Computers & Operations Research 179 (2025) 107034

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Multi-armed bandit for the cyclic minimum sitting arrangement problem ☆

Marcos Robles [a], Sergio Cavero [a], Eduardo G. Pardo [a,*], Oscar Cordón [b]

[a] Universidad Rey Juan Carlos, Spain
[b] Universidad de Granada, Spain

ARTICLE INFO

Keywords:
Signed graphs
Cyclic minimum sitting arrangement
Multi-armed bandit
Variable neighborhood descent

ABSTRACT

Graphs are commonly used to represent related elements and relationships among them. Signed graphs are a special type of graphs that can represent more complex structures, such as positive or negative connections in a social network. In this work, we address a combinatorial optimization problem, known as the Cyclic Minimum Sitting Arrangement, that consists of embedding a signed input graph into a cycle host graph, trying to locate in the embedding positive connected vertices closer than negative ones. This problem is a variant of the well-known Minimum Sitting Arrangement where the host graph has the structure of a path graph. To tackle the problem, we propose an algorithm based on the Multi-Armed Bandit method that combines three greedy-randomized constructive procedures with a Variable Neighborhood Descent local search algorithm. To assess the merit of our proposal, we compare it with the state-of-the-art method. Our experiments show that our algorithm outperforms the best-known method in the literature to date, and the results are statistically significant, establishing itself as the new state of the art for the problem.

1. Introduction

A Graph Layout Problem (GLP) consists of embedding a graph, denoted as the input graph, into another graph, denoted as host graph, while optimizing a certain objective function. An embedding (Cygan et al., 2017), which is also known in the literature as arrangement (Petit, 2003), labeling (Chung, 1988), ordering (Ravi et al., 1991), or layout (Dujmović and Wood, 2004; Pardo et al., 2018), consists of the vertex assignment of an input graph to the vertices of a host graph (Díaz et al., 2002). This embedding may require associating the edges of the input graph with one or more paths in the host graph.

Problems belonging to the GLP family are of great interest to the scientific community because they have applications in a wide range of fields, such as Very Large Scale Integration (VLSI) design (Newton, 1981), assignment of network resources (Sahhaf et al., 2015) and graph drawing (Petit, 2003), among others (Mitchison and Durbin, 1986; Ravi et al., 1991). In fact, any real-life problem that requires mapping the elements of a specific network into a particular layout could be modeled as a GLP. These layouts are typically modeled as graphs with a well-known structure, such as paths (Petit, 2003), cycles (Cavero et al., 2021), grids (Shafaei et al., 2013), or trees (Shiloach, 1979), among others. However, it is possible to find GLP in the literature defined on unstructured host graphs (Sahhaf et al., 2015).

In this paper, we study a combinatorial optimization problem belonging to the GLP family. Particularly, we tackle the Cyclic Minimum Sitting Arrangement (CMinSA) problem (Benítez et al., 2018), a variant of the well-known Minimum Sitting Arrangement (MinSA) problem (Cygan et al., 2012; Pardo et al., 2020). In both problems, each of the edges in the input graph is assigned either a positive or a negative sign. The main difference between them is that the MinSA is defined on a path host graph (as it is represented in the example of Fig. 1(b)) and the CMinSA is defined over a cycle host graph (as it is represented in the example of Fig. 1(d)).

The goal of the CMinSA is to find an embedding where vertices of the input graph connected by positive edges are placed in closer positions with respect to other adjacent vertices connected by a negative edge. In this assignment, distance is measured as the number of vertices of the shortest path connecting both vertices in the embedding. Every time that the previous situation is not satisfied (i.e., for a given vertex of the input graph, a vertex connected by a negative edge is placed closer in the embedding than a vertex connected with a positive edge),

☆ This research has been partially supported by grants: PID2021-125709OA-C22 and PID2021-126605NB-I00, funded by MCIN/AEI/10.13039/501100011033 and "ERDF A way of making Europe"; project CIAICO/2021/224, funded by Generalitat Valenciana; project M2988, funded by "Proyectos Impulso de la Universidad Rey Juan Carlos 2022"; the "Cátedra en Innovación y Digitalización Empresarial funded by Universidad Rey Juan Carlos and Second Episode" (Ref. MCA06); and "Red Española de optimización heurística 4.0 digitalización" (Ref. RED2022-134480-T); project CIRMA-CM (Ref. TEC-2024/COM-404) funded by "Comunidad Autónoma de Madrid"; and TSI-100930-2023-3 (MCA07) funded by "Ministerio para la Transformación Digital y de la Función Pública".

