Metaheuristic Approach to Spectral Reconstruction of Graphs

Petar Ćirković[1], Predrag Đorđević[1], Miloš Milićević[2], Tatjana Davidović[3]

[1] Faculty of Science and Mathematics, University of Niš
{petar.cirkovic,predrag.djordjevic}@pmf.edu.rs
[2] Faculty of Mathematics, University of Belgrade
mi19019@alas.matf.bg.ac.rs
[3] Mathematical Institute of the Serbian Academy of Sciences and Arts
tanjad@mi.sanu.ac.rs

Abstract. Characterization of a graph by its spectrum is a very attractive research problem that has numerous applications. It is shown that the graph is not necessarily uniquely determined by its spectrum in the most general case, i.e., there could be several non-isomorphic graphs corresponding to the same spectrum. All such graphs are called cospectral. However, in most of the cases, it is important to find at least one graph whose spectrum is equal to a given constant vector. This process is called Spectral Reconstruction of Graph (SRG) and it is known as one of the most difficult optimization problems. We address the SRG problem by the metaheuristic methods, more precisely, by Basic Variable Neighborhood Search (BVNS) and improvement-based Bee Colony Optimization (BCOi) methods. The resulting heuristics are called SRG-BVNS and SRG-BCOi, respectively. Both methods are implemented in such a way to take into account the graph properties defined by its spectrum. We compare the performance of the proposed methods with each other and with the results obtained by other approaches from the relevant literature on the reconstruction of some well-known graphs.

Keywords: Spectral graph theory; spectral distance; cospectral graphs; metaheuristics.

1 Introduction

Graphs are mathematical objects defined as 2-tuples $G = (V, E)$ [8], where $V = \{v_1, v_2, \ldots, v_n\}$, represents the set of *vertices* v_i, while $E \subseteq V \times V$ denotes the connections (relations) between the pairs of vertices and is called the set of *edges*. If there is a connection (edge) between vertices v_i and v_j, we say that $\{v_i, v_j\} \in E$ and that vertices v_i and v_j are *adjacent*. Graphs are used to model numerous problems in science, engineering, industry, etc. Usually, V is finite set, however, the infinite cases are also studied in the literature starting with [22]. In this paper, we consider only finite and undirect graphs.

The simplest graph representation is by the *Adjacency matrix* A with elements 0 or 1 defined as follows:

$$a_{ij} = \begin{cases} 1, \text{ if } \{v_i, v_j\} \in E; \\ 0, \text{ otherwise.} \end{cases}$$

If graph is undirected, A is symmetric, i.e., $a_{ij} = a_{ji}$.The *degree* of vertex v_i (denoted by d_i) in graph G represents the number of vertices adjacent to v_i, i.e., the number of edges having v_i as an end-vertex and it is calculated as $d_i = \sum_{j=1}^{n} a_{ij}$. *Eigenvalues* λ_i, $i = 1, 2, \ldots, n$ for the graph G are actually the eigenvalues of matrix A, i.e., the roots of its characteristic polynomial $P_G(x) = det(xI - A)$. As the adjacency matrix A is symmetric its eigenvalues are real numbers. The set of all eigenvalues of graph G is called *spectrum*. It can contain negative, positive values and zeros, with some repeated values. It is usual to represent the spectrum as a non-increasing array of values $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$. Then, the largest eigenvalue λ_1 of graph G is called *index*. An array x such that $Ax = \lambda x$ is known as *eigenvector* (corresponding to the eigenvalue λ) of graph G, and it actually represents the eigenvector of matrix A.

Spectral graph theory (SGT) [11, 23] studies graphs based on their adjacency matrix, more precisely, based on the *eigenvalues* and *eigenvectors* of this matrix. In recent literature, some other matrices associated with graphs are defined and analyzed, such as Laplacian matrix and signless Laplacian matrix ([11], section 1.3). However, they will not be considered in this paper. SGT has important applications in various fields of computer science [13], some of them including graph recognition problems [5, 7, 13, 19] as graphs represent natural models for various types of objects. It has been shown in the literature that some special classes of graphs, e.g., complete graphs, paths, cycles, are determined (to the isomorphism) by the spectrum with respect to the adjacency matrix A. However, it has been proved that it does not hold in the general case, i.e., an arbitrary graph cannot be fully characterized by its spectrum, and there may exist non-isomorphic graphs having the same spectrum. In particular, it has been shown that trees cannot be characterized by a spectrum, nor can molecules in chemistry. Non-isomorphic graphs that have identical spectra are called *cospectral*. In [17, 24], the number of non-isomorphic cospectral graphs is analyzed in relation to the three mentioned matrices for all graphs with $n \leq 11$ vertices. Graphs with $n = 12$ vertices are considered in [3]. In these papers, it was noticed that the number of graphs with non-isomorphic co-spectral mates decreases from $n = 10$ with respect to the total number of graphs with the same number of vertices. Based on that observation, a hypothesis has been introduced stating that graphs with a large number of vertices (for $n \to \infty$) may be determined by their spectrum. The hypothesis is still an open problem in SGT.

