between nodes i and j, the corresponding fuzzy number $\tilde{a}ij = ((a_L)ij, (a_U)ij, \alpha_{ij}, \beta_{ij})$ is randomly selected from a uniform distribution over $(0, 1)$. If no connection exists, $\tilde{a}ij$ is set to the fuzzy number $(0, 0, 0, 0)$. The fuzzy diagonal entries are set as $\tilde{d}ii = [2, 4, 0.5, 0.5]$ for $i = 1, 2, \ldots, 1000$. Using the ranking function defined in (6), the conditions in (10) are satisfied. The lower and upper bounds are set to $l = 3$ and $u = 5$.

The results obtained using the GA, the TS, and the SLSQP [28] are summarized in Table 4. For the GA, the initial population consists of 800 individuals, and the top 600 are selected for the next generation. The SLSQP method starts from a randomly generated vector uniformly distributed in $(0, 1)$. The TS algorithm uses a tabu list size of 50. All three methods achieve nearly the same objective function value, approximately 226. Notably, the execution time of TS is significantly lower than that of the SLSQP and the GA. The final results of GA with different population sizes are also presented, with the third column indicating the set V_1 as defined in (1).

Table 4: Graph partitioning results for Example 2 using GA, TS, and SLSQP.

Method	Objective Value	V_1	CPU Time (s)	Iterations	Population
GA	226.12	[73, 231, 529]	40.73	4	800
TS	226.04	[231, 527, 529]	2.67	11	—
SLSQP	226.04	[231, 527, 529]	81.87	8	—

Example 3. We consider a graph with 1,500 vertices, where 30% of the edges are randomly assigned nonzero weights. In the fuzzy adjacency matrix $\tilde{A} = (\tilde{a}ij)$, if there is a connection between nodes i and j, the fuzzy number $\tilde{a}ij = ((a_L)ij, (a_U)ij, \alpha_{ij}, \beta_{ij})$ is randomly generated from a uniform distribution over $(0, 1)$. For non-connected pairs, $\tilde{a}_{ij}$ is set to $(0, 0, 0, 0)$.

The fuzzy diagonal elements are set as in Example 2, and the bounds l and u are also taken from that example.

The results of applying the GA, the TS, and the SLSQP are summarized in Table 5. For the GA, the initial population is randomly selected from 1,000 individuals, with the top 700 advancing to the next generation. The initial guess for the SLSQP is a random vector uniformly distributed in $(0, 1)$. The TS algorithm uses a tabu list of size 50. The objective function values obtained by all three algorithms are nearly identical. However, it is observed that the SLSQP requires significantly more CPU time compared to the GA and the TS.

Table 5: Graph partitioning results for Example 3 using GA, TS, and SLSQP.

Method	Objective Value	V_1	CPU Time (s)	Iteration	Population
GA	578.03	[113, 260, 665]	155.29	5	1000
TS	578.03	[113 260 665]	10.85	11	—
SLSQP	578.03	[113 260 665]	689.05	15	—

Example 4. To assess the effectiveness of the proposed fuzzy graph partitioning method, we utilize a set of test problems sourced from the Matrix Market repository http://www.cise.ufl.edu/research/sparse/matrices/. Each test case involves a symmetric, positive-definite matrix, which can be interpreted as the adjacency matrix of a graph. Although the original matrices are not fuzzy, we introduce trapezoidal fuzzy numbers to preserve the linear ranking functions defined in Equation (6), ensuring the original crisp values are maintained.

Furthermore, to satisfy the conditions specified in Equation (10), we consider the diagonal matrix $\mathcal{R}(\tilde{D})$ with diagonal entries strictly greater than the maximum magnitude of the non-diagonal entries in the adjacency matrix $\mathcal{R}(\tilde{A})$.

The fuzzy graph partitioning problem outlined in Equation (8) is solved using three different optimization approaches: the GA, the TS, and the SLSQP. Their performance is evaluated based on the objective function values and computational efficiency, with the CPU time recorded for each method to facilitate comparison.

