SPECIAL ISSUE ARTICLE
Networks

A Genetic Approach for the 2-Edge-Connected Minimum Branch Vertices Problem

Francesco Carrabs[1] | **Raffaele Cerulli**[1] | **Federica Laureana**[1] | **Domenico Serra**[1] | **Carmine Sorgente**[1]

[1]Department of Mathematics, University of Salerno, Fisciano SA, Italy

Correspondence
Domenico Serra, Department of Mathematics, University of Salerno, Fisciano SA, Italy
Email: dserra@unisa.it

This paper addresses the 2-Edge-Connected Minimum Branch Vertices problem, a variant of the Minimum Branch Vertices problem in which the spanning subgraph is required to be 2-edge-connected for survivability reasons. The problem has been recently introduced and finds application in optical networks design scenarios, where branch vertices are associated to switch devices that allow to split the entering light signals and send them to several adjacent vertices. An exact approach to the problem has been proposed in the literature. In this paper, we formally prove its NP-completeness and propose a genetic algorithm, which exploits some literature-provided procedures for efficiently checking and restoring solutions feasibility, and makes use of novel ad-hoc designed operators aiming to improve their values, reducing the number of branch vertices. The computational tests show that, on the benchmark instances, the genetic algorithm very often finds the optimal solution. Moreover, in order to further investigate the effectiveness and the performance of our algorithm, we generated a new set of random instances where the optimal solution is known a priori.

KEYWORDS
Network Optimization, Network Design, Optical Networks, Branch Vertices, 2-edge-connectivity, Genetic Algorithms

1 | INTRODUCTION

In this paper, we study the *2-Edge-Connected Minimum Branch Vertices* (2ECMBV) problem introduced by Laureana [10]. The 2ECMBV is a variant of the *Minimum Branch Vertices* (MBV) problem which looks for a spanning 2-edge-connected subgraph having minimum number of vertices with degree strictly greater than two.

The MBV problem was introduced by Gargano et al. [7] and consists in finding a spanning tree of a given graph, containing the minimum number of branch vertices, namely vertices having a degree greater than two in the tree. Carrabs et al. [2] introduced four IP formulations for the MBV problem, Silva et al. [17] proposed an edge-swap heuristic algorithm, while Marín [12] experimented both exact and heuristic solutions. A further contribution on the MBV problem has been provided by Silvestri et al. [18], who introduced some valid inequalities and a Branch and Cut approach. Furthermore, Cerulli et al. [3] presented some heuristic approaches able to quickly find high-quality solutions.

Variants of the problem have been considered by several authors: Moreno et al. [13] proposed a Miller–Tucker-Zemlin based formulation with valid inequalities and a heuristic approach for the d-Minimum Branch Vertices problem. In this problem the aim is to minimize the number of vertices with degree strictly greater than d. Carrabs et al. [1] introduced the Generalized Minimum Branch Vertices Problem, which partitions the starting graph into clusters and identifies a spanning tree that covers exactly one vertex per cluster, minimizing the number of branch vertices.

The MBV problem arises in the field of optical networks design problems where it is required to connect the vertices in a way such that the number of connections of each vertex is limited. More in detail, in an optical network, the wave division multiplexing technology allows sending, on the same optical fiber, several light beams having a different fixed wavelength. Multicast technology on an optical network allows replicating the optical signal from one source to many destination vertices by means of a network device (*switch*) that permits to split an entering light signal and to send it to more adjacent vertices. A light-tree connects the vertices of the network allowing multicasting communications. The vertices of the tree having degree greater than two are named *branch* vertices and they require a switch to split and propagate the light signal to the adjacent vertices. Due to budget constraints, the number of switches should be limited, hence the aim of the problem consists of finding a spanning tree of the network with the minimum number of branch vertices. In the 2ECMBV problem, it is added a survivability constraint that requires to find a 2-edge-connected spanning subgraph of the network with the minimum number of branch vertices. This new constraint guarantees the restoring of the network services in case of edges failures.

In this paper, we prove that 2ECMBV is NP-complete by showing that it is in NP and by providing a polynomial reduction of the MBV problem to it. Moreover, we developed an algorithm, based on 2-edge-connectivity, branch vertices and bridge edges properties, that finds and removes redundant edges from a solution to decrease the number of its branch vertices. Finally, we propose a genetic algorithm based on ad-hoc operators designed to widely explore the solution space. The computational results carried out on benchmark and new instances, show that our algorithm is fast and effective because, in the worst case, it returns solutions having one branch vertex more than the optimal/best ones.

The remainder of the paper is organized as follows. Section 2 introduces the definitions and the notations that are used throughout the paper, while Section 3 provides a proof of NP-completeness of the 2ECMBV problem. In Section 4, existing procedures for checking and restoring the 2-edge-connectivity of a graph are detailed, while in Section 5 a procedure to decrease the number of branch vertices, based on some theoretical results, is presented. Section 6 describes a randomized algorithm designed to quickly find starting feasible solutions for the 2ECMBV problem. Our genetic algorithm is presented in Section 7. Finally, the computational results are reported in Section 8, followed by the conclusions in Section 9.

2 | NOTATIONS AND DEFINITIONS

Let $G = (V, E)$ be an undirected, unweighted, connected graph, where V is the set of vertices, E is the set of edges, $|V| = n$ and $|E| = m$. Given a vertex $v \in V$, we denote by $d_G(v)$ the degree of v in G and by $\delta_G(v)$ the set of edges incident to v in G. Furthermore, given a set of vertices $W \subset V$, we denote by $\delta_G(W)$ the set of edges of G that connect any vertex in W with a vertex in $V \setminus W$, and by $G[W]$ the subgraph of G induced by the vertices in W. A vertex $v \in V$ is a *branch vertex* if its degree is strictly greater than two. Moreover, G is *2-edge-connected* if and only if, by removing one edge, the resulting subgraph is connected. The edge connectivity version of Menger's theorem [14] leads to a further characterization of the 2-edge-connectivity property: a graph results to be 2-edge-connected if at least two edge-disjoint paths between each pair of vertices exist. In the following, we only consider simple paths, i.e., paths without repeated vertices. Lastly, a 2-edge-connected spanning subgraph of G is a graph $G' = (V', E')$ where $V' = V$, $E' \subseteq E$ and every pair of vertices is connected by at least two edge-disjoint paths in G'.

The *2-Edge-Connected Minimum Branch Vertices* (2ECMBV) problem consists in finding a 2-edge-connected spanning subgraph G' of G having the minimum number of branch vertices. It is easy to see that the optimal solution value of the 2ECMBV is always equal to zero for any Hamiltonian graph, namely a graph G that contains a Hamiltonian cycle. Indeed, a Hamiltonian cycle visits every vertex of G exactly once and constitutes a 2-edge-connected spanning subgraph of G with no branch vertices. On the contrary, if G is 2-edge-connected and non-Hamiltonian, the optimal solution value of the 2ECMBV is strictly greater than zero, as proved by Proposition 1

Proposition 1 *Let $G(V, E)$ be a 2-edge-connected and non-Hamiltonian graph. Then, the optimal solution value of the 2ECMBV problem on G is strictly greater than zero (i.e., the optimal solution contains at least one branch vertex).*

Proof By contradiction, let us suppose that there exists a spanning subgraph G' in G that is 2-edge-connected and without branch vertices. The 2-edge-connectivity and the absence of branch vertices imply that the degree of each node in G' is equal to 2. Moreover, since G' is spanning, it is a Hamiltonian cycle of G, contradicting the hypothesis that G is non-Hamiltonian. ∎

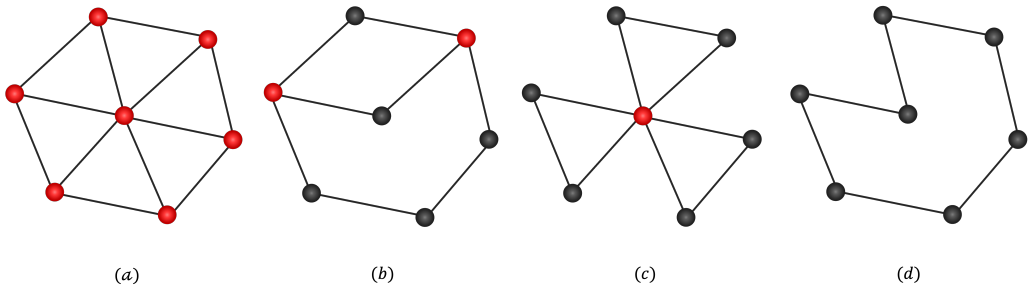

FIGURE 1 (a) A connected, undirected and unweighted graph G. The branch vertices are shown in red. (b) A 2-edge-connected spanning subgraph of G with two branch vertices. (c) A 2-edge-connected spanning subgraph of G with one branch vertex. (c) A 2-edge-connected spanning subgraph of G with no branch vertices.

For instance, given the graph G in Figure 1(a), three feasible solutions for the 2ECMBV are shown in Figures 1(b), 1(c) and 1(d). In particular, the subgraph in Figure 1(c) is better than the one in Figure 1(b) because it contains one branch vertex less, while Figure 1(d) shows an optimal solution for the 2ECMBV, that is a Hamiltonian cycle of G.