Our work is inspired by the results published in [5] and presents their generalization and expansion. In the first part of [5], the authors discussed the problem of Spectral Reconstruction of Graphs (SRG) [9] with the help of AutoGraphiX (AGX) software package developed at GERAD Institute in Montreal [1, 6]. AGX uses the Variable Neighborhood Search (VNS) metaheuristic method [18, 21] to find graphs that have extreme values of selected invariants or their combinations.

We have also applied AGX, optimized its execution by adding new constraints that enable to reduce the search space, and consequently, to decrease the time required to obtain the results.

As AGX is a general purpose software, it obviously contains many auxiliary functions that are not necessary for the considered problem. Therefore, we do not expect its good performance and we propose the application of metaheuristic methods to efficiently find a graph with the given spectrum. We have implemented a basic version of VNS (BVNS) and an improvement-based Bee Colony Optimization (BCOi) [14, 15] to tackle the SRG problem. The methods are called SRG-BVNS and SRG-BCOi, respectively. The stochastic nature of metaheuristic methods allows us to perform restarts from different random initial graphs, and to generate mutually non-isomorphic cospectral graphs (if any). However, finding all cospectral graphs still remains a challenging task because it is actually a NP-hard optimization problem: it is necessary to examine all graphs with n vertices and m edges and the number of such graphs is $\binom{n(n-1)/2}{m}$, i.e., the number of ways m edges can be distributed in $n(n-1)/2$ places.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of the relevant literature. The SRG problem is described in detail in Section 3, specifying its complexity and some special cases in which there are efficient algorithms for finding all non-isomorphic cospectral graphs with a given spectrum. In Sections 4 and 5, the implementations of SRG-BVNS and SRG-BCOi are described. The results obtained by applying the implemented methods to some known graphs from the literature, are presented in Section 6. Concluding remarks and guidelines for future work are given in Section 7.

2 Literature review

The study of graphs based on their spectra has become very popular in the past two decades because the spectrum can be determined relatively quickly (the computational complexity is, in the general case, $O(n^3)$, and for special classes of graphs this complexity may be significantly reduced). Based on the spectrum, various information can be determined on the structure of the corresponding graph [9, 11, 19], especially on some parameters of the graph that require exponential time for calculation. As already mentioned in the introduction, graphs are not uniquely determined by the spectrum, i.e., for some graphs there exist non-isomorphic cospectral graphs. However, due to its great importance, spectral recognition of graphs is intensively studied in the literature [9, 17, 24].

The review paper [9] defines 4 basic problems that are considered in connection with spectral recognition of graphs: characterization of graphs with a given spectrum; construction (exact or approximate) of a graph with a given spectrum; spectral similarity of graphs; and spectral perturbations of graphs. Let us note once again that all these problems are related to the spectrum of the graph and use *spectral distance*. This distance is defined for graphs with the same number of vertices as the distance between their spectra. Various types of distances may

be used, such as Euclidean, Manhattan or some other distance between ordered sequences of eigenvalues, i.e., vectors in the n-dimensional space.

The first problem (characterization of a graph with the given spectrum) [9] involves describing as many of its properties as possible based on the spectrum. For the second problem, it is necessary to find a graph whose spectrum is represented by a given vector (which actually represents the SRG problem that is considered in this paper). The characterization of a graph by the spectrum is achieved by solving the optimization problem representing minimization of the spectral distance between the given vector and the spectrum of the constructed graph. The solution to this problem does not have to be unique, several mutually non-isomorphic cospectral graphs can be obtained. It is obvious that the cospectral graphs are at a distance equal to zero, which is the minimal value of any spectral distance. The spectral distance can be considered as a measure of graphs' similarity, i.e., we say that the two graphs are similar if their spectral distance is small. Similar graphs are obtained from each other by small perturbations (changes in the structure or spectrum of graphs). Examples of perturbations are removing or adding edges, moving edges from one position to some other, and so on.

One of the first algorithms for spectral reconstruction of graphs based on the Laplacian matrix was developed in [7]. It is based on the Tabu Search (TS) metaheuristic method, starts from a random graph with n vertices and tries to minimize the spectral distance. The algorithm was tested on several classes of networks (random, regular, cluster graphs, etc.).

The VNS method is used in [5] indirectly, through the AGX program package. The authors have performed the reconstruction of some classes of graphs based on Euclidean and Manhattan spectral distances defined with respect to the various matrices associated with the graph. Graphs of up to 20 vertices have been analyzed and the number of successful reconstructions in 100 restarts was reported. The stopping criterion in each execution was 100,000 evaluations of the objective function (i.e., calculations of the spectral distance), and the initial solution was always a randomly generated graph. The paper [5] served as the inspiration for our work. We aim to maximally exploit the information that can be obtained about the target graph from its spectrum and to develop efficient implementations of our methods and to generate the desired graph in the shortest possible time. By repeated restarts, it is possible to obtain several non-isomorphic cospectral graphs.