Table 6 summarizes the objective function values obtained for each test problem, including matrix identities, the number of nodes, edges, and connected components. The results show that both the GA and the TS consistently attain lower objective values than the SLSQP across all tested problems.

Table 7 reports the CPU times for each algorithm. The findings indicate that the TS generally outperforms both the GA and the SLSQP in terms of computational efficiency, achieving substantially shorter execution times across all problem instances. While the GA produces objective values comparable to the TS, it involves higher computational costs, especially for large matrices. The SLSQP exhibits the longest runtimes, particularly for high-dimensional cases, which reduces its practicality for large-scale datasets.

In summary, the experimental results demonstrate that the TS offers the best compromise between solution quality and computational time. Although the GA yields similar objective values, it demands significantly more computational resources. The SLSQP, despite being a gradient-based method, struggles with efficiency as the problem size increases. These insights suggest that the TS is a more suitable and scalable approach for addressing fuzzy graph partitioning problems in large datasets.

Table 6: Comparison of objective function values across different test problems.

Name	# Nodes	# Edges	# Connected Components	GA	TS	SLSQP
Trefethen-20b	19	83	1	12	13	23
ash85	85	304	1	6	6	9
Journals	124	6096	1	421	421	467
Trefethen-150	150	1095	1	21	21	57
Trefethen-200b	199	1536	1	21	21	70
Trefethen-200	200	1545	1	21	21	74
lshp-265	265	1009	1	7	14	14
mesh3e1	289	833	1	3	12	10
ash292	292	1250	1	6	10	9
Trefethen-300	300	2489	1	24	24	64
lshp-406	406	1561	1	7	13	16
Trefethen-500	500	4489	1	24	24	78
lshp-577	577	2233	1	7	7	16
Trefethen-700	700	6677	1	29	27	88
lshp-778	778	3025	1	8	8	12
G4	800	19976	1	92	92	92
G43	1000	10990	1	24	24	25
Trefethen-2000	2000	21953	1	38	30	67

Figure 5 illustrates the Dolan-Moré performance profile, which shows, for each $\tau \in \mathbb{R}$ (as defined in [11]), the proportion of test problems for which each algorithm's execution time is within a factor τ of the best. The profile indicates that the TS consistently outperforms the GA in terms of computational efficiency, achieving significantly faster runtimes across all tested instances.

Table 7: Comparison of CPU times for some test problems.

Name	GA	TS	SLSQP
Trefethen-20b	0.141	0.003	0.4
ash85	0.82	0.009	0.03
Journals	20	0.4	23
Trefethen-150	4.89	0.039	0.10
Trefethen-200b	2.14	0.07	0.38
Trefethen-200	5.32	0.04	0.38
lshp-265	8.10	0.036	0.54
mesh3e1	6.54	0.025	0.83
ash292	2.32	0.100	0.73
Trefethen-300	2.42	0.12	0.81
lshp-406	4.07	0.073	1.28
Trefethen-500	9.28	0.21	2.53
lshp-577	4.64	0.30	4.64
Trefethen-700	16.39	0.50	7.07
lshp-778	5.65	0.65	8.89
G4	22.62	0.86	31.37
G43	19.90	4.31	51.98
Trefethen-2000	248.73	27.59	2001.20

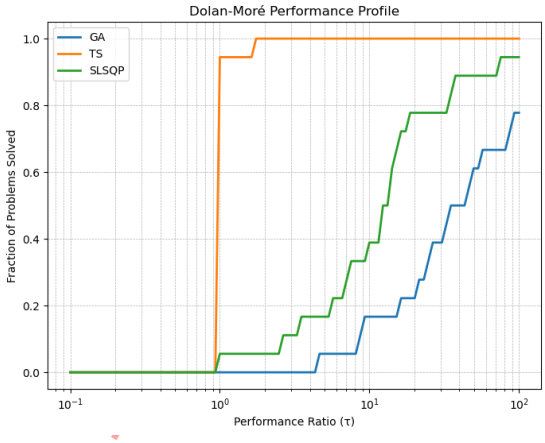


Figure 5: Dolan-Moré performance profile for comparing the CPU times.