3 | COMPLEXITY RESULTS

In this section we prove that the 2ECMBV problem is NP-complete. To this end, we consider the corresponding decision version of the problem and, after showing that it is in NP, we provide a polynomial reduction from the MBV problem, which is known to be NP-complete [7].

The decision version of the 2ECMBV and MBV problems are the following:

$\mathbf{MBV_d}$: *Given a connected, undirected and unweighted graph G, is there a spanning tree T of G, having at most b branch vertices?*

$\mathbf{2ECMBV_d}$: *Given a connected, undirected and unweighted graph $G = (V, E)$, is there a 2-edge-connected spanning subgraph $G' = (V, E')$, with at most b branch vertices?*

The $2ECMBV_d$ is clearly in NP because verifying the feasibility of a given solution of the problem can be done in polynomial time. Indeed, to certify the feasibility of a solution we have to: *i)* check the subgraph 2-edge-connectivity; *ii)* check the subgraph spanning property; and *iii)* count the number of branch vertices in the subgraph. Step *i)* can be carried out in $O(m + n)$ using Tarjan algorithm [20] to find bridges in a graph, step *ii)* can be done with a simple breadth-first search of the subgraph, while step *iii)* is linear in the number of vertices.

To show the reduction, we transform in polynomial time an instance of MBV_d into an instance of $2ECMBV_d$. The answer of the $2ECMBV_d$ problem on this last instance is Yes if and only if the answer of the MBV_d problem is Yes on the original instance.

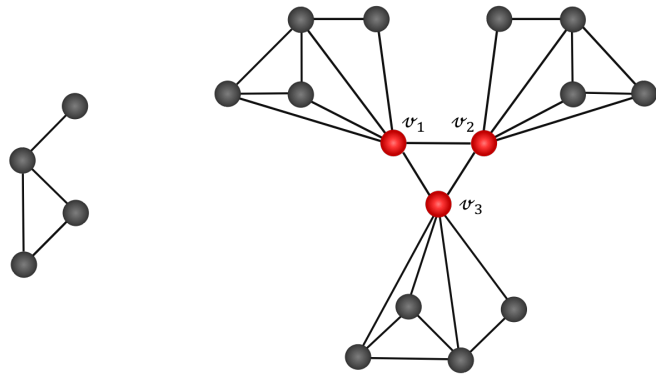

FIGURE 2 (a) Instance graph G of the MBV_d problem. (b) Instance graph G' of the $2ECMBV_d$ problem built from G, in the context of the $MBV_d \leq_p 2ECMBV_d$ reduction. The vertices v_1, v_2 and v_3 are used to connect three copies of G.

Proposition 2 *Given an instance (G, b) of MBV_d, such that G contains at least two vertices, $MBV_d \leq_p 2ECMBV_d$.*

Proof Let (G, b) be an instance of MBV_d. From this instance, we build an instance (G', b') of 2ECMBV_d as follows. G' contains three copies of G: $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G_3 = (V_3, E_3)$. Furthermore, three vertices, directly connected to each other, are added to G': one for each copy of G. Each vertex of G_i is connected, through an additional edge, to the corresponding vertex v_i. The value of b' is set to $3b+3$. An example of the described procedure is showed in Figure 2.

Let us suppose that T is a spanning tree of G having at most b branch vertices and let T_1, T_2 and T_3 be the three copies of T in G_1, G_2 and G_3, respectively. By construction, each T_i has at most b branch vertices. By connecting each leaf of T_i with the corresponding vertex v_i, the subgraph induced by $T_i \cup \{v_i\}$ is 2-edge-connected with at most $b+1$ branch vertices because the selected edges from v_i to the leaves do not transform these last vertices in branch vertices. It is easy to see that the subgraph induced by $T_1 \cup T_2 \cup T_3 \cup \{v_1, v_2, v_3\}$ is a 2-edge-connected subgraph of G' and has $3b+3$ branch vertices, at most.

On the other hand, let $T = (V_T, E_T)$ be a 2-edge-connected spanning subgraph of G' having at most b' branch vertices. It is easy to see that v_1, v_2 and v_3 are branch vertices in T. This means that the number of branch vertices in the subgraph of T induced by $\bigcup_{i=1}^{3} V_i$ is at most equal to $(b'-3)$. As a consequence, among the set of vertices V_1, V_2 and V_3, there must be at least one of them, w.l.o.g let us suppose V_1, such that the subgraph of T induced by V_1 contains at most $(b'-3)/3$ branch vertices. ∎

4 | 2-EDGE-CONNECTIVITY OPERATORS

In this section, we describe two operators used in our genetic algorithm to assure the feasibility of the chromosomes. The first operator verifies the 2-edge-connectivity of a subgraph, while the second one restores this property, when needed.

4.1 | 2-Edge-Connectivity Checker Operator

In the literature, there exist several algorithms to verify whether a graph is 2-edge-connected. Tarjan [19] proposed the biconnectivity algorithm to find biconnected components of a graph in $O(m+n)$ time. This algorithm can be used to determine if a graph is 2-vertex-connected. Galil et al. [6] polynomially reduced edge-connectivity to vertex-connectivity, thus every known algorithm for checking the 2-vertex-connectivity, including the biconnectivity algorithm, can be used to verify the 2-edge-connectivity. Tarjan [20] also provided a $O(m+n)$ algorithm to determine bridges in a graph. An edge $e \in E$ is a bridge in $G = (V, E)$ if the number of connected components in $(V, E \setminus \{e\})$ is strictly greater than the number of connected components in G. Since a connected graph with at least two vertices is 2-edge-connected if and only if it has no bridges, this algorithm can be used to check 2-edge-connectivity, too. Finally, 2-connectivity and 2-edge-connectivity can also be checked in $O(m+n)$ time using the Schmidt [16] algorithm, which relies on chain decomposition to identify bridges and cut vertices of a graph. More in detail, after conducting a depth-first search, the fundamental cycles in the resulting depth-first search tree are considered, and each of them is added to the chain decomposition if it does not overlap with any previously added cycle; otherwise, only the initial non-overlapping segment is added. An edge is a bridge if it is not contained in any chain of the decomposition; accordingly, a connected graph with at least two vertices is 2-edge-connected if its chain decomposition partitions its set of edges. In the implementation of our genetic algorithm, we check the 2-edge-connectivity of subgraphs by using the Schmidt algorithm.

4.2 | 2-Edge-Connectivity Restorer Operator

When the 2-edge-connectivity property does not hold for a subgraph $G' = (V, E')$ of $G = (V, E)$, we restore this property by adding to E' edges from $E \setminus E'$. Eswaran et al. [5] showed that the minimum number of edges to be added to E', in order to make 2-edge-connected the subgraph G', can be determined in $O(m+n)$ if every edge in the complement graph of G' is available. This last condition does not hold in our case and then, to prevent the selection of non-existing edges in the original graph G, we have to solve the edge-weighted version of this problem, which is NP-hard [5]. Moreover, since we also aim to minimize the number of branch vertices in the new subgraph, the weights' assignment is carried out according to this purpose.

More in detail, we assign a weight w_e to every edge $e \in E$ as follows: *(i)* $w_e = \infty$ if $e \notin E \setminus E'$; *(ii)* $w_e = 1+\beta$ otherwise, where $\beta \in \{0, 1, 2\}$ is the number of no branch vertices incident on e. By doing so, we encourage the selection of edges incident on already branch vertices, which do not increment the cost of a solution. In order to quickly select good quality edge augmentations for G', we adopt an implementation of the 2-approximation algorithm proposed by Khuller et al. [15], that runs in $O(m + n \log n)$ time. In the remaining of the paper, we refer to the invocation of such procedure as the restorer operator, and denote it by `Restore2EC`.

5 | REDUCING BRANCH VERTICES

In this section, we introduce a procedure, named `BranchReduction`, which aims to reduce the number of branch vertices in the subgraph. Given a 2-edge-connected graph $G = (V, E)$, an edge subset $\bar{E} \subset E$ is considered *redundant* if the graph $(V, E \setminus \bar{E})$ is 2-edge-connected. `BranchReduction` removes the redundant edges that are incident to the branch vertices of G. The procedure is based on some 2-edge-connected subgraph properties, introduced by Laureana [10], whose statements and proofs are reported below for ease of reading.

By removing a branch vertex v from G, we obtain a graph $G' = (V', E')$, where $V' = V \setminus \{v\}$ and $E' = E \setminus \{e : e \in \delta_G(v)\}$, for which one of the following three cases occurs:

- **(1)** G' is not connected;
- **(2)** G' is 2-edge-connected;
- **(3)** G' is connected but not 2-edge-connected.

Moreover, since in case (3) the set of bridges $B(G')$ in G' is not empty, we can distinguish two further cases by removing all the bridges from G' and investigating the connected components $C_1, \ldots, C_t$ left in the subgraph of G' obtained by removing the bridges $B(G')$ from G':

- **(3a)** $|\delta_G(C_i) \cap B(G'))| \leq 2$, for any $i \in \{1, \ldots, t\}$;
- **(3b)** there exists $i \in \{1, \ldots, t\}$, such that $|\delta_G(C_i) \cap B(G'))| \geq 3$.

In case (1), v is said to be a cut vertex in G and, by Lemma 3, it is necessarily branch in any feasible solution to the 2ECMBV problem. In case (2), (3a) and (3b), Lemma 4, Lemma 5 and Lemma 6 hold, respectively.