* Corresponding author.
E-mail addresses: marcos.robles@urjc.es (M. Robles), sergio.cavero@urjc.es (S. Cavero), eduardo.pardo@urjc.es (E.G. Pardo), ocordon@ugr.es (O. Cordón).

https://doi.org/10.1016/j.cor.2025.107034
Received 3 September 2024; Received in revised form 19 February 2025; Accepted 20 February 2025
Available online 1 March 2025

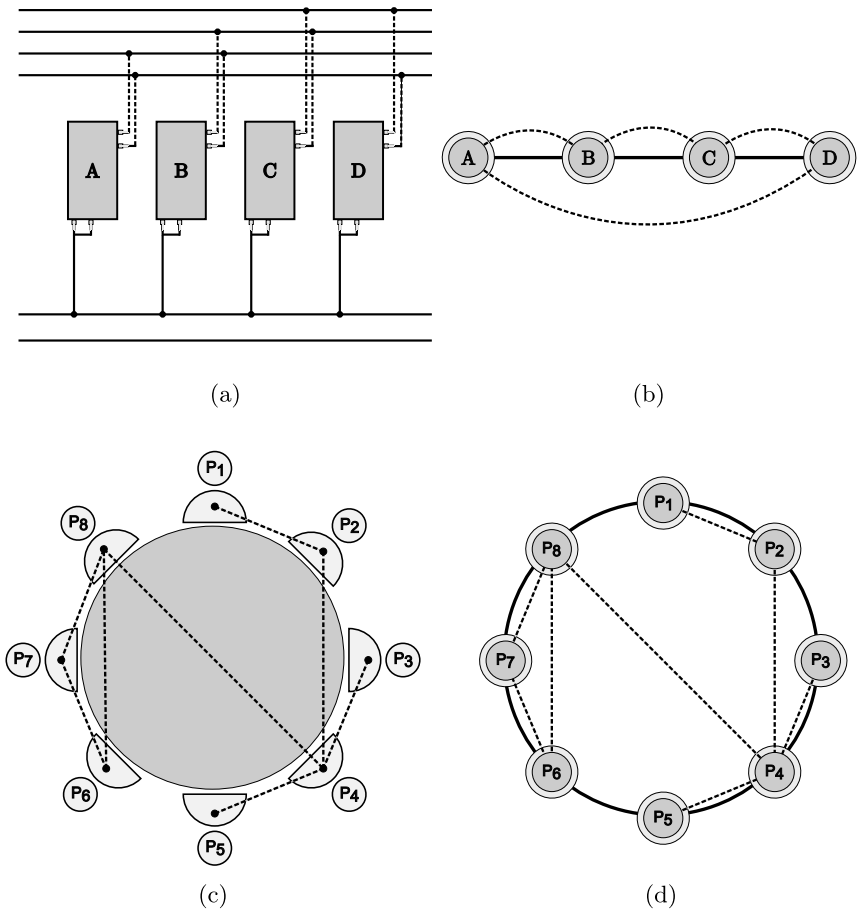


Fig. 1. (a) Example of a real application where chips and connectors (input graph) are placed in a circuit (path host graph). (b) Visual representation of the embedding resulting from the example of Fig. 1a. (c) Example of a social network of people (input graph) assigned to seats in a round table (cycle host graph). (d) Visual representation of the embedding resulting from the example of Fig. 1c.

an error is quantified. Therefore, each vertex of the input graph can produce zero, one, or multiple errors in the embedding. The goal of the CMinSA is to minimize the objective function which is calculated as the total number of errors in the embedding. The formal definition of the CMinSA problem and its objective function is presented in Section 2.