8 Conclusion

This study presented a novel approach to addressing the graph partitioning problem with trapezoidal fuzzy edges utilizing a linear ranking function to facilitate the process. We applied three optimization algorithms, Genetic Algorithm (GA), Tabu Search (TS), and Sequential Least

Squares Programming (SLSQP), to conduct a thorough comparison based on objective values, computational efficiency, and the number of iterations. The use of Dolan-Moré performance profiles further highlighted the robustness and the effectiveness of the proposed approach. The findings confirm that our approach offers a competitive and efficient solution for fuzzy graph partitioning, contributing to advancement of fuzzy optimization techniques.

Declarations

Availability of Supporting Data

All data generated or analyzed during this study are included in this published paper.

Funding

The first, second, third and fifth authors thank the Ferdowsi University of Mashhad. The fourth author acknowledges support from the Mosaheb Institute of Mathematics, Kharazmi University, for this work.

Competing Interests

The authors declare that they have no competing interests relevant to the content of this paper.

Authors' Contributions

The main text of manuscript is collectively written by the authors.

References

[1] Abouyee Mehrizi, A., Ghanbari, R., Sadeghi, S., Ghorbani-Moghadam, Kh. (2024). "Solving continuous quadratic programming for the discrete balanced graph partitioning problem using simulated annealing algorithm hybridized local search and multi-start algorithms", *Applied Soft Computing Journal*, 165, 112109, doi:https://doi.org/10.1016/j.asoc.2024.112109.

[2] Bäck, T. (1996). "Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms", *New York, 1996; Online Edn, Oxford Academic*, doi:https://doi.org/10.1093/oso/9780195099713.001.0001.

[3] Baeck, T., Fogel, D.B, Michalewicz, Z. (1997). "Handbook of evolutionary computation", *(1st ed.). CRC Press.* doi:https://doi.org/10.1201/9780367802486.

[4] Benlic, U., Hao, J.K. (2011). "An effective multilevel tabu search approach for balanced graph partitioning", *Computers & Operations Research*, 38(7), 1066-1075, doi:https://doi.org/10.1016/j.cor.2010.10.007.

[5] Boulif, M. (2010). "Genetic algorithm encoding representations for graph partitioning problems", *International Conference on Machine and Web Intelligence, Algiers, Algeria*, 2010, 288-291, doi:https://doi.org/10.1109/ICMWI.2010.5648133.

[6] Bruglieri, M., Cordone, R. (2021). "Metaheuristics for the minimum gap graph partitioning problem", *Computers & Operations Research*, 132, 105301, doi:https://doi.org/10.1016/j.cor.2021.105301.

[7] Brunetta, L., Conforti, M., Rinaldi, G. (1997). "A branch-and-cut algorithm for the equicut problem", *Mathematical Programming*, 78, 243-263, doi:https://doi.org/10.1007/BF02614373.

[8] Bui, T.N., Moon, B.R. (1996). "Genetic algorithm and graph partitioning", *IEEE Transactions on Computers*, 45(7), 841-855, doi:https://doi.org/10.1109/12.508322.

[9] Chaouche, A., Boulif, M. (2019). "Solving the unsupervised graph partitioning problem with genetic algorithms: Classical and new encoding representations", *Computers & Industrial Engineering*, 137, 106025, doi:https://doi.org/10.1016/j.cie.2019.106025.

[10] Delgado, M., Verdegay, H.K., Vila, A.A. (1990). "On valuation and optimization problems in fuzzy graphs: A general approach and some particular cases", *ORSA Journal on Computing*, 2(1), 74-83, doi:https://doi.org/10.1287/ijoc.2.1.74.