Lemma 3 *Given a 2-edge-connected graph $G = (V, E)$ and a vertex $v \in V$, if v is a cut vertex in G, then it is branch in any 2-edge-connected spanning subgraph of G.*

Proof By definition of cut vertex, $G[V \setminus \{v\}]$ consists of several connected components $C_1, \ldots, C_\ell$, $\ell \geq 2$. At least two

of the edges in $\delta_G(v)$ having one endpoint in C_i belong to any spanning 2-edge-connected subgraph $\bar{G}$ of G, otherwise the subset of vertices $C_i \cup \{v\}$ would not be 2-edge-connected in $\bar{G}$. Since this holds for every $i \in \{1, \ldots, \ell\}$, the number of edges in $\delta_{\bar{G}}(v)$ is at least equal to 2ℓ, which implies that v is a branch vertex in $\bar{G}$. ∎

Lemma 4 ([10]) *Given a 2-edge-connected graph $G = (V, E)$ and a branch vertex $v \in V$, if $G[V \setminus \{v\}]$ is 2-edge-connected, then there exists a 2-edge-connected spanning subgraph G_v of G, such that v is not branch in G_v.*

Proof Since $G[V \setminus \{v\}]$ is 2-edge-connected, the subgraph $G_v = (V, (E \setminus \delta_G(v)) \cup \{e, f\})$ is 2-edge-connected for any $e, f \in \delta_G(v)$. Moreover, v is not branch in G_v, as it has degree two in it. ∎

Let us denote by $CG(G')$ the component graph associated to G', defined as follows. The set of vertices of $CG(G')$ contains a vertex for each connected component C_i in the subgraph obtained by removing from G' the bridges $B(G')$. Furthermore, two vertices of $CG(G')$ are connected by an edge if and only if the associated components C_i and C_j are connected by a bridge in G'. By construction, the edges incident on C_i are the edges in $\delta_G(C_i) \cap B(G')$, and $CG(G')$ is a tree.

Lemma 5 ([10]) *If (3a) holds, then there exists a 2-edge-connected spanning subgraph G_v of G, such that v is not branch in G_v.*

Proof If (3a) holds, the graph $CG(G')$ is a path, then there are exactly two leaves in it, namely there are two connected components C_i and C_j with $i, j \in \{1, \ldots, t\}$, such that $|\delta_G(C_i) \cap B(G')| = |\delta_G(C_j) \cap B(G')| = 1$, while $|\delta_G(C_k) \cap B(G')| = 2$, for any $k \neq i, j$. Since G is 2-edge-connected, there exist $e \in \delta_G(v) \cap \delta_G(C_i)$ and $f \in \delta_G(v) \cap \delta_G(C_j)$. The subgraph $G_v = (V, E_v)$, where $E_v = E \setminus \delta_G(v) \cup \{e, f\}$, is 2-edge-connected and v is not branch in it. ∎

Lemma 6 ([10]) *If (3b) holds, vertex v is branch in any feasible solution to the 2ECMBV problem.*

Proof When (3b) holds, in the graph $CG(G')$ there exists at least a vertex with degree three then there are at least three leaves, namely there are three connected components C_i, C_j and C_k such that $|\delta_G(C_i) \cap B(G')| = |\delta_G(C_j) \cap B(G')| = |\delta_G(C_k) \cap B(G')| = 1$. Since G is 2-edge-connected, there exist $e \in \delta_G(v) \cap \delta_G(C_i)$, $f \in \delta_G(v) \cap \delta_G(C_j)$ and $g \in \delta_G(v) \cap \delta_G(C_k)$. To ensure 2-edge-connectivity e, f and g must be selected in any feasible solution, thus v is branch in any feasible 2ECMBV solution. ∎

The pseudocode of the `BranchReduction` procedure is given in Algorithm 1. The algorithm takes as input a 2-edge-connected subgraph G' of G from which redundant edges are removed, trying to reduce the number of branch vertices. Using a priority queue Q, the procedure processes the branch vertices in G' one by one, according to their degree, in ascending order (lines 1-4). Since, at each iteration, the algorithm updates G', for each vertex v extracted from the queue, it is necessary to check whether v is still a branch vertex (line 5). If this is the case, the number of connected components in the graph G'' obtained by removing v from G' is computed. If more than a single connected component exist, v is a cut vertex in G' and, by Lemma 3, it is not possible to obtain a 2-edge-connected subgraph where it is not branch; thus, no more action is performed in this iteration (lines 6-8). Otherwise, bridges B in G'' are computed (line 9). If no bridge exists, we can use Lemma 4 and remove from G' all the edges in $\delta_{G'}(v)$, except for the two edges whose other endpoints have highest degree (lines 10-11); in addition to making v not branch, this choice potentially reduces the number of branch vertices in the neighborhood of v. Otherwise, we remove existing bridges from G'' and recompute the connected components in G'': for each found component C_i such that a single bridge in B is incident on C_i, we remove all the edges in $\delta_{G'}(v)$ except the edge (v, z) which maximizes the degree of z

(lines 13-20). This means reducing the number of edges incident on v: when every found component has at most two incident bridges, Lemma 5 holds and only two edges in $\delta_{G'}(v)$ are selected; otherwise, we know from Lemma 6 that v is necessarily a branch vertex, thus at least three edges in $\delta_{G'}(v)$ are selected. The algorithm performs at most one iteration for each branch vertex in the input subgraph G'. In each iteration, computing the connected components of G'' is the most expensive operation and it is carried out at most two times. Connected components are identified through a breadth-first search of the subgraph, thus the time complexity of the algorithm in the worst case is $O(b(n+m))$, where b is the number of branch vertices in the subgraph.

Algorithm 1: BranchReduction

Input: A 2-edge-connected graph $G' = (V', E')$

1 $Q \leftarrow \{v : d_{G'}(v) > 2\}$
2 **while** $|Q| > 0$ **do**
3 $v \leftarrow v \in Q : d_{G'}(v) \leq d_{G'}(u), \forall u \in Q$
4 $Q \leftarrow Q \setminus \{v\}$
5 **if** $d_{G'}(v) > 2$ **then**
6 $G'' = (V'', E'') \leftarrow (V' \setminus \{v\}, E' \setminus \{e : e \in \delta_{G'}(v)\})$
7 $C \leftarrow$ connected components of G''
8 **if** $|C| \leq 1$ **then**
9 $B \leftarrow$ bridges in G''
10 **if** $|B| == 0$ **then**
11 $G' \leftarrow (V', E'' \cup \arg\max_{(v,x),(v,y) \in \delta_{G'}(v)} \{\min(d_{G''}(x), d_{G''}(y))\})$
12 **else**
13 $G'' \leftarrow (V'', E'' \setminus B)$
14 $C_1, C_2, \ldots, C_k \leftarrow$ connected components of G''
15 $F = \emptyset$
16 **for** $i = 1 \ldots k$ **do**
17 **if** $|\{(x, y) \in B : x \in C_i$ **xor** $y \in C_i\}| == 1$ **then**
18 $e = \arg\max_{e=(v,z) \in \delta_{G'}(v)} \{d_{G''}(z)\}$
19 $F \leftarrow F \cup \{e\}$
20 $G' \leftarrow (V', E'' \cup F)$
21 **else**
22 **break**
23 **return** G'

6 | FINDING FEASIBLE SOLUTIONS

In this section we present a randomized procedure, named `BuildSolution`, designed to quickly find feasible and highly diversified solutions for the 2ECMBV problem. A set of so generated solutions constitutes a heterogeneous starting population for our genetic algorithm. The procedure is based on the following 2-edge-connectivity property.

Proposition 7 *Given a graph $G = (V, E)$ and two distinct vertices $u, v \in V$, if u and v are connected by $k \geq 2$ edge-disjoint paths $P_1, P_2, \ldots, P_k$, any pair of vertices in $P_1, P_2, \ldots, P_k$ is connected by at least 2 edge-disjoint paths.*

Proof *Let $G = (V, E)$ be a graph, and let u and v be two vertices in V. Let $P = \{P_1, P_2, \ldots, P_k\}$, $k \geq 2$, be the k edge-disjoint paths between u and v. Let x be a generic vertex in P_i and let y be a generic vertex in P_j, with $P_i, P_j \in P$, then we have to prove that there exist at least 2 edge-disjoint paths connecting x and y. The following two cases may occur:*

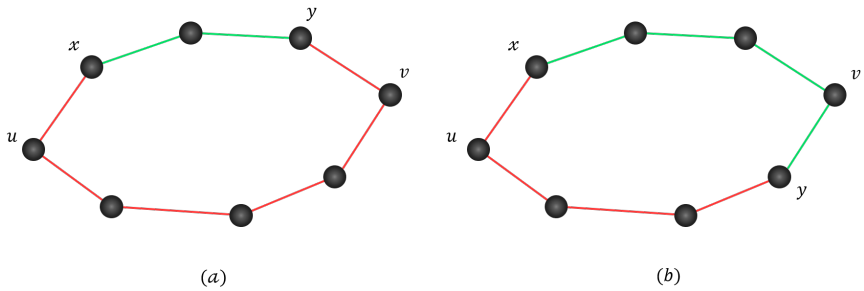

FIGURE 3 (a) Case where the vertices x and y are on the same path that connects u and v. (b) Case in which they are on different paths. In both cases there are 2 edge-disjoint paths between x and y, highlighted in the figure by different colors.