3 Finding graph with a given spectrum

As it is already mentioned, SRG implies finding (one or more) graphs whose spectrum is equal to a given vector. In this paper we use the Euclidean distance, that is (among others) used in [5] as well. Let $C = (c_1, c_2, \ldots, c_n)$ be a given vector, let $G = (V, E)$ be a graph having n vertices, and let $S = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ represent its spectrum. It is necessary to perform transformations of the graph

G with an aim to minimize (nullify) the spectral distance defined by Eq. (1).

$$d = \sqrt{(c_1 - \lambda_1)^2 + (c_2 - \lambda_2)^2 + \cdots + (c_n - \lambda_n)^2}. \quad (1)$$

First, we applied AGX software [1, 6]. It is an interactive software package designed to find extreme graphs, i.e., graphs that minimize or maximize certain graph invariant or a function of graph invariants. A graph invariant is a parameter of a graph that is independent of the vertex and edge labeling. Graph invariants are, for example, the index of a graph (i.e., the largest eigenvalue λ_1), minimum (δ) and maximum (Δ) vertex degree, etc. [8, 11]. Searching for extremal graphs, AGX uses an optimization module based on the VNS metaheuristics and generates the corresponding graph examples for some special cases (for some given values of n). The researchers use these experimentally obtained graphs to set hypotheses for the general case and then try to prove them theoretically, "by hand" or applying some automatic theorem prover [2, 5, 6]. The latest version of the AGX software package (AGX 3.3.9) as well as the accompanying documentation can be downloaded from the Internet address `https://www.gerad.ca/Gilles.Caporossi/agx/AGX/AutoGraphiX.html`.

To solve SRG problem, we need to ask AGX to minimize the Euclidean distance between the given constant vector C and the ordered vector of eigenvalues of the required graph. To make things easier for AGX, we exploit the fact ([11], p. 85) that the number of edges in a graph can be calculated by the following equation $m = \frac{1}{2}\sum_{i=1}^{n} \lambda_i^2$. As the input vector C is actually the spectrum of the desired graph, we can calculate the number of its edges by Eq. (2), i.e., using the scalar product of the vector C with itself.

$$m = \frac{1}{2}\sum_{i=1}^{n} c_i^2. \quad (2)$$

On the other hand, the number of edges in the graph equals one half the sum of all elements of the adjacency matrix A, and we can reduce the search space for AGX by equalizing this sum with the scalar product of the vector C with itself. The block-diagram of the corresponding optimization task performed by AGX softvare is presented in Fig. 1. Constant vector C is an input parameter, while the adjacency matrix A is provided by AGX in the process of graph generation/transformation. The initial graph is generated randomly with the number of edges depending on the input vector C according to Eq. (2), while all other graphs are obtained by performing transformations (that preserve the number of edges) of the currently best found graph. The goal is to minimize the spectral distance d (given by Eq. (1)) between input vector C and graph defined by the adjacency matrix A, and therefore, the loop is executed until d becomes zero. However, it may take to much time and it is necessary to define some stopping criterion that will interrupt the execution of AGX even if the solution is not found. This is required also for fair comparison of AGX with other approaches.

It is important to note that AGX is a stochastic search engine, and therefore, each of its executions can give a different result, with respect to either the