In Fig. 1, we show two examples of real-world applications where the problems can be modeled as a graph and it is necessary to embed them into two different graphs with a known structure, one in a path and the other in a cycle. Specifically, the example in Fig. 1(a) represents a VLSI circuit made of chips and connectors (Newton, 1981), which needs to be placed on a board. The chips and their connections can be modeled as a graph (input graph) where the chips are represented as vertices and the connections are represented as edges. Similarly, the positions to place the chips and their connections can also be modeled as a graph (host graph). In this case, the host graph has the structure of a path graph. For this example, the corresponding embedding of the input graph in the path host graph is represented in Fig. 1(b). In this real-world problem, there are several objectives to optimize, such as the number of connector crossings (Cavero et al., 2021) or the total length of the connectors used (Petit, 2003).

The example in Fig. 1(c) represents a situation in which it is necessary to distribute a group of people on a circular table (Cavero et al., 2022b; Lozano et al., 2013). This situation may arise in many different and daily contexts, such as a wedding or conference banquet. People can be represented as a complex network of guests (a social network) which might include relationships among them. This network can be modeled as a graph, where the vertices represent the people who are connected by edges that represent the relations between the guests. In this case, a cycle host graph is used to represent the seats of the table. The corresponding embedding from the real example shown in Fig. 1(c) is represented in Fig. 1(d), where the embedding is done on a cycle host graph. One possible objective of this problem is to seat people who get along well as closely as possible (Cavero et al., 2022b), or to avoid seating adversaries close to each other (Lozano et al., 2013).

To tackle the CMinSA, we present a novel approach that leverages the strengths of three constructive methods and adaptive search strategies. Our method combines randomized and adaptive multi-start techniques, incorporating various constructive methods tailored to the structure of the input graph. The selection of these methods is guided by a reinforcement learning algorithm derived from probability theory and machine learning. Specifically, we use a Multi-Armed Bandit algorithm (Bouneffouf et al., 2020; Lattimore and Szepesvári, 2020; Slivkins, 2019) that uses information from previous iterations to choose the most effective constructive approach for a given input graph. This ensures that our algorithm remains responsive to the unique characteristics of each instance. Furthermore, the generated solutions undergo an efficient intensification phase that explores two neighborhoods following a Variable Neighborhood Descent scheme (Hansen et al., 2019). In addition, the proposed local searches are enriched by a fast evaluation of the objective function and a neighborhood reduction technique. Our proposed algorithm consistently outperforms the current state-of-the-art method for the problem, highlighting its potential to effectively address the complexities of the CMinSA problem.

In the following sections, we formally define the CMinSA (Section 2) and then review the literature on the problem (Section 3). Next, we present our proposal encompassing a detailed explanation of our general framework, heuristic and metaheuristic methods, and advanced strategies (Section 4). Then, we describe the experimental setup, detailing the instances used, and presenting a comprehensive comparison with the best previous state-of-the-art method (Section 5). Finally, we conclude presenting our most relevant findings and contribution, outlining implications for future research (Section 6).

2. Problem definition

To formally define the CMinSA, we first need to establish some fundamental notation. Let $G = (V_G, E_G, E_G^+, E_G^-)$ be a signed, finite, simple, and undirected input graph, where V_G denotes the set of vertices, and E_G represents the set of edges, which is partitioned into the subsets E_G^+ and E_G^-. A signed graph is a specific type of graph where each edge is assigned a weight of either +1 or −1. Edges with a weight of +1 are denoted as *positive edges* and belong to the set E_G^+. It is worth mentioning that a pair of vertices u and v in G such that $(u, v) \in E_G^+$ is called a pair of *positively connected vertices*. Analogously, a pair of vertices u and v in G such that $(u, v) \in E_G^-$ is called a pair of *negatively connected vertices*. Consequently, E_G fulfills the condition $E_G = \{E_G^+ \cup E_G^-\}$, indicating that it is the union of the sets of positive and negative edges. For the sake of simplicity, in the rest of the paper, we graphically label edges just with a "+ " or "−" sign, representing weights +1 and −1, respectively.

Given an input graph G, we can now define the host graph for the CMinSA problem. Let $H = (V_H, E_H)$ be a finite, simple, and undirected cycle graph, where V_H and E_H represent the sets of vertices and edges, respectively. The cycle host graph must satisfy the following properties:

- $|V_H| = |V_G| \geq 3$, i.e., the number of vertices of the host graph equals the number of vertices of the input graph.
- $|E_H| = |V_H|$, i.e., the number of edges equals the number of vertices.
- The degree of every vertex in V_H is 2, i.e., each vertex of V_H is connected to exactly two other vertices of V_H.
- The edges of E_H are arranged in a way that there is exactly one cycle in the graph.