- $\mathbf{P_i = P_j}$*: In this case, the vertices x and y are on the same path and, without loss of generality, let us suppose that x precedes y, that is P_i has the following structure: $(u, \ldots, x, \ldots, y, \ldots, v)$ (Figure 3(a)). The vertices from x to y that appear along P_i represent a first path p' connecting the two vertices. On the other hand, a second path will have the form $p'' : (x, \ldots, u, \ldots, v, \ldots, y)$, where the edges from x to u and from v to y belong to P_i, while the edges from u to v belong to a path $P_t \in P$, with $P_t \neq P_i$, that exists because, by hypothesis, $k \geq 2$. The two paths p' and p'' do not share edges.*
- $\mathbf{P_i \neq P_j}$*: In this case, the x and y vertices are located on two separate paths, P_i and P_j. The first path between x and y has the form $p' : (x, \ldots, u, \ldots, y)$ and it includes the vertices from x to u along P_i and the vertices from u to y along P_j. The second path between x and y has the form $p'' : (x, \ldots, v, \ldots, y)$, which contains the vertices from x to v along P_i and the vertices from v to y along P_j. Since P_i and P_j, by hypothesis, are edge-disjoint paths, then p' and p'' are edge-disjoint paths, too.* ∎

By using Proposition 7, we designed the constructive procedure `BuildSolution` which incrementally builds a set S of vertices inducing a 2-edge-connected subgraph of G, by adding edge-disjoint paths between vertices in S and vertices in $V \setminus S$, until S contains all the vertices of G. The pseudocode of this procedure is given in Algorithm 2.

The algorithm takes as input the original 2-edge-connected graph G. After initializing the output graph G' with an empty set of edges (lines 1-2), a first vertex of G is randomly selected and added to S (line 3). By now, the algorithm adds a vertex $u \in V \setminus S$ to S only if u is 2-edge-connected with some vertex in S. To this end, at each iteration, the following operations are performed: *(i)* vertices x and y are randomly selected from S and $V \setminus S$, respectively (lines 4-5); *(ii)* a path P_1 in G between x and y is found and the edges in P_1 are temporarily removed from G (lines 7-8); *(iii)* a second x-y path P_2 in G is found (line 9) and, since the edges of P_1 have been removed from G, P_1 and P_2 are edge-disjoint paths connecting x and y; *(iv)* the edges in P_1 are re-added to G (line 10); *(v)* finally, all the edges in P_1 and P_2 are added to G' and, since we know from Proposition 7 that every pair of vertices in P_1 and P_2 is 2-edge-connected

Algorithm 2: BuildSolution

Input: The original 2-edge-connected graph $G = (V, E)$

1 $E' = \emptyset$
2 $G' \leftarrow (V, E')$
3 $S \leftarrow$ {a random vertex in V}
4 **while** $|V \setminus S| > 0$ **do**
5 $\quad x \leftarrow$ random vertex in S
6 $\quad y \leftarrow$ random vertex in $V \setminus S$
7 $\quad P_1 \leftarrow x$-y path in G
8 $\quad E \leftarrow E \setminus$ {edges in P_1}
9 $\quad P_2 \leftarrow x$-y path in G
10 $\quad E \leftarrow E \cup$ {edges in P_1}
11 $\quad E' \leftarrow E' \cup$ {edges in P_1} $\cup$ {edges in P_2}
12 $\quad S \leftarrow S \cup$ {vertices in P_1} $\cup$ {vertices in P_2}
13 **return** G'

in G', we add every vertices in P_1 and P_2 to G' (lines 11-12).

The algorithm ends when $V \setminus S$ is empty. This requires a variable number of iterations, depending on which vertices are randomly selected at each iteration.

In the best case, a single iteration is sufficient, while in the worst case, that occurs when only one vertex per iteration is added to S, the main loop requires $n - 1$ iterations, which is $O(n)$. The most expensive operation of each iteration is finding a path between two vertices in the graph and it is performed exactly two times per iteration. This can be achieved with several algorithms. We adopt a simple visit of the graph, which takes time $O(n + m)$. As a consequence, the complexity of the algorithm is $O(n(n + m))$ in the worst case. Since the input graph is 2-edge-connected, $m > n$ and the worst case time complexity is equivalent to $O(nm)$.

Generally, when studying the complexity of randomized algorithms, the expected running time is discussed depending on the random choices made by the algorithm [4]. Indeed, the worst case cannot be determined on the basis of the input, but it is only determined by the random choices made. We can express the time complexity of the algorithm through a recurrence relation. Denoting by $T(n)$ the number of iterations needed to build a solution for a graph with n vertices, we obtain the following recurrence relation:

$$T(|V \setminus S^{(i)}|) = 2 \cdot O(n + m) + T(|V \setminus S^{(i+1)}|),$$

where $S^{(i)}$ denotes set S at iteration i, with $|V \setminus S^{(0)}| = n$. Let us observe that the total number of operations carried out by the algorithm to build a solution for an instance of size n includes: *i)* $2 \cdot O(n + m)$ operations to find two edge-disjoint paths between a vertex in S and one in $V \setminus S^{(i)}$ in the original graph G; *ii)* the number of operations needed to perform the next iterations, that depends on the number of vertices still present in $V \setminus S^{(i+1)}$.

At each step, the set $V \setminus S^{(i)}$ is partitioned into $V \setminus S^{(i+1)}$ and $(V \setminus S^{(i)}) \setminus (V \setminus S^{(i+1)})$. The balance of such partitioning affects the recurrence describing the running time. If the best-case partitioning takes place, i.e., $|V \setminus S^{(1)}| = \emptyset$, a single iteration is sufficient and the recurrence has the solution $O(m+n)$. On the other hand, if the partitioning is unbalanced in every step, i.e., $|V \setminus S^{(i+1)}| = |V \setminus S^{(i)}| - 1 \ \forall i$, the recurrence has the solution $O(n(m+n)) = O(nm)$, which matches

the worst case. Finally, in the average case, on random 2-edge-connected graphs, the `BuildSolution` procedure has an expected running time of $O((n+m)\log_2 n)$.

For the sake of intuition, when after each step of the recurrence relation the number of vertices remaining in $V \setminus S^{(i+1)}$ is a fraction of n, a logarithmic number of steps is needed to solve the resulting recurrence: no matter how small this fraction is, the only required condition is that we do not remove from $V \setminus S^{(i)}$ a fixed number of vertices at each step i. In fact, let us say $\frac{n}{\alpha^i}$ is the number of vertices remaining in $V \setminus S^{(i+1)}$ after each step i, with $\alpha \in \mathbb{R}$ and $\alpha > 1$. We can rewrite the recurrence relation as follows:

$$T(|V \setminus S^{(0)}|) = 2c(n+m) + T\left(\frac{n}{\alpha}\right) \tag{1}$$

$$= 2c(n+m) + 2c\,(n+m) + T\left(\frac{n}{\alpha^2}\right) \tag{2}$$

$$= 2c(n+m) + 2c\,(n+m) + \cdots + 2c\,(n+m) + T\left(\frac{n}{\alpha^j}\right), \tag{3}$$

where $S^{(0)} = \varnothing$ and j is the number of iterations needed to solve the relation, that is $O(\log_2 n)$.

Clearly, the partitioning does not generally produce splits of constant proportionality at every step, but sufficiently well balanced and fairly unbalanced splits are expected to be randomly distributed in the recursion tree. It can be proved that, even in this case, the expected number of iterations until a best-case partitioning takes place is $O(\log_2 n)$ [4].

7 | GENETIC ALGORITHM

Genetic algorithms are powerful evolutionary algorithms, originally introduced by John Holland [9] and widely used to solve hard combinatorial optimization problems. Holland was inspired by Darwin's theory of evolution, based on the idea of natural selection. Genetic algorithms simulate the evolution of a population of individuals, each of which represents a feasible solution of the problem. Starting with an initial population, the algorithm performs a certain number of iterations and, at each iteration, a new population is generated on the basis of the previous one.

In this section, we describe our genetic algorithm (*GA*) for the 2ECMBV problem. In the algorithm, a solution of the problem is represented by a chromosome thus, as first step, we define how these chromosomes are encoded. Then, we define the fitness function, used to evaluate the quality of each chromosome, and the stopping criteria. Finally, we describe the selection, crossover and mutation operators.

7.1 | Chromosome representation and fitness function

In our algorithm, each chromosome C is associated to a feasible solution of the 2ECMBV problem on G. For this reason, C is a binary vector whose size is equal to the number of edges of G and the ith gene in it is equal to 1 if the edge e_i of G belongs to the solution and zero otherwise. We denote by $C[i]$ the value of ith gene of chromosome C.

An example of a feasible solution encoding is shown in Figure 4. Here $C_1[i] = 1$ for all $i = 1, \ldots, 9$ while $C_2[i] = 1$, for i={1,2,5,6,7,8} and $C_2[i] = 0$, for i={3,4,9}. From now on, we denote by $G(C)$ the subgraph of G induced by selected edges in C. According to this definition, $G(C_2)$ is the subgraph of G depicted in Figure 4(b).