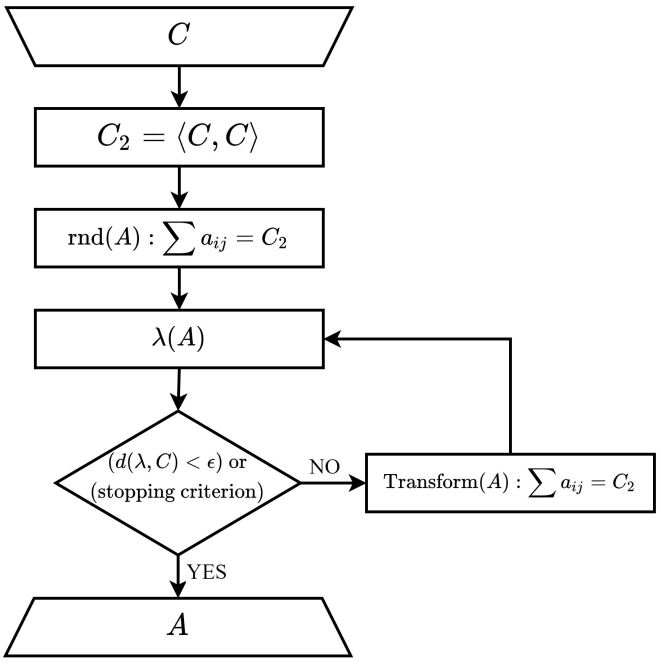

Fig. 1. Minimization of spectral distance according to AGX software

solution itself (when a non-isomorphic cospectral graph is obtained) or the time required to find the same graph. Consequently, for the analysis of AGX performance, it is necessary to repeat executions and determine some average (mean) results. Although AGX cannot guarantee a complete search of the solution space (graphs with a given number of vertices and edges), repeating the execution allows to find some non-isomorphic cospectral graphs (if any). Of course, the fact that AGX failed to generate a cospectral graph, i.e., it obtained the same solution in all executions, does not imply that there are no non-isomorphic cospectral graphs for the given graph. As it is already mentioned, in order to find all non-isomorphic cospectral graphs for a given graph, it is necessary to perform a complete search of graphs with the same number of vertices n and edges m. For example, for a graph with $n = 8$ vertices and $m = 9$ edges, it is necessary to examine 6,906,900 graphs. It is clear that (in the general case) as the number of vertices increases, the complexity of complete search is increasing nonlinearly. On the other hand, there are algorithms developed for some special classes of graphs that employ *a priori* knowledge about these graphs in order to reduce the number of analyzed graphs. An example of such an algorithm is described in [12]. The authors considered Smith graphs (whose spectrum is limited to the interval [-2,2]). They identified transformations that translate one Smith graph into an-

other, mutually non-isomorphic cospectral with the starting one, and developed an algorithm for generating all such graphs. Our goal is to develop algorithm that can be applied to any graph, and therefore, we cannot compare against the methodology proposed in [12]. We develop two metaheuristic methods (VNS and BCOi) that are compatible with the block-diagram from Fig. 1, however, transformations of solutions are performed in more systematic ways.

4 Variable neighborhood search for SRG

In this section we briefly recall some information about the VNS method and then describe its implementation for the considered SRG problem.

4.1 Variable neighborhood search

Variable Neighborhood Search (VNS) is a trajectory-based metaheuristic method proposed in [21]. It uses distances between solutions and employs one or more neighborhood structures to efficiently search the solution space of a considered optimization problem. VNS uses some problem-specific local search procedure(s) in the exploitation phase and changing distances between solutions to ensure the exploration of solution space. The role of exploration (diversification, perturbation) phase is to ensure escaping from local optima traps. VNS is widely used optimization tool with many variants and successful applications [18] and we used its basic variant (BVNS) for the SRG problem.

Algorithm 1 Pseudo-code for BVNS method

```
procedure BVNS(Problem input data, k_max, STOP)
    x_best ← InitSolution()
    repeat
        k ← 1
        repeat
            x' ← RandomSolution(x_best, N_k)                ▷ Shaking
            x'' ← LS(x')                                    ▷ Local Search
            if (f(x'') < f(x_best)) then                    ▷ Neighborhood Change
                x_best ← x''
                k ← 1
            else
                k ← k + 1
            end if
            Terminate ← StoppingCriterion(STOP)
        until (k > k_max ∨ Terminate)
    until (Terminate)
    return (x_best, f(x_best))
end procedure
```

BVNS employs a single type of neighborhood and consists of three main steps: Shaking, Local Search, and Neighborhood Change (see Alg. 1). The role of Shaking step is to ensure the diversification of the search. It performs a random perturbation of x_{best} in the given neighborhood and provides a starting solution x' to the next step. Local Search tries to improve x' by visiting its neighbors with respect to the selected neighborhood. After the Local Search, BVNS performs Neighborhood Change step in which it examines the quality of the obtained local optimum x''. If it is better than x_{best}, the search is concentrated around it (the global best solution x_{best} and the neighborhood index k are updated properly). Otherwise, only k is changed. The three main steps are repeated until a pre-specified stopping criterion is satisfied [18].

The main parameter of BVNS is k_{max}, the maximum number of neighborhoods for Shaking. Actually, the current value of k represents the distance between x_{best} and x' obtained within the Shaking phase. BVNS is known as the First Improvement (FI) search strategy because the search is always concentrated around x_{best}: as soon as this solution is improved, k is reset to 1.

4.2 Implementation details

Let us remind that the number of edges in the graph, which we want to generate based on the given spectrum C, is known, i.e., it can be calculated by Eq. (2). Therefore, we have implemented BVNS because it is enough to consider only one type of neighborhood: moving an edge from one place to another. This neighborhood preserves the number of edges in the graph. The resulting graphs do not have to be connected because this condition is not set for the starting graphs either (although all analyzed examples are connected graphs, they may have non-connected cospectral mates).

The solution of the considered problem is a graph denoted here by g, which should have a spectral distance (1) from the given constant vector C less than some predetermined constant ε. The initial solution is chosen randomly from all graphs with a given number of vertices n and edges m calculated by Eq. (2). Then, the transformations of the initial graph are performed following the steps of BVNS. The solutions in BVNS are represented by three data structures in order to reduce the computational complexity required to find neighbors of the considered graph in Local Search, as well as a random graph at a given distance (with respect to the number of transformations). Obviously, memory usage is sacrificed to increase the efficiency.