[11] Dolan, E., Moré, J. (2002). "Benchmarking optimization software with performance profiles", *Mathematical Programming*, 91, 201-213, doi:https://doi.org/10.1007/s101070100263.

[12] Eiben, A.E., Smith, J.E. (2015). "Introduction to evolutionary computing", *Springer*, doi:https://doi.org/10.1007/978-3-662-44874-8.

[13] Farshbaf, M., Feizi-Derakhshi, M.R. (2009). "Multi-objective optimization of graph partitioning using genetic algorithms.", *Third International Conference on Advanced Engineering Computing and Applications in Sciences, Sliema, Malta,* 1-6, doi:https://doi.org/10.1109/ADVCOMP.2009.8.

[14] Firouzian, S., Adabitabar Firozja, M. (2016). "Fuzzy number-valued fuzzy graph", *Control and Optimization in Applied Mathematics*, 1(2), 77-86.

[15] Firouzian, S., Sedghi, S., Shobe, N. (2021). "On edge fuzzy line graphs and their fuzzy congraphs", *Control and Optimization in Applied Mathematics*, 6(1), 81-89, doi:https://doi.org/10.30473/coam.2021.44084.1102.

[16] Gill, P.E., Murray, W., Wright, M.H. (2021). "Numerical linear algebra and optimization", *Book Series Name: Classics in Applied Mathematics, SIAM*, doi:https://doi.org/10.1137/1.9781611976571.

[17] Glover, F. (1986). "Future paths for integer programming and links to artificial intelligence", *Computers & Operations Research*, 13(5), 533-549, doi:https://doi.org/10.1016/0305-0548(86)90048-1.

[18] Goldberg, D.E. (1989). "Genetic algorithms in search, optimization, and machine learning", *Addison-Wesley Longman Publishing Co., Inc.*, doi:https://doi.org/10.5555/534133.

[19] Hager, W.W., Krylyuk, Y. (1999). "Graph partitioning and continuous quadratic programming", *SIAM Journal on Discrete Mathematics*, 12(4), 500-523, doi:https://doi.org/10.1137/S0895480199335829.

[20] Hager, W.W., Phan, D.T., Zhang, H. (2013). "An exact algorithm for graph partitioning", *Mathematical Programming*, 137(1), 531-556, doi:https://doi.org/10.1007/s10107-011-0503-x.

[21] Holland, J.H. (1975). "Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence", *Complex Adaptive Systems, University of Michigan Press*, doi:https://doi.org/10.7551/mitpress/1090.001.0001.

[22] Johnson, E.L., Mehrotra, A., Nemhauser, G.L. (1993). "Min-cut clustering", *Mathematical Programming*, 62, 133-151, doi:https://doi.org/10.1007/BF01585164.

[23] Kadluczka, P., Wala, K. (1995). "Tabu search and genetic algorithms for the generalized graph partitioning problem", *Control and Cybernetics*, 24, 459-476.

[24] Kaufman, W.E., Krambeck, F.J., Prater, C.D., Weekman, V.W. (1977). "Automation and databases in an industrial laboratory", *IEEE Conference on Decision and Control including the 16th Symposium on Adaptive Processes and A Special Symposium on Fuzzy Set Theory and Applications, New Orleans, LA*, 494-497, doi:https://doi.org/10.1109/CDC.1977.271623.

[25] Kernighan, B.W., Lin, S. (1970). "An efficient heuristic procedure for partitioning graphs", *The Bell System Technical Journal*, 49(2), 291-307, doi:https://doi.org/10.1002/j.1538-7305.1970.tb01770.x.

[26] Kóczy, László T. (1992). "Fuzzy graphs in the evaluation and optimization of networks", *Fuzzy Sets and Systems*, 46(3), 307-319, doi:https://doi.org/10.1016/0165-0114(92)90369-F.