These properties ensure that the host graph has the same number of vertices as the input graph, and that it is possible to find a bijection between the vertices of the two graphs as it is commonly required in a GLP.

In this context, a path in a graph is a sequence of vertices such that each vertex is connected to the next vertex by an edge. Formally, given two vertices $v_i, v_j \in V_H$, such that $1 \leq i < j \leq |V_H|$, a path without cycles from v_i to v_j is a sequence of vertices $v_i, v_{i+1}, \ldots, v_j$ such that $(v_i, v_{i+1}) \in E_H$. Notice that no vertex appears more than once in the path. Note that since H' is a cycle graph, there are two possible paths to get from one vertex to any other.

Let P_H be the set of all possible paths without cycles in the host graph between any pair of vertices. Then, we denote by $P_{v_i,v_j} \subset P_H$ as the subset of paths from v_i to v_j and $p_{min}(v_i, v_j)$ as the path with the minimum cardinality in P_{v_i,v_j} (i.e, it is the "shortest path" from v_i to v_j). Notice that in the case that $|H|$ is even, the set of shortest paths from v_i to v_j might have two elements. In this case, we evaluate both paths using the objective function of the problem tackled and select the one that minimizes it. If a tie remains, then the path which follows a clockwise order is selected.

Given an input graph $G = (V_G, E_G)$ and a host graph $H = (V_H, E_H)$, an embedding of G into H is a pair of functions (φ, ψ) that satisfy the following conditions:

1. $\varphi : V_G \to V_H$ is a bijection.
2. $\psi : E_G \to P_H$ is a function that maps each edge $(u, v) \in E_G$ to a path $p \in P_H$ such that:
   - p is a path from $\varphi(u)$ to $\varphi(v)$ in H.
   - p is a shortest path from $\varphi(u)$ to $\varphi(v)$ in H.
   - If there are multiple paths which results to be the shortest, p is chosen to minimize the number of errors in the embedding (as defined by the objective function of the problem).
   - If there is still a tie, the path following a clockwise order is selected.

In other words, an embedding is a mapping of the vertices of G to the vertices of H using the function φ, such that each edge in G is mapped to a carefully selected shortest path in H that connects the corresponding vertices of the edge, using the function ψ. The selection of the path, when multiple shortest paths exist, is based on minimizing errors and following a consistent tie-breaking rule.

Given a specific φ, the function ψ is uniquely determined by the rules described above. Therefore, for the rest of the paper, we will often refer to an embedding simply by its φ function, with the understanding that the corresponding ψ function is implicitly defined according to these rules.

With the previous definitions at hand, the CMinSA looks for an embedding of the vertices of the input graph such that vertices connected with positive edges are placed closer in the embedding than those with a negative connection. Specifically, every time a negatively connected vertex appears in the path between two positively connected vertices a conflict occurs. This concept has previously been referred to in the literature as an error in an input vertex for a particular embedding, simply denoted as "error" (Aracena and Thraves Caro, 2023; Benítez et al., 2018). Therefore, to compute the number of errors of a vertex $v \in V_G$ we consider the paths that connect v to any adjacent vertex in E_G^+. Then, an error is produced if any other adjacent vertex, negatively connected to v, is found in the considered paths. Therefore, the quality of an embedding for the CMinSA is computed as the sum of the errors associated with all the vertices of the input graph when embedded in the host graph. Notice that one vertex might produce zero, one, or multiple errors for a given embedding.