The first structure is the adjacency matrix $A = [a_{ij}]_{n \times n}$. It is needed for calculating the spectrum. The second structure contains the lists of adjacent $g[i].ls$ and non-adjacent $g[i].ln$ vertices for each vertex i in graph g. In addition, for each vertex i, it is necessary to always know the number of neighbors/non-neighbors, and this information is stored in the arrays $g[i].ns$ and $g[i].nn$. All these data structures allow to perform any transformation of the graph in a constant number steps $O(1)$. Each deleted vertex is replaced by the last one in the list and the corresponding number of elements is reduced by one ($--ns[i]$ and $--nn[i_1]$). A new vertex is always added to the end of the list, while

the number of list elements is increased by one ($nn[i]++$ and $ns[i_1]++$). Of course, it must be checked that some of the used lists are not empty. The mentioned operations are performed on randomly selected pairs of vertices (i, j) and (i_1, j_1) in the Shaking, while they are applied to all pairs of vertices from the neighborhood of the current solution in the Local Search. Of course, there are still some steps that cannot be performed in less than polynomial (or at least $\log n$) number of operations.

5 Bee Colony Optimization for SRG

This section contains the brief description of Bee Colony Optimization (BCO) metaheuristic, more precisely its improvement-based variant BCOi, as well as the implementation of BCOi for finding graphs with given spectrum.

5.1 Bee Colony Optimization

Bee Colony Optimization (BCO) is a population-based metaheuristic that mimics the foraging process of honeybees in nature [15]. The population consists of artificial bees, each responsible for one solution of the considered problem. During the execution of BCO, artificial bees build (in the constructive BCO variant, BCOc) or transform (in the improvement-based BCOi) their solutions in order to find the best possible with respect to the given objective. The BCO algorithm runs in iterations until a stopping condition is met and the best found solution (the so called global best) is reported as the final one.

Algorithm 2 Pseudo-code of the BCO algorithm

```
procedure BCO(Problem input data, B, NC, STOP)
    repeat                                              ▷ Main BCO loop
        for b ← 1, B do                                 ▷ Initializing population
            Sol(b) ← SelectSolution()
        end for
        for u ← 1, NC do
            for b ← 1, B do                             ▷ Forward pass
                EvaluateMove(Sol(b))
                SelectMove(Sol(b))
            end for
            EvaluateSolutions()                         ▷ Backward pass
            Loyalty()
            Recruitment()
        end for
        Update(x_best, f(x_best))
        Terminate ← StoppingCriterion(STOP)
    until (Terminate)
    return (x_best, f(x_best))
end procedure
```

Each BCO iteration contains several execution steps divided into two alternating phases: *forward pass* and *backward pass* (see Alg. 2). Within forward passes, all bees explore the search space by applying a predefined number of moves and obtain new population of solutions. Moves are related to building or transforming solutions, depending on the used BCO variant and they explore *a priori* knowledge about the considered problem. When a new population is obtained, the second phase (backward pass) is executed, where the information about the quality of solutions is exchanged between bees. The solution's quality is defined by the corresponding value of the objective function. The next step in backward pass is to select a subset of promising solutions to be further explored by applying *loyalty decision* and *recruitment* steps. Depending on the relative quality of its current solution with respect to the best solution in the current population, each bee decides with a certain probability should it stay *loyal* to that solution and become a *recruiter* that advertises its solution by simulating waggle dance of honeybees [15]. Obviously, bees with better solutions should have more chances to keep their solutions. A non-loyal bees are referred to as *uncommitted followers*, they abandon current solutions, and have to select one of the solutions held by recruiters. This selection is taken with a probability, such that better advertised solutions have greater opportunities to be chosen for further exploration. In the basic variant of BCO there are only two parameters:
• B – the number of bees involved in the search and
• NC – the number of forward/backward passes in a single BCO iteration.

5.2 Implementation of SRG-BCOi

As in BVNS, we used multiple data structures to represent solutions. The first is adjacency matrix, represented by 2-D arrays in the C(C++) programming language. For each bee b we introduced variable $A[b]$ as array of arrays containing $n * n$ elements. Therefore, our data structure A is actually an 3-D array. If in the solution handled by bee b, vertices i and j of the corresponding graph are connected, then $A[b][(i-1)*n+j-1] = A[b][(j-1)*n+i-1] = 1$ (as A is symmetric matrix), otherwise, the corresponding elements are equal to 0. We chose 1-D array for storing matrix because it is used in a version of Jacobi algorithm for calculating eigenvalues, which we found on the internet [4]. Our algorithm heavily relies on powerful data structures *vector* and *unordered_set* from C++. Unordered sets take (approximately) constant time to perform insert, delete and find operations, which is very important for obtaining efficient implementation of iterative algorithms. These data structures are used to model lists of adjacent and non-adjacent vertices for a given vertex v_i. It is important to note that here the dimension of used vectors and unordered sets increases by one, for counting bees in our population-based BCOi algorithm. To increase efficiency even more, we store in a separate vector (for each bee) non-isolated vertices, i.e., the ones that have at least one neighbour. We use this vector to select the first end-vertex of an edge to be removed. In addition, we list vertices that have less than $n-1$ adjacent vertices, to efficiently select the end-vertices of an edge to be added.