The chromosomes of the population are ranked according to a *fitness function* $\mathcal{F}$. In our algorithm, the fitness function $\mathcal{F}(C)$ of a chromosome C is equal to the number of branch vertices in $G(C)$. The lower is the number of branch vertices the better is the fitness value. In Figure 4, we have that $\mathcal{F}(C_1) = 5$ while $\mathcal{F}(C_2) = 0$ and then $G(C_2)$

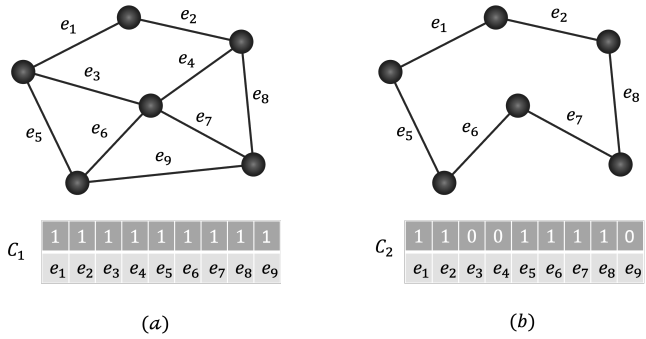

FIGURE 4 (a) The original graph G and (b) a 2-edge-connected subgraph of G with their corresponding encodings, C_1 and C_2, respectively.

is a better solution than $G(C_1)$.

7.2 | Initial population

The initial population is generated by using the `BuildSolution` procedure described in Section 6. Thanks to its randomized nature, the algorithm is able to generate, from the same input graph, different and heterogeneous feasible solutions. We repeatedly invoke this procedure until a fixed number of individuals, equal to the size $\mathcal{N}$ of the population, is obtained.

7.3 | Selection, Crossover and Mutation operators

The selection of two parents for the reproduction is carried out by using the *Tournament Selection* policy. This technique consists of randomly choosing t chromosomes from the current population and then the one with the best fitness function value is chosen as first parent. The process is iterated to select the second parent. In our implementation, t is equal to 3.

Two *crossover* operators, both taking as input two parent chromosomes and generating a new child chromosome, were designed: the former focuses on feasibility and naturally guarantees the 2-edge-connectivity and spanning properties, while the latter tries to reduce the number of branch vertices and exploits the restorer operator to assure that the child chromosome is a feasible solution. More in detail, the first crossover operator generates the child chromosome by copying one of two parents and then by adding to it some edges of the other parent. Let C_1 and C_2 be the two parent chromosomes and let $G(C_1) = (V_{C_1}, E_{C_1})$ and $G(C_2) = (V_{C_2}, E_{C_2})$ be the two induced subgraphs. To produce the child chromosome C_c, the crossover operator randomly select one of two parents, w.l.o.g. let us suppose C_1, and makes C_c a copy of this parent. Then, each edge in E_{C_2} is added to $E(C_c)$, i.e., the related gene is set to 1 in C_c, with probability 0.5. Figure 5 shows how this crossover works. As said, this first operator focuses on feasibility, indeed it always produces feasible solutions whose values are worse than or equal to those of the reference parents: the improvement of such values is left to the next operators. On the other hand, the second crossover operator generates the child chromosome C_C by performing the following three steps: *(i)* initially, all the edges in $E_{C_1} \cap E_{C_2}$ are added to E_{C_C}; then *(ii)* the edges incident on branch vertices in $G(C_C)$ are removed; and finally the restorer operator is invoked on $G(C_C)$. Figure 6 depicts an example of this crossover. Considering the edges from both the parents which do not

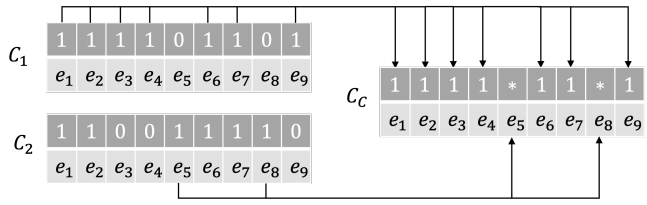

FIGURE 5 Illustration of the first crossover: the arrows represent the actual parent gene determining the value of the resulting child gene C_{C_i}. The special character $*$ means random choice between 0 and 1.

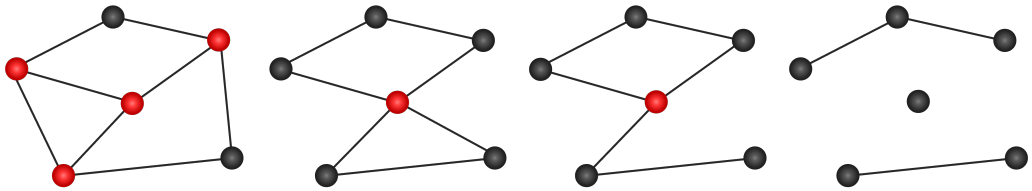

FIGURE 6 Illustration of the second crossover. (a) The first parent $G(C_1)$. (b) The second parent $G(C_2)$. (c) The subgraph induced by the set of common edges $E_{C_1} \cap E_{C_2}$. (d) The subgraph remaining after removing the edges incident on branch vertices. The child chromosome C_C is obtained by invoking the restorer operator on the last graph.

lead to branch vertices and then completing the solution is more expensive, but is more likely to produce promising child chromosomes. In our implementation, the two designed crossover operators are performed alternatively, with probabilities of 30% and 70%, respectively.

Generally, in genetic algorithms, after the generation of a new individual from two parents, a *mutation* operation is invoked with a certain probability. The motivation is to introduce new genetic material in the population, allowing to consider unexplored areas of the feasible region and thus reduce the probability to be trapped in local optima. In our algorithm, the mutation operation, performed with probability $\mathcal{P}_m$, randomly chooses $\lceil \mathcal{J}_m \cdot 1_C \rceil$ bits equal to one and $\lceil \mathcal{J}_m \cdot 0_C \rceil$ bits equal to zero and inverts them, where $\mathcal{J}_m \in [0, 1]$ is the mutation impact parameter, while 1_C and 0_C denote the number of genes equal to one and equal to zero in C, respectively.

Since the mutation operator and the local search procedure, described in the next section and invoked after the mutation, do not preserve the feasibility of the solution, `Restore2EC` operator is invoked at the end of the creation process of a new individual.

7.4 | Local Search

After the application of the crossover and mutation operators, the new individual can contain much more branch vertices than its parents. For this reason, we implemented a `LocalSearch` procedure which aims to reduce the number of these branch vertices by performing a sequence of edge replacements.

The idea behind the replacements of the edges is the following. Let $G(C)$ be the current solution and let u and v be two branch vertices in $G(C)$; `LocalSearch` removes an edge $(u, w) \in \delta_{G(C)}(u)$ and it inserts in $G(C)$ a new

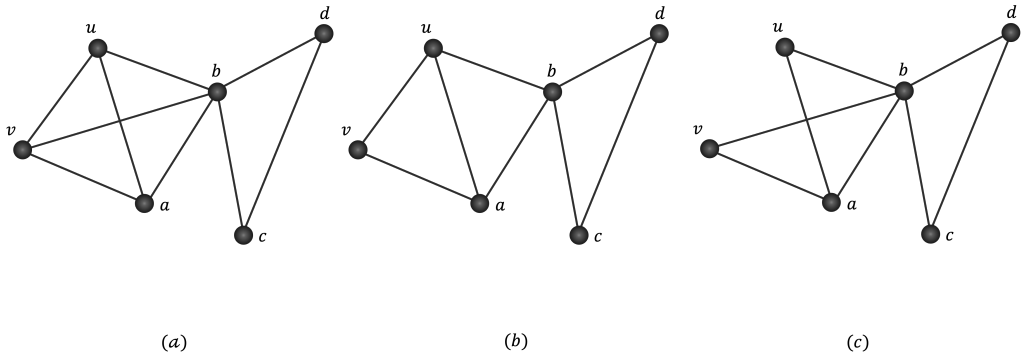

FIGURE 7 (a) A graph G with four branch vertices u, b, a and v; (b) a subgraph G' of G having three branch vertices: u, b and a; (c) a subgraph G'' of G with two branch vertices, a and b, obtained from G' by replacing the edge (v, u) with the edge (v, b).

edge $(w, v) \in \delta_G(w) \setminus \delta_{G(C)}(w)$, currently not in C, to try to preserve the 2-edge-connectivity. As a consequence of this replacement, the degree of u in $G(C)$ is decreased by one while the degree of v is increase by one. The aim of `LocalSearch` is to remove branch vertices from $G(C)$ by reducing their degree to 2 thanks to these replacements. Since such replacements cannot assure that the 2-edge-connectivity constraint is satisfied then, as last step, `LocalSearch` invokes the `Restore2EC` procedure on the new solution.

For instance, let us consider the graph $G = (V, E)$ shown in 7(a) and the 2-edge connected subgraph $G' = (V', E')$ in 7(b) with three branch vertices a, b and u. By invoking `LocalSearch` procedure on G', it removes from the branch vertex u the edge $(u, v) \in E'$ and adds to the branch vertex b the edge $(v, b) \in E \setminus E'$ (Figure 7(c)). In this way, b remains a branch vertex while u is no longer a branch vertex. It is easy to see that, according to this strategy, the `LocalSearch` procedure never introduces new branch vertices but it could reduce their number.