Formally, given three vertices u, v, and w in V_G, where (u, v) is an edge in E_G^+ and (u, w) is an edge in E_G^-, an error occurs if $\varphi(w)$ is in $\psi((u, v))$. We mathematically denote the error of a vertex $u \in V_G$ in an embedding φ as $\mathcal{E}(u, \varphi)$. Finally, the objective function for a solution φ, denoted as $\mathcal{E}(\varphi)$, is computed as follows:

$$\mathcal{E}(\varphi) = \sum_{u \in V_G} \mathcal{E}(u, \varphi). \tag{1}$$

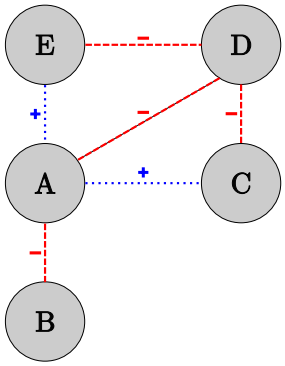
Finally, the goal of the CMinSA problem is to find an embedding φ^*, among the set of all feasible embeddings $\boldsymbol{\Phi}$, that minimizes the overall number of errors, which is formally defined as follows:

$$\varphi^* = \underset{\varphi \in \boldsymbol{\Phi}}{\operatorname{argmin}}\, \mathcal{E}(\varphi). \tag{2}$$

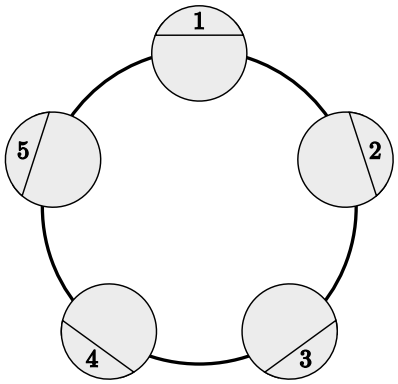
To illustrate the concepts introduced in this section, let us consider the example depicted in Fig. 2. Particularly, in Fig. 2(a) we show a signed input graph, G'. Additionally, positive edges are represented with blue dotted lines and negative edges with red dashed lines. In this case, we have that $V_{G'} = \{A, B, C, D, E\}$, and that $E_{G'} = \{(A, B), (A, C), (A, D), (A, E), (D, E)\}$. The subset of positive edges consists of the edges $E_{G'}^+ = \{(A, C), (A, E)\}$ and the subset of negative edges consists of the edges $E_{G'}^- = \{(A, B), (A, D), (D, E)\}$.

Similarly, in Fig. 2(b) we show an example of a cycle host graph H' for the graph G' with 5 vertices and 5 edges since $|V_{G'}| = 5$. The set of vertices of H' is $V_{H'} = \{1, 2, 3, 4, 5\}$ and the set of edges is $E_{H'} = \{(1, 2), (2, 3), (3, 4), (4, 5), (5, 1)\}$.

Based on this cycle host graph, we depict an example of the computation of the paths between two vertices. In particular, the paths without cycles from vertex 1 to vertex 4 are $P_{1,4} = \{\{1, 2, 3, 4\}, \{1, 5, 4\}\}$.



(a) An example of a signed input graph G'.



(b) An example of a cycle host graph H'.

Fig. 2. Examples of a signed input graph and a cycle host graph.

Then, the shortest path that joins 1 and 4 is $p_{min}(1,4) = \{1,5,4\}$ since it is the path in $P_{1,4}$ with the minimum cardinality.

Next, in Fig. 3, we show an example of an embedding (φ', ψ') of the input graph G', previously introduced, into the host graph H'. In this case, φ' is defined as: $\varphi'(A) = 1$, $\varphi'(B) = 2$, $\varphi'(C) = 3$, $\varphi'(D) = 4$, and $\varphi'(E) = 5$. Similarly, ψ' is defined as: $\psi'((A,B)) = \{\{1,2\}\}$, $\psi'((A,C)) = \{\{1,2,3\}\}$, $\psi'((A,D)) = \{\{1,5,4\}\}$, $\psi'((A,E)) = \{\{1,5\}\}$, and $\psi'((E,D)) = \{\{5,4\}\}$.

To evaluate the solution for the CMinSA of the embedding φ' let us first consider the vertex A. In this case, its positive neighbors are C and E, assigned to vertices 3 and 5, respectively. Then, the paths that ψ' assigns to the positive neighbors of A are $p_{min}(\varphi'(A), \varphi'(C)) = \{1,2,3\}$ and $p_{min}(\varphi'(A), \varphi'(E)) = \{5,1\}$, respectively. Additionally, negative neighbors are B and D, assigned to vertices 2 and 4, respectively. Since $\varphi'(B) = 2$ is on the path $p_{min}(\varphi'(A), \varphi'(C))$, vertex A has an error, so $\mathcal{E}(A, \varphi') = 1$. Similarly, we evaluate the vertices B, C, D, and E, obtaining no errors in any of them.