An initial population of each BCOi iteration is constructed randomly, by adding edges starting from an empty graph (containing only vertices). For each initial solution, we calculate spectrum by Jacobi algorithm and its spectral distance from the input vector C to evaluate the obtained solutions and to check if we already found the desired graph.

Forward pass involves the required transformations. Each transformation consists of moving a (randomly selected) number (o) of edges from one position to another one. As the first step, we need to select a random value for variable o from the interval $[1, 2 * m]$. The range for o is determined experimentally, having in mind that we should enable performing significant changes of the current solution. Although the total number of edges to be moved is only m, we allow o to take larger values, i.e., to move some edges more than once and, possibly, increase the diversity of the obtained solution. The value for o is determined for each bee separately, ensuring various treatment of the same solutions assigned to different bees (after recruitment). The second step in solution transformation assumes substituting o times an existing edge with an non-existing one. To determine the edge to be removed, we randomly select an element i from the set of non-isolated vertices (as the first end-vertex of the corresponding edge) and then pick randomly one of the vertices (j) adjacent to i from the corresponding unordered set. In a similar way, we select an edge (i_1, j_1) to be included in the transformed graph. Vertex i_1 is selected randomly from the set of vertices that have less than $n - 1$ adjacent vertices, while j_1 is determined as a random elements from the set of non-adjacent vertices of i_1. Random selection from a set usually takes linear time but we found smart trick to avoid it, on the internet [20]. When o transformations are completed, the spectrum of the resulting graph is calculated by Jacobi algorithm and used to determine spectral distance from the input vector C. Among all B solutions, the one with the smallest spectral distance is identified and used to check if we already found the desired graph or if the current best solution is improved.

The backward pass is performed in standard way described in [15]. The probability that bee is loyal to the current solution equals the normalized value of the corresponding objective function, while the recruitment is performed using the roulette wheel composed of solutions advertised by recruiters. Redundant solution representation helps to reduce the complexity of these steps also.

6 Experimental evaluation

Here we present the results of applying AGX software and the proposed SRG-BVNS and SRG-BCOi methods to the reconstruction of some graph examples with a small number of vertices.

6.1 Testing environment

SRG-BVNS method is implemented in the R language ([16]) within RStudio Ver. 2022.02.0 for Windows and executed on Intel Core i7-11800H 2.30GHz (24MB

Cache, up to 4.6GHz) 16GB DDR4, 512GB SSD, NVidia GeForce RTX 3050 Ti, GDDR6 4GB VRAM. AGX software is run on the same computer. SRG-BCOi is coded in C++ and executed on Intel(R) Core(TM) i3-7020U CPU 2.30GHz, 8GB DDR4, 512 GB NVMe SSD, Nvidia GeForce MX130 with 2GB VRAM. In order to be able to ensure fair comparison of the tested methods, we set the stopping criterion for all of them to be the maximum number of objective function evaluation. As in [5], this number is set to 100000.

SRG is specific optimization problem because we know the optimal value of the objective function (when the optimal solution found, the spectral distance between the corresponding graph and a given input vector equals zero). Therefore, we "just" need to find a graph with n vertices and m (calculated by Eg. (2)) edges satisfying the condition $d = 0$, where d is calculated by Eq. (1). This also means that we can stop the execution of the algorithms after the optimal solution is found. As we already noted, the solution space (depending on n and m) can be quite large, making our task very hard.

All of the compared methods are stochastic search algorithms, and therefore, we need to execute them repeatedly (for different values of random generator's seed) in order to evaluate their stability and performance. We set the number of repetitions to 100 as it ensures statistical significance of the obtained results. As the performance measure, we report the number of successful runs, i.e., the number of graph reconstructions in 100 repetitions, as well as the average number of required objective function evaluations. In the cases when solution was not found in each of 100 executions, we report the average value of the objective function. Regarding the parameters of the compared methods, we used default settings for AGX and performed some preliminary experiments to determine the values of SRG-BVNS and SRG-BCOi parameters. For SRG-BVNS the parameters are specified as follows: $k_{max} = m$ and we apply a FI strategy in LS in order to reduce the time spent in the intensification phase. Parameters of SRG-BCOi are set to the following values: $B = 6$ and $NC = 30$.

6.2 Results of spectral reconstruction of some graph examples

The graphs that we selected as the test examples for comparison are presented in Fig. 2 and Fig. 3. These are graphs with 8, 9, and 10 vertices that have been identified in [10] as suitable models for multiprocessor systems. To be able to control the experiment and to replicate the results, we used a fixed set of values for seed in SRG-BVNS, and SRG-BCOi. For the sake of simplicity, seed value in the i-th execution equals i. To the best of our knowledge, it is not possible to control seed value in AGX and its results may be slightly different in some new executions. We hope that 100 repetitions is enough to have a general judgement about AGX performance. The comparison results of these three algorithms are presented in Table 1.