The pseudocode of the `LocalSearch` procedure is reported in Algorithm 3.

Given the graph G and the chromosome C, the procedure sets $G(C)$ as the subgraph of G associated to C and sets B to the empty set (line 1-2). All the branch vertices in $G(C)$ are then added to B (line 3-4). The while loop, in line 5, iterates until the set B gets empty. In this loop a branch vertex b of B is randomly selected (line 6) and for each edge (b, v) incident to b that is not in E_C, the procedure looks for another edge (v, u) in E_C having u as branch vertex (line 10). If this last edge is found, `LocalSearch` adds (b, v) to E_C, removes (v, u) from E_C and stops the for loop of line 9 (lines 11-12). In order to increase the possibility to preserve the 2-edge-connectivity while keeping the fitness value unchanged, in line 13 the procedure adds to E_C all the edges in $E \setminus E_C$ which endpoints are both branch vertices. Finally, `LocalSearch` invokes `Restore2EC` to assure that the solution returned satisfy the 2-edge-connectivity constraint.

7.5 | Shaking and Cleaning

A shaking operator has been implemented to escape from the local optimum and to encourage a wider exploration of the solution space. After $\mathcal{I}_{\mathsf{SH}}$ iterations of the genetic algorithm without improvements of the incumbent solution, $\lfloor \mathcal{N} \cdot \mathcal{J}_{\mathsf{SH}} \rfloor$ random individuals are replaced with new individuals generated from scratch. Here, $\mathcal{J}_{\mathsf{SH}} \in [0, 1]$ is the shaking impact parameter, which determines the number of individuals of the population to be replaced.

When the genetic algorithm ends its execution, a cleaning policy is applied by carrying out the `BranchReduction` procedure on all the individuals of the final population. As previously described, this allows to decrease the number

Algorithm 3: LocalSearch

Input: G and C

$G(C) = (V_C, E_C)$ //subgraph of G associated to C
$B \leftarrow \emptyset$
foreach *vertex* $u \in V_C$ **do**
  if $d_{G(C)}(u) > 2$ **then** $B \leftarrow B \cup \{u\}$
while *B is not empty* **do**
  $b \leftarrow$ random branch vertex in B
  for $(b, v) \in \delta_G(b)$ **do**
    if $(b, v) \notin E_C$ **then**
      for $(v, u) \in \delta_{G(C)}(v)$ **do**
        if $d_{G(C)}(u) > 2$ **then**
          $E_C \leftarrow E_C \setminus \{(v, u)\} \cup \{(b, v)\}$
          break
`add in` E_C `all` (u, v) `edges with` u `and` v `branch.`
return *Restore2EC*$(G, G(C))$

of branch vertices by removing a large number of redundant edges.

7.6 | Termination criteria

GA iterates until one of the two following stopping criteria is reached.

- The first criterion is based on $\mathcal{I}_{\mathsf{MAX}}$ parameter, representing the maximum number of iterations that the algorithm can carry out;
- The second criterion is based on the $\mathcal{I}_{\mathsf{SH}}$ parameter: if, after the application of the shaking operator, a new sequence of $\mathcal{I}_{\mathsf{SH}}$ iterations not improving the best chromosome fitness value occur, the algorithm stops.

8 | COMPUTATIONAL TESTS

In this section we present the computational results of the tests we made in order to evaluate the effectiveness and the performance of GA. The algorithm was coded in Python by using *NetworkX* [8] library on a Linux platform running on an Intel Xeon E5 2.3 GHz processor with 128 GB of RAM.

In the following two subsections, we describe how we generated the instances for the 2ECMBV and how the parameters of GA were chosen. The computational results are reported in subsection 8.3.

8.1 | Test instances

To the best of our knowledge, the only instances available in the literature for the 2ECMBV problem are the ones presented in [10]. However, since these instances were generated to test the B&C algorithm, presented in that work, they are too small to be used to test the effectiveness and performance of a metaheuristic. For this reason, we extended this dataset with new larger instances generated by using the same strategy proposed in [10]. The idea behind the instance generation procedure is to obtain a non-Hamiltonian and 3-connected graph, ensuring that there are not essential edges or vertices which must necessarily be branch.

More in detail, given a starting clique graph $G' = (V', E')$ with $|V'| = n' \geq 4$, let q be an integer such that $n' \geq 3q$. The procedure defines q disjoint subsets $W_1, \ldots, W_q$ of V' with $|W_i| = 3, i = 1 \ldots, q$. Then it generates q disjoint sets, $T_1, \ldots, T_q$, of new vertices, not in V', with $|T_i| \geq 3, i = 1, \ldots, q$. Finally, the procedure build the graph $G = (V, E)$ with $V = V' \cup \bigcup_{i=1}^{q} T_i$ and $E = E' \cup \bigcup_{i=1}^{q} \{(u, v) : u \in T_i, v \in W_i\}$. The graph $G = (V, E)$ just built is 3-connected and non-Hamiltonian (the proof is provided in [10]).

Note that $|V| = n' - \bar{n}$ where $\bar{n} = \sum_{i=1}^{q} |T_i|$ is the number of vertices that are added to the starting graph G'. In our computational tests, we grouped the instances in two sets: the *Small instances* where $n' \in \{20, 30, 40, 50\}$, which are the same used in [10] and the *Large instances* where $n' \in \{60, 70, 80, 90, 100\}$. For each combination of n', $\bar{n}$, and q, five different instances were generated that together represent a scenario. Thus, in total, this first set consists of 630 individual instances, grouped in 126 scenarios.

To further investigate the effectiveness of GA, we generated a set of random Hamiltonian instances. Since the optimal solution of 2ECMBV is always zero in these instances, then we can compare the solutions found by GA with the optimal ones. The generation of the instances is carried out as follows. Given n vertices and a probability d, we first create a Hamiltonian cycle randomly and then, for every couple of vertices i and j we introduce the edge (i, j) in the graph with probability d. For the generation of the instances we used the following values: $n \in \{100, 150, 200, 250, 300, 350, 400\}$ and $d \in \{0.3, 0.5, 0.7\}$. For each combination of these two parameters, we generated five different instances for a total of 105 Hamiltonian instances grouped in 21 scenarios.

8.2 | Parameters tuning

To find the best performing values for the parameters of the GA, we used the *IRACE* package [11], an automatic configuration tool for parameter setting. IRACE was executed on a subset of 147 instances selected according to all the possible combinations of the n and m parameters values. In particular, 126 instances were selected from the non-Hamiltonian and 3-connected set, while the remaining 21 instances were selected from the Hamiltonian set. Table 1 reports, for each parameter, the set of tested values (*Values*) and the corresponding target value (*Target*) in the best configuration found by IRACE.

8.3 | Computational results

In this section, we verify the effectiveness of GA algorithm by comparing it with the B&C proposed in [10]. The two algorithms are executed on the same machine and for the B&C we set a time limit equal to 2 hours.

Table 2 reports the results of GA and B&C on the Small instances. The first five columns show the characteristics of the instances: the number n' of vertices of the graph G', the cardinality $\bar{n}$ of $T_1 \cup \cdots \cup T_q$, the value q, the number n of vertices and the number m of edges of G.

The next four columns report the solution value (Obj) and the computational time ($Time$), in seconds, of B&C and

Parameter	Values	Target
$\mathcal{N}$	$\{30, 50, 70\}$	30
$\mathcal{I}_{\text{MAX}}$	$\{30, 50, 70\}$	50
$\mathcal{P}_m$	$\{0.05, 0.1, 0.15\}$	0.1
$\mathcal{J}_m$	$\{0.02, 0.04\}$	0.02
$\mathcal{I}_{\text{SH}}$	$\{3, 5, 7\}$	7
$\mathcal{J}_{\text{SH}}$	$\{1, 1/3, 1/4\}$	1/3

TABLE 1 Tested sets of values and *IRACE* choices for GA parameters.

GA, respectively. Whenever B&C reaches the time limit of two hours, the related solution value is marked with a "*" symbol to highlight that this value is an upper bound of the optimal solution value. Each row in the tables represents a scenario composed of five instances with the same characteristics but different seed (used to initialize the random number generator), and the results shown in each line are the average values of these five instances. The last column shows the gap (Gap) between the solution found by GA and the best/optimal solutions. This gap is computed by using the formula: $Gap = Obj(GA) - Obj(B\&C)$. Finally, at the bottom of the table, the Avg row reports the average values of Obj, $Time$ and Gap while the $\#Best$ row shows how many times GA finds the best/optimal solution.

The results of Table 2 show that B&C provides the optimal solution on 39 out of 56 scenarios with an average time equal to 1759 seconds. On 55 out of 56 scenarios, GA returns the same solutions of B&C and, in particular, it always finds the optimal solution in the scenarios where this solution is known. In the remaining scenario (50-50-16) the gap from the optimal/best solution is equal to 0.2. The results of the Avg row show that the difference between the average Obj values of the two algorithms is equal to 0.01. Moreover, the average computational time of GA in the Small instances is equal to 60.15 seconds, and it never exceeds the 3 minutes in this set of instances. However, it is worth noting that there are several scenarios where GA is slower than B&C. This occurs because, despite the stopping criteria used by GA, it has to always carry out a minimum number of iterations before stopping, even when it finds the best solution at the first iteration.