It is worth mentioning that vertex B has only one adjacent vertex, so it cannot produce errors. Analogously, vertex D has only negative edges so it cannot produce errors either (this would also happen if all adjacent vertices were connected by positive edges). Vertex C does not have any errors because $\varphi'(D) = 4$ is not included in the path $p_{min}(\varphi'(C), \varphi'(A)) = \{3,2,1\}$. Finally, in the case of vertex E it has zero errors, because $\varphi'(D) = 4$ is not in the path $\{5,1\}$. Therefore, the value of the objective function for the solution is $\mathcal{E}(\varphi') = 1$.

Finally, we evaluate the embedding φ' depicted in Fig. 3. Let us first consider the vertex A. In this case, its positive neighbors are C and E, assigned to vertices 3 and 5, respectively. Then, the paths that ψ' assigns to the positive neighbors of A are $p_{min}(\varphi'(A), \varphi'(C)) = \{1,2,3\}$ and $p_{min}(\varphi'(A), \varphi'(E)) = \{5,1\}$, respectively. Additionally, negative neighbors are B and D, assigned to vertices 2 and 4, respectively. Since $\varphi'(B) = 2$ is on the path $p_{min}(\varphi'(A), \varphi'(C))$, vertex A has an error, so $\mathcal{E}(A, \varphi') = 1$. Similarly, we evaluate the vertices B, C, D, and E, obtaining no errors in any of them.

It is worth mentioning that vertex B has only one adjacent vertex, so it cannot produce errors. Analogously, vertex D has only negative edges so it cannot produce errors either (this would also happen if all adjacent vertices were connected by positive edges). Vertex C does not have any errors because $\varphi'(D) = 4$ is not included in the path $p_{min}(\varphi'(C), \varphi'(A)) = \{3,2,1\}$. Finally, in the case of vertex E it has zero errors, because $\varphi'(D) = 4$ is not in the path $\{5,1\}$. Therefore, the value of the objective function for the solution is $\mathcal{E}(\varphi') = 1$.

3. Literature review

The CMinSA problem was first introduced in 2018 as a decision problem, aiming to determine whether a given input graph could be

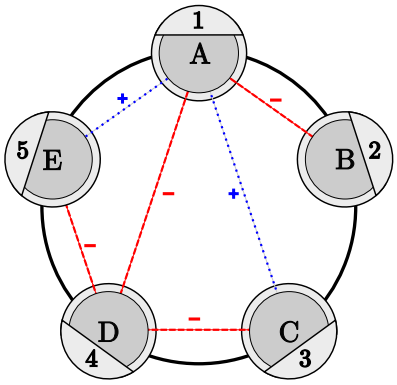


Fig. 3. An example of an embedding of G' in H' denoted as (φ', ψ').

embedded in a cycle graph without errors (Benítez et al., 2018). In this seminal work, the authors demonstrated that an error-free embedding is possible if the positive edges of the input graph form a circular arc graph. This type of graph ensures that the vertices form intersecting arcs, creating a cycle. However, it was not until 2022 that the Cyclic Minimum Sitting Arrangement was explored as an optimization problem (Robles et al., 2022). In this study, the objective was to minimize the number of errors in the embedding, and a Basic Variable Neighborhood Search (BVNS) algorithm (Hansen et al., 2019) was proposed to address the problem. This method was complemented by a constructive procedure inspired by the properties of circular arc graphs identified by Benitez (Benítez et al., 2018). The procedure begins by generating a list of cliques based solely on the positive edges of the input graph. These cliques are then assigned to consecutive positions in the embedding, ensuring no errors occur. The vertices within each clique are ordered by decreasing size and inserted sequentially. To date, these are the most significant works focusing on the CMinSA.