Table 1 is organized as follows: the first column contains the name of the graph example used to define the input vector C; the remaining columns are grouped by three and they contain the results for each of the compared methods. The first group of three columns show the number of successful reconstructions,

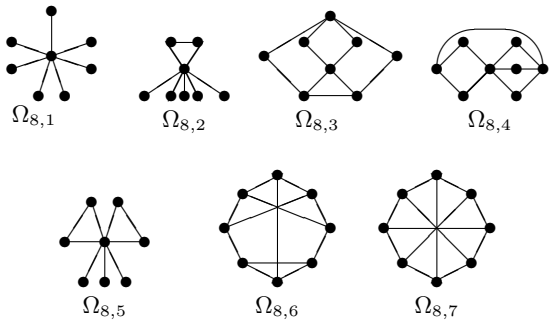

Fig. 2. Test examples with 8 vertices

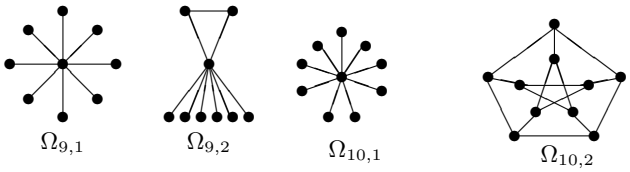

Fig. 3. Test examples with 9 and 10 vertices

Table 1. Comparison of AGX, SRG-BVNS, SRG-BCOi

Graph	AGX			SRG-BVNS			SRG-BCOi		
	#graphs	av. eval.	av. obj.	#graphs	av. eval.	av. obj.	#graphs	av. eval.	av. obj.
$\Omega_{8,1}$	42	65676.66	0.56	100	**445.76**	0.00	100	595.37	0.00
$\Omega_{8,2}$	16	88186.25	0.23	100	735.64	0.00	100	**339.73**	0.00
$\Omega_{8,3}$	100	1390.70	0.00	100	**324.88**	0.00	100	1017.03	0.00
$\Omega_{8,4}$	45	58939.98	0.20	100	**489.14**	0.00	100	1557.70	0.00
$\Omega_{8,5}$	100	1698.89	0.00	100	336.16	0.00	100	**123.73**	0.00
$\Omega_{8,6}$	100	2454.73	0.00	100	**409.97**	0.00	100	*10288.97	0.00
$\Omega_{8,7}$	100	4563.33	0.00	100	**415.99**	0.00	100	11818.75	0.00
$\Omega_{9,1}$	98	23256.09	0.05	100	**369.55**	0.00	100	446.10	0.00
$\Omega_{9,2}$	0	100000.00	0.27	100	**1182.50**	0.00	100	1611.08	0.00
$\Omega_{10,1}$	0	100000.00	0.97	100	**1143.97**	0.00	100	1866.42	0.00
$\Omega_{10,2}$	95	20094.89	0.06	100	**1513.41**	0.00	46	*75939.58	1.01

$*$ - the results are obtained when 2 (out of 6) initial solutions are set to the current best solution.

the average number of function evaluations, and the average value of the objective function for AGX, respectively. The corresponding results for SRG-BVNS and SRG-BCOi are presented in the columns 5-7 and 8-10.

Comparing the results from Table 1 we can conclude that both problem-oriented metaheuristic implementations outperformed AGX (except for one example where AGX performed better than SRG-BCOi). This result was expected having in mind that AGX is a general-purpose graph optimization software. SRG-BVNS was able to find a graph with given spectra in all executions, while SRG-BCOi had troubles with the last tested graph, the Petersen graph $\Omega_{10,2}$. With respect to the average number of objective function evaluations, the supe-

riority of SRG-BVNS is evident in all but two examples, where SRG-BCOi managed to reconstruct a graph faster. Our main conclusion is that single-solution metaheuristic performs better for these examples and we believe that it is a consequence of more systematic search with less randomness, that may lead to the situations where some solutions are visited more than once. We tried to resolve this problem by recording visited solutions in a hash table, however, it turned out that searching this table is also time consuming.

7 Conclusion

We considered the problem of Spectral Reconstruction of a Graph (SRG) and developed the Basic Variable Neighborhood Search (BVNS) and the improvement-based Bee Colony Optimization (BCOi). The SRG problem consists of finding at least one graph whose spectrum coincides with a given vector. The implemented metaheuristic methods take into account the well-known relationship between the number of edges in the graph and its spectrum. The results of applying SRG-BVNS and SRG-BCOi to the reconstruction of some known graphs are compared with each other and with the results obtained using the AutoGraphiX (AGX) package. They clearly show the superiority of the proposed SRG-BVNS implementation with respect to both solution quality and search speed measured by the number of objective function evaluations needed for reconstruction. Potential topics for future research include experiments with graphs of larger dimensions, comparison with similar methods from the literature, and generation of more (as much as possible) non-isomorphic cospectral graphs (if any). In addition, we plan to incorporate other known connections between the parameters of the graph and its spectrum in order to reduce the search space and speedup the execution of our SRG-BVNS and SRG-BCOi methods. Other matrices associated with graphs and other types of distances can be used as well.