[27] Kohmoto, K., Katayama, K., Narihisa, H. (2003). "Performance of a genetic algorithm for the graph partitioning problem", *Mathematical and Computer Modelling*, 38(11-13), 1325-1332, doi:http://dx.doi.org/10.1016/S0895-7177(03)90134-8.

[28] Lawson, C.L., Hanson. R.J. (1995). "Solving least squares problems", *Society for Industrial and Applied Mathematics, SIAM*, doi:https://doi.org/10.1137/1.9781611971217.

[29] Li, M., Chi, H., Zhou, C., Xu, S. (2020). "GAP: Genetic algorithm based large-scale graph partition in heterogeneous cluster", *IEEE Access*, 8, 144197-144204, doi:https://doi.org/10.1109/ACCESS.2020.3014351.

[30] Lim, A., Chee, Y-M. (1991). "Graph partitioning using tabu search", *1991 IEEE International Symposium on Circuits and Systems (ISCAS), Singapore*, 2, 1164-1167, doi:https://doi.org/10.1109/ISCAS.1991.176574.

[31] Mahdavi-Amiri, N., Nasseri, S.H. (2007). "Duality results and a dual simplex method for linear programming problems with trapezoidal fuzzy variables", *Fuzzy Sets and Systems*, 158(17), 1961-1978, doi:https://doi.org/10.1016/j.fss.2007.05.005.

[32] Pieter, M., Mouton, S. (2012). "Francis Guthrie: A colourful life", *The Mathematical Intelligencer*, 34, 67-75, doi:https://doi.org/10.1007/s00283-012-9307-y.

[33] Michalewicz, Z. (2013). "Genetic algorithms + Data structures = Evolution programs", *Springer-Verlag Berlin Heidelberg*, doi:https://doi.org/10.1007/978-3-662-03315-9.

[34] Panos, P., Birce B.B., Feyza, G., Ozgur, D., Birce, B. (2013). "Fuzzy combinatorial optimization problems", *Handbook of Combinatorial Optimization*, 1357-1413, doi:http://dx.doi.org/10.1007/978-1-4419-7997-1_68.

[35] Pillai, S.U. Suel, T., Cha, S. (2005). "The Perron-Frobenius theorem: Some of its applications", *IEEE Signal Processing Magazine*, 22(2), 62-75, doi:https://doi.org/10.1109/MSP.2005.1406483.

[36] Rosenfeld, A. (1975). "Fuzzy graphs", *Fuzzy Sets and their Applications to Cognitive and Decision Processes, Academic Press*, 77-95, doi:https://doi.org/10.1016/B978-0-12-775260-0.50008-6.

[37] Sharma, P.D., Rallapalli, S., Lakkaniga, N.R. (2023). "An innovative approach for predicting pandemic hotspots in complex wastewater networks using graph theory coupled with fuzzy logic", *Stochastic Environmental Research and Risk Assessment*, 37, 3639–3656, doi:https://doi.org/10.1007/s00477-023-02468-3.

[38] Shazely, S., Baraka, H., Abdel-Wahab, A. (1998). "Solving graph partitioning problem using genetic algorithms", *1998 Midwest Symposium on Circuits and Systems (Cat. No. 98CB36268)*, Notre Dame, IN, USA, 302-305, doi:https://doi.org/10.1109/MWSCAS.1998.759492.

[39] Yager, R.R. (1981). "A procedure for ordering fuzzy subsets of the unit interval", *Information Sciences*, 24(2), 143-161, doi:https://doi.org/10.1016/0020-0255(81)90017-7.

[40] Zadeh, L.A. (1976). "A fuzzy-algorithmic approach to the definition of complex or imprecise concepts", *International Journal of Man-Machine Studies*, 8(3), 249-291, doi:https://doi.org/10.1016/S0020-7373(76)80001-6.