It is worth mentioning that the CMinSA is closely related to its linear variant, the MinSA, where the embedding is done on a path host graph. The MinSA problem can be traced back to 2011, when the Sitting Arrangement (SA) decision problem was first proposed (Kermarrec and Thraves, 2011). This problem seeks to determine whether a signed input graph can be embedded in a path host graph without errors. Cygan et al. (2012) extended this work by examining the complexity of the problem. They proved that it is NP-complete to decide if an input signed graph has a valid 1-dimensional drawing for general graphs and NP-hard to find the smallest valid dimension for such a drawing. In addition, they conducted a study similar to Benítez et al. (2018),

focusing on specific types of graphs. They found that a complete graph has a valid 1-dimensional space drawing if the positive edges form an interval graph, which is analogous to a circular arc graph for a path host graph. In 2015, the MinSA problem was introduced as an optimization variant of the SA (Pardo et al., 2015). To tackle this problem, the authors presented a Greedy Randomized Adaptive Search Procedure (GRASP) (Resende and Ribeiro, 2016) that combines a greedy randomized construction followed by a local search method. Later, in 2020, a BVNS was proposed for the problem (Pardo et al., 2020), which remains the state-of-the-art method for the MinSA. The literature on MinSA provides valuable information to define new strategies for the CMinSA.

The relationship between the MinSA and CMinSA problems was established in 2022 (Robles et al., 2022). This work demonstrated the connection between the two problems and adapted the BVNS algorithm, initially proposed for the MinSA (Pardo et al., 2020), to optimize the CMinSA. Both MinSA and CMinSA pose significant challenges, which is made evident due to their NP-hard complexity, that was proven in Cygan et al. (2012). Therefore, exact algorithms such as branch and bound/price methods or mathematical models have not been employed to solve these problems due to their computational complexity, leading researchers to utilize heuristic methods. Notably, heuristic methods, such as greedy constructive procedures (Stützle and Ruiz, 2018) or local search algorithms (Hansen et al., 2019), have been crucial to obtain good quality solutions in reasonable computing times for these problems.

Beyond the foundational work on CMinSA and MinSA, recent years have seen a growing interest in problems involving graph embeddings. These problems, which share similar applications, characteristics, and properties with the CMinSA problem, can provide valuable insights and techniques for its resolution (Díaz et al., 2002; Pardo et al., 2015). Among these, we find problems that involve embedding a non-weighted graph into a cycle host graph, such as the Cyclic Bandwidth Problem (CBP), the Cyclic Bandwidth Sum Problem (CBS), the Cyclic AntiBandwidth Problem (CAB), and the Cyclic Cutwidth Minimization Problem (CCMP).

The CBP aims to minimize the maximum distance between any two adjacent vertices in the embedding (Lin, 1995), while the CBS aims to minimize the sum of the distances between any two adjacent vertices in the embedding (Cavero et al., 2022b). Both problems are closely related to the CMinSA problem since, as empirically observed, reducing the distance between positively connected vertices helps to decrease the number of errors in the solution. The CAB aims to maximize the minimum distance between any two adjacent vertices in the embedding (Cavero et al., 2022a). This situation is similar to CMinSA: when maximizing the distance between negatively connected vertices, the number of errors in the solution is reduced. Finally, the CCMP aims to minimize the congestion of each pair of adjacent vertices in the host graph. In other words, it tries to avoid locating adjacent vertices of the input graph far away in the host graph (Cavero et al., 2021). Like the CBP or the CBS, the CCMP and the CMinSA can be compared in the way in which adjacent vertices have to be as close as possible. One of the main differences between the CMinSA and the previously introduced problems is that none of them have been studied for signed input graphs.

4. Algorithmic approach

In this section, we present the methodological background and our algorithmic proposal for tackling the CMinSA problem. Specifically, we propose the use of a Multi-Armed Bandit hyperheuristic which combines several heuristic procedures for generating and improving solutions for the problem. The general framework, and the constructive and local search procedures are introduced next.

4.1. Multi-Armed Bandit hyperheuristic

In this paper, we propose a Multi-Armed Bandit (MAB) hyperheuristic for the CMinSA. To that aim, we briefly introduce the concept of hyperheuristic and the methodological principles behind MAB. Then, we describe our specific adaptation of the methodology for the CMinSA.