Table 3 shows the computational results of the two algorithms on the Large instances. Table headings have the same meaning that they have for Table 2. The results show that B&C provides the optimal solution only in 5 scenarios, while, on the remaining ones, it reaches the time limit. GA finds the same solutions of B&C on 65 out of 70 scenarios and, in the remaining five scenarios, its gap is always equal to 0.2. Moreover, the average gap is very low (0.014). These results highlight the effectiveness of GA which very often finds the best/optimal solution and when this does not occur, its gap from this solution is very low. Regarding the computational time, GA requires on average 319 seconds and never exceeds 1170 seconds. As expected, by increasing the size of the instances the computational time of the two algorithms is not comparable anymore.

Since in the scenarios where B&C reaches the time limit GA never finds a better solution, even in the largest instances, we suspect that the solutions provided by B&C could be the optimal ones or very close to the optimal ones but the algorithm fails to certify this optimality within the time limit.

As described in Section 8.1, we generated a new random set of instances to further investigate the effectiveness of our algorithm. To this end, we have chosen to generate Hamiltonian instances to know a priori the optimal solution value that is zero. Table 4 reports the results of GA on this new set of instances.

The first three columns show the characteristics of the instance: the number n of vertices of the graph, the

					B&C		GA		
n′	n̄	q	n	m	Obj	Time	Obj	Time	Gap
20	10	2	30	220	3.40	0.04	3.40	10.50	0.0
		3	30	220	3.00	0.06	3.00	9.56	0.0
	16	3	36	238	4.40	0.07	4.40	12.11	0.0
		5	36	238	5.00	2.60	5.00	9.68	0.0
	20	4	40	250	6.00	0.12	6.00	12.67	0.0
		6	40	250	7.00	12.55	7.00	10.19	0.0
	30	4	50	280	7.20	0.21	7.20	17.85	0.0
		6	50	280	8.60	2.33	8.60	14.26	0.0
	40	4	60	310	6.60	0.23	6.60	24.38	0.0
		6	60	310	8.40	1.47	8.40	19.99	0.0
	50	4	70	340	7.80	0.40	7.80	33.32	0.0
		6	70	340	10.40	5.92	10.40	26.91	0.0
	60	4	80	370	7.40	0.33	7.40	42.35	0.0
		6	80	370	10.40	2.61	10.40	33.41	0.0
30	15	3	45	480	4.40	0.17	4.40	22.35	0.0
		5	45	480	5.00	2.13	5.00	18.04	0.0
	24	4	54	507	6.60	0.34	6.60	24.94	0.0
		8	54	507	8.00	25.43	8.00	19.29	0.0
	30	6	60	525	8.40	7.15	8.40	24.06	0.0
		10	60	525	10.00	1030.81	10.00	19.30	0.0
	45	6	75	570	10.40	2.72	10.40	36.49	0.0
		10	75	570	11.00	605.38	11.00	30.54	0.0
	60	6	90	615	10.40	8.94	10.40	49.90	0.0
		10	90	615	15.40	248.00	15.40	37.15	0.0
	75	6	105	660	11.60	9.97	11.60	67.43	0.0
		10	105	660	15.40	646.42	15.40	50.79	0.0
	90	6	120	705	11.20	11.91	11.20	84.79	0.0
		10	120	705	18.20	617.87	18.20	65.67	0.0
40	20	4	60	840	7.00	1.73	7.00	37.00	0.0
		6	60	840	7.00	11.34	7.00	32.43	0.0
	32	6	72	876	7.60	10.75	7.60	39.98	0.0
		10	72	876	11.00*	7201.41	11.00	32.33	0.0
	40	8	80	900	11.40	509.73	11.40	43.49	0.0
		13	80	900	13.00*	7215.75	13.00	34.25	0.0
	60	8	100	960	13.60	111.74	13.60	58.73	0.0
		13	100	960	14.00*	7214.94	14.00	59.44	0.0
	80	8	120	1020	13.40	173.60	13.40	80.98	0.0
		13	120	1020	20.20*	4262.95	20.20	61.63	0.0
	100	8	140	1080	13.80	211.74	13.80	109.36	0.0
		13	140	1080	22.40*	6289.50	22.40	84.20	0.0
	120	8	160	1140	14.20	149.54	14.20	148.16	0.0
		13	160	1140	22.60*	3559.44	22.60	111.85	0.0
50	25	5	75	1300	6.00	2.84	6.00	58.27	0.0
		8	75	1300	8.00	364.32	8.00	52.12	0.0
	40	8	90	1345	9.00	713.76	9.00	61.05	0.0
		13	90	1345	13.00*	6810.18	13.00	57.51	0.0
	50	10	100	1375	11.00*	2492.98	11.00	72.03	0.0
		16	100	1375	17.00*	7218.56	17.20	58.70	0.2
	75	10	125	1450	11.00*	3271.07	11.00	101.91	0.0
		16	125	1450	17.00*	7218.50	17.00	78.62	0.0
	100	10	150	1525	11.00*	4580.52	11.00	132.02	0.0
		16	150	1525	17.00*	7218.60	17.00	116.93	0.0
	125	10	175	1600	11.00*	2496.49	11.00	191.37	0.0
		16	175	1600	17.00*	7218.80	17.00	156.33	0.0
	150	10	200	1675	11.00*	1557.97	11.00	258.74	0.0
		16	200	1675	17.00*	7218.64	17.00	211.11	0.0
Avg					10.87	1759.89	10.88	60.15	0.004
#Best									55

TABLE 2 Comparison between the solutions of B&C and GA on the Small instances.

					B&C		GA								B&C		GA		
n′	n̄	q	n	m	Obj	Time	Obj	Time	Gap	n′	n̄	q	n	m	Obj	Time	Obj	Time	Gap
60	30	6	90	1860	8.20	40.05	8.20	84.16	0.0	90	45	9	135	4140	10.00*	1595.33	10.00	199.60	0.0
		10	90	1860	10.00	1169.83	10.00	68.67	0.0			15	135	4140	15.00*	5689.53	15.00	171.51	0.0
	48	9	108	1914	13.60	727.07	13.60	87.42	0.0		72	14	162	4221	15.00*	6113.03	15.00	208.69	0.0
		16	108	1914	16.00*	7218.46	16.00	67.82	0.0			24	162	4221	24.00*	7219.43	24.00	163.24	0.0
	60	12	120	1950	16.60*	7217.62	16.60	101.83	0.0		90	18	180	4275	19.00*	7212.73	19.00	218.79	0.0
		20	120	1950	20.00*	7219.04	20.00	69.26	0.0			30	180	4275	30.00*	7212.53	30.00	184.76	0.0
	90	12	150	2040	19.40*	5893.34	19.40	122.03	0.0		135	18	225	4410	19.00*	7212.18	19.00	337.66	0.0
		20	150	2040	21.00*	7218.70	21.20	117.73	0.2			30	225	4410	31.00*	7212.78	31.00	297.96	0.0
	120	12	180	2130	21.20*	5057.12	21.20	177.71	0.0		180	18	270	4545	19.00*	7212.54	19.00	537.10	0.0
		20	180	2130	30.80*	7211.34	30.80	129.99	0.0			30	270	4545	31.00*	7212.27	31.00	385.45	0.0
	150	12	210	2220	21.20*	5643.38	21.20	236.21	0.0		225	18	315	4680	19.00*	7213.94	19.00	736.06	0.0
		20	210	2220	34.20*	7201.44	34.20	173.92	0.0			30	315	4680	31.00*	7214.27	31.00	565.67	0.0
	180	12	240	2310	22.60*	4318.54	22.60	320.99	0.0		270	18	360	4815	19.00*	7215.76	19.00	857.13	0.0
		20	240	2310	35.00*	7199.04	35.00	228.70	0.0			30	360	4815	31.00*	7216.75	31.00	750.56	0.0
70	35	7	105	2520	8.00	106.44	8.00	113.83	0.0	100	50	10	150	5100	11.00*	5003.63	11.00	222.28	0.0
		11	105	2520	12.00*	1834.72	12.00	106.95	0.0			16	150	5100	17.00*	7218.58	17.00	210.87	0.0
	56	11	126	2583	12.00	823.36	12.00	130.54	0.0		80	16	180	5190	17.00*	7219.34	17.20	254.29	0.2
		18	126	2583	19.00*	7219.10	19.00	114.08	0.0			26	180	5190	27.00*	7219.55	27.20	219.37	0.2
	70	14	140	2625	15.00*	6739.40	15.00	137.69	0.0		100	20	200	5250	21.00*	7214.37	21.00	296.72	0.0
		23	140	2625	23.00*	7213.12	23.00	129.46	0.0			33	200	5250	33.00*	7214.10	33.00	238.51	0.0
	105	14	175	2730	15.00*	7208.77	15.00	191.50	0.0		150	20	250	5400	21.00*	7209.18	21.00	423.98	0.0
		23	175	2730	24.00*	7209.88	24.00	174.34	0.0			33	250	5400	34.00*	7209.86	34.20	325.13	0.2
	140	14	210	2835	15.00*	7215.04	15.00	271.18	0.0		200	20	300	5550	21.00*	7210.20	21.00	582.60	0.0
		23	210	2835	24.00*	7215.46	24.00	244.17	0.0			33	300	5550	34.00*	7210.07	34.00	481.60	0.0
	175	14	245	2940	15.00*	5266.06	15.00	418.21	0.0		250	20	350	5700	21.00*	7214.75	21.00	955.59	0.0
		23	245	2940	24.00*	7212.86	24.00	292.64	0.0			33	350	5700	34.00*	7215.47	34.20	684.00	0.2
	210	14	280	3045	15.00*	7214.44	15.00	532.53	0.0		300	20	400	5850	21.00*	7212.96	21.00	1169.45	0.0
		23	280	3045	24.00*	7214.29	24.00	406.82	0.0			33	400	5850	34.00*	7213.10	34.00	1013.71	0.0
80	40	8	120	3280	9.00*	2052.80	9.00	140.54	0.0	**Avg**					21.00	6270.12	21.01	319.12	0.014
		13	120	3280	13.00*	7217.67	13.00	144.72	0.0	**#Best**									65
	64	12	144	3352	13.00*	5661.32	13.00	167.71	0.0										
		21	144	3352	21.00*	7217.17	21.00	148.49	0.0										
	80	16	160	3400	17.00*	7218.16	17.00	171.42	0.0										
		26	160	3400	27.00*	7218.77	27.00	149.07	0.0										
	120	16	200	3520	17.00*	7219.19	17.00	281.02	0.0										
		26	200	3520	27.00*	7219.76	27.00	205.11	0.0										
	160	16	240	3640	17.00*	7219.22	17.00	374.10	0.0										
		26	240	3640	27.00*	7219.73	27.00	329.79	0.0										
	200	16	280	3760	17.00*	7219.35	17.00	605.52	0.0										
		26	280	3760	27.00*	7220.03	27.00	420.54	0.0										
	240	16	320	3880	17.00*	7219.19	17.00	672.45	0.0										
		26	320	3880	27.00*	7219.82	27.00	585.43	0.0										