Acknowledgements. This work has been supported by the Serbian Ministry of Education, Science and Technological Development, Agreement No. 451-03-9/2021-14/200029, by the Serbian Academy of Sciences and Arts under the project F-159, and by the Science Fund of Republic of Serbia, under the project AI4TrustBC. The authors are grateful to Academician Dragoš Cvetković for numerous suggestions and comments that contributed the quality of this paper.

References

1. Aouchiche, M., Bonnefoy, J.M., Fidahoussen, A., Caporossi, G., Hansen, P., Hiesse, L., Lacheré, J., Monhait, A.: Variable Neighborhood Search for Extremal Graphs 14: The SutoGraphiX 2 System. In: Global Optimization, pp. 281–310. Springer (2006)
2. Aouchiche, M., Hansen, P.: A survey of automated conjectures in spectral graph theory. Linear algebra and its applications **432**(9), 2293–2322 (2010)
3. Brouwer, A.E., Spence, E.: Cospectral Graphs on 12 Vertices. The Electronic Journal of Combinatorics **16**, N20:1–3 (2009)

4. Burkardt, J.: Eigenvalues and Eigenvectors of a Symmetric Matrix (Jacobi Algorithm) (2019), https://people.sc.fsu.edu/~jburkardt/c_src/jacobi_eigenvalue/jacobi_eigenvalue.html
5. Caporossi, G., Cvetković, D., Rowlinson, P.: Spectral Reconstruction and Isomorphism of Graphs Using Variable Neighbourhood Search. Bull. Acad. Serbe Sci. Arts, Cl. Sci. Math. Natur., Sci. Math. **146**(39), 23–38 (2014)
6. Caporossi, G., Hansen, P.: Variable Neighborhood Search for Extremal Graphs: 1 the AutoGraphiX System. Discrete Mathematics **212**(1-2), 29–44 (2000)
7. Comellas, F., Diaz-Lopez, J.: Spectral Reconstruction of Complex Networks. Physica A: Statistical Mechanics and its Applications **387**(25), 6436–6442 (2008)
8. Cvetković, D.: Graph Theory and Applications. Naučna knjiga, Beograd, 3rd ed. (1990)
9. Cvetković, D.: Spectral Recognition of Graphs. YUJOR **22**(2), 145–161 (2012)
10. Cvetković, D., Davidović, T.: Multiprocessor Interconnection Networks. In: Zbornik radova, special issue Selected Topics on Applications of Graph Spectra, pp. 35–62. 2nd ed. 14(22), Mathematical Institute SANU (2011)
11. Cvetković, D., Doob, M., Sachs, H.: Spectra of Graphs: Theory and Application. Johann Ambrosius Barth Verlag, Heidelberg–Leipzig, 3rd ed. (1995)
12. Cvetković, D., Jerotijević, M.: Compositions of Cospectrality Graphs of Smith Graphs. Kragujevac Journal of Mathematics, **47**(2), 271-–279 (2023)
13. Cvetković, D., Simić, S.: Graph Spectra in Computer Science. Linear Algebra and its Applications **434**(6), 1545–1562 (2011)
14. Davidović, T., Ramljak, D., Šelmić, M., Teodorović, D.: Bee Colony Optimization for the p-center Problem. Comput. Oper. Res. **38**(10), 1367–1376 (2011)
15. Davidović, T., Teodorović, D., Šelmić, M.: Bee Colony Optimization Part I: The Algorithm Overview. YUJOR **25**(1), 33–56 (2015)
16. Dessau, R.B., Pipper, C.B.: "R"-project for Statistical Computing. Ugeskrift for laeger **170**(5), 328–330 (2008)
17. Haemers, W.H., Spence, E.: Enumeration of Cospectral Graphs. European Journal of Combinatorics **25**(2), 199–211 (2004)
18. Hansen, P., Mladenović, N., Brimberg, J., Moreno Pérez, J.A.: Variable Neighborhood Search. In: Handbook of Metaheuristics, pp. 57–97. Updated edition of a trailblazing volume, Springer (2019)
19. Jovanović, I.M.: Spectral Recognition of Graphs and Networks. Ph.D. thesis (in Serbian), University of Belgrade, Faculty of Mathematics (2014)
20. Matovitch: Random Element from unordered_set in $O(1)$ (2015), https://stackoverflow.com/questions/12761315/random-element-from-unordered-set-in-o1/31522686\#31522686
21. Mladenović, N., Hansen, P.: Variable Neighborhood Search. Comput. Oper. Res. **24**(11), 1097–1100 (1997)
22. Nash-Williams, C.: Infinite Graphs—A Survey. Journal of Combinatorial Theory **3**(3), 286–301 (1967)
23. Spielman, D.: Spectral Graph Theory. In: Naumann, U., Schenk, O. (eds.) Combinatorial Scientific Computing, pp. 18:1–30. No. 18, CRC Press (2012)
24. Van Dam, E.R., Haemers, W.H.: Which Graphs are Determined by their Spectrum? Linear Algebra and its Applications **373**, 241–272 (2003)