TABLE 3 Comparison between the solutions of B&C and GA on the Large instances.

probability d used to generate the edges (see Section 8.1) and the number of edges m. This last column reports an interval of values representing the minimum and the maximum number of edges contained inside the graphs of that scenario. Indeed, each line in the tables represents a scenario composed of five instances with the same characteristics but a different seed, and the results shown in each line are the average values of these five instances. The remaining

				GA	
n	**d**	**m**	**Opt**	**Obj**	**Time**
100	0.3	[1554 - 1615]	0.00	2.20	91.19
100	0.5	[2494 - 2550]	0.00	1.80	111.40
100	0.7	[3464 - 3566]	0.00	1.40	133.46
150	0.3	[3493 - 3587]	0.00	2.60	192.94
150	0.5	[5694 - 5747]	0.00	2.60	264.18
150	0.7	[7846 - 7935]	0.00	1.60	334.73
200	0.3	[6174 - 6307]	0.00	3.00	356.46
200	0.5	[10155 - 10262]	0.00	2.60	482.03
200	0.7	[13970 - 14134]	0.00	2.00	607.44
250	0.3	[9631 - 9779]	0.00	3.80	435.34
250	0.5	[15818 - 16026]	0.00	3.20	749.63
250	0.7	[21870 - 22083]	0.00	2.00	1106.88
300	0.3	[13753 - 14135]	0.00	3.20	763.16
300	0.5	[22649 - 22852]	0.00	3.00	1135.25
300	0.7	[31517 - 31622]	0.00	2.40	1552.63
350	0.3	[18838 - 19004]	0.00	3.20	1168.42
350	0.5	[30846 - 31038]	0.00	3.00	1749.42
350	0.7	[42757 - 42932]	0.00	2.60	2128.46
400	0.3	[24630 - 24776]	0.00	4.20	1457.40
400	0.5	[40340 - 40678]	0.00	3.80	2032.20
400	0.7	[56119 - 56498]	0.00	3.20	2646.19
Avg				2.73	928.51

TABLE 4 Computational results for the Hamiltonian instances.

three columns show the optimal solution value (*Opt*), the solution value (Obj) and the computational time ($Time$), in seconds, of GA, respectively. Notice that we do not report the column Gap in this table because, having an optimal solution value always equal to zero, this column coincides with the column Obj. Finally, at the bottom of the table, the Avg row reports the average values of Obj and $Time$.

The results of Table 4 show that these new instances are harder to solve for GA. Indeed, the algorithm does not find the optimal solution in these instances. However, the average gap value from the optimal solution is equal to 2.73 and only in one case this gap is over 4. From these results, we derive that GA remains effective even on this type of instance but less effective if compared with the results obtained on the previous benchmark instances. It is worth noting that as the density of the instances increases as the solution quality, provided by GA, improves. For instance, in scenario 100-0.3 the gap is equal to 2.20 while in scenario 100-0.7 this gap decreases to 1.40. We suspect that this occurs because, by increasing the number of edges inside the graph, we increase the chance to have more Hamiltonian cycles and then more optimal solutions. Even the computational time increases in these instances with an average value equal to 928.51 seconds and a peak equal to 2646.19 seconds in the largest scenario 400-0.7. Overall, GA

remains sufficiently fast despite the density of these graphs.

9 | CONCLUSION

In this paper, we developed a genetic algorithm to solve the 2ECMBV problem. This algorithm is based on a procedure able to find and remove useless edges from the feasible solution and some ad-hoc operators that increase the effectiveness of the procedure by performing a wider exploration of the solution space. We tested the performance of the genetic algorithm on benchmark and new instances with respect to accuracy and running time. The computational results show that our algorithm is very effective on the benchmark instances where only in one case it does not find the best/optimal solution. In the Hamiltonian instances, it is less effective, but the gap from the best/optimal solution remains low. Finally, GA results fast, with a computational time that is almost always lower than 1200 seconds.

Conflict of interest

The authors have no conflict of interests to declare.

Data availability statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

[1] F. Carrabs, R. Cerulli, C. D'Ambrosio, and F. Laureana, *The generalized minimum branch vertices problem: Properties and polyhedral analysis*, J. Optim. Theory Appl. **188** (2021), 356–377.

[2] F. Carrabs, R. Cerulli, M. Gaudioso, and M. Gentili, *Lower and upper bounds for the spanning tree with minimum branch vertices*, Comput. Optim. Appl. **56(2)** (2013), 405–438.

[3] R. Cerulli, M. Gentili, and A. Iossa, *Bounded-degree spanning tree problems: models and new algorithms*, Comput. Optim. Appl. **42** (2009), 353.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to algorithms, 4th edition*, The MIT Press, 2022.

[5] K.P. Eswaran and R.E. Tarjan, *Augmentation problems*, SIAM J. Comput. **5** (1976), 653–665.

[6] Z. Galil and G.F. Italiano, *Reducing edge connectivity to vertex connectivity*, SIGACT News **22** (Mar. 1991), 57–61.

[7] L. Gargano, P. Hell, L. Stacho, and U. Vaccaro, *Spanning trees with bounded number of branch vertices*, Int. Colloq. Automata, Languages, Program., Springer Berlin Heidelberg **2380** (2002), 355–365.

[8] A.A. Hagberg, P.J. Swart, and D.A. Schult, *Exploring network structure, dynamics, and function using networkx*, Proceedings of the 7th Python in Science Conference, Pasadena, CA USA, 2008, pp. 11 – 15.

[9] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.

[10] F. Laureana, *Polyhedral analysis and branch and cut algorithms for some np-hard spanning subgraph problems*, Ph.D. thesis, University of Salerno, 2019.

[11] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle, *The irace package: Iterated racing for automatic algorithm configuration*, Oper. Res. Perspectives **3** (2016), 43–58.

[12] A. Marín, *Exact and heuristic solutions for the minimum number of branch vertices spanning tree problem*, Eur. J. Oper. Res. **245** (2015), 680–689.

[13] J. Moreno, Y. Frota, and S. Martins, *An exact and heuristic approach for the d-minimum branch vertices problem*, Comput. Optim. Appl. **71** (2018), 829–855.

[14] K.A. Ravindra, L.M. Thomas, and B.O. James, *Network Flows, theory, algorithms, and applications*, Prentice-Hall, New Jersey, 1993.

[15] R.T. S. Khuller, *Approximation algorithms for graph augmentation*, J. Algorithms **14** (1993), 214–225.

[16] J.M. Schmidt, *A simple test on 2-vertex- and 2-edge-connectivity*, Informat. Process. Lett. **113** (2013), 241 – 244.

[17] R.M.A. Silva, D.M. Silva, M.G.C. Resende, G.R. Mateus, J.F. Gonçalves, and P. Festa, *An edge-swap heuristic for generating spanning trees with minimum number of branch vertices*, Optim. Lett. **8** (2014), 1225–1243.

[18] S. Silvestri, G. Laporte, and R. Cerulli, *A branch-and-cut algorithm for the minimum branch vertices spanning tree problem*, Comput. Oper. Res. **81** (2017), 322–332.

[19] R. Tarjan, *Depth-first search and linear graph algorithms*, SIAM J. Comput. **1** (1972), 146–160.

[20] R.E. Tarjan, *A note on finding the bridges of a graph*, Informat. Process. Lett. **2** (1974), 160–161.