Methodological background

A hyperheuristic operates as a high-level methodology that performs a search over a space of heuristics, dynamically selecting or designing low-level heuristics based on the problem studied. The fundamental mechanism relies on a learning algorithm that processes feedback from the search process to determine effective combinations of available heuristics and components (Almeida et al., 2020). This approach is particularly relevant in online hyperheuristic implementations, where decisions within the high-level search space are informed by continuous feedback from the performance of low-level heuristics during the search process, thereby managing the selection among various heuristic components with uncertain rewards. Such methodologies are formally categorized as selection hyperheuristics (Dokeroglu et al., 2024), characterized by their dynamic heuristic selection mechanism at each iteration of the optimization process.

MAB is a reinforcement learning algorithm, that has been used as a well-established framework in the field of computer science, operations research, economics, and statistics, since it provides a robust mechanism for decision-making under uncertainty (Bouneffouf et al., 2020; Lattimore and Szepesvári, 2020; Slivkins, 2019). The "Multi-Armed Bandit" concept is derived from a gambler's dilemma of choosing among multiple slot machines, or "one-armed bandits", each with uncertain payouts, also denoted as rewards (Clotfelter and Cook, 1993). The challenge lies in the trade-off between sticking with a machine that has yielded high payoffs in the past, or exploring other machines that might offer higher future payoffs (Sutton and Barto, 2018). There are several variants of the MAB framework. The most relevant is the classic MAB. This method considers arms with unknown reward distributions and aims to maximize long-term reward, by identifying high-reward arms without wasting tries on low-reward arms, thus minimizing the regret of choosing those low-reward arms. Notable algorithms for the classic MAB include the Upper Confidence Bound version 1 (UCB1) (Chu et al., 2011; Li et al., 2010), Epsilon-Greedy strategy (Kuleshov and Precup, 2014), and the Softmax (Kuleshov and Precup, 2014).

From an optimization perspective, the core challenge in the MAB framework is the trade-off between "exploration" and "exploitation". That is, the algorithm must balance the acquisition of new knowledge (exploration) with optimizing decisions based on existing knowledge (exploitation) to maximize the total value over a given time period divided into turns (Almeida et al., 2020; Meidani et al., 2022). In the context of our work on the CMinSA, the MAB framework can be particularly useful to guide the search process, helping to balance the exploration of new solutions using different constructive approaches with the exploitation through local search strategies.

In this research, we tested different variants of the MAB, in an increasing order of complexity, finally opting for UCB1, since it was the most successful one (Chu et al., 2011; Li et al., 2010). UCB1 is a multi-armed bandit algorithm, also known as "optimism under uncertainty". This algorithm belongs to the subset of adaptative exploration MAB methods, which defines lower and upper bounds in order to identify underperforming arms as soon as possible. These bounds are defined as $UCB_t(a) = \bar{\mu}(a) + r_t(a)$ and $LCB_t(a) = \bar{\mu}(a) - r_t(a)$, where $\bar{\mu}(a)$ is the average reward of the arm a and $r_t(a)$ is a confidence radius at turn t. Therefore, the decisions are made using information from previous turns and a measure of uncertainty or variance. Specifically, the UCB1 algorithm operates by maintaining only an upper confidence bound for the expected reward of each arm. This bound is calculated based on the $UCB_t(a)$ formula previously mentioned. The arm with the highest

upper confidence bound is selected for the next turn. Since the goal of the CMinSA is to minimize errors (instead of maximizing revenue) we have to adapt UCB1. Instead of using the upper confidence bound, we use the equivalent Lower Confidence Bound (LCB). The rest of the algorithm is adjusted accordingly. Next, we detail the adapted UCB1.

Formally, let us denote $N_t(a)$ as the number of times the arm a has been played up to turn t. Let $R_{t,a}$ denote the reward obtained from employing arm a at turn t, where the reward is the objective function evaluation of the solution generated by that arm. Then, the average reward received from arm a, from turn $t' = 1$ up to t, denoted $X_t(a)$, is given by:

$$X_t(a) = \frac{\sum_{t'=1}^{N_t(a)} R_{t',a}}{N_t(a)}. \tag{3}$$

With this definition, the LCB for arm a at turn t is then calculated by:

$$LCB_t(a) = X_t(a) - \sqrt{\frac{2 \ln t}{N_t(a)}}. \tag{4}$$

The arm selected at turn t is the one with the smallest LCB:

$$A_t = \arg\min_a LCB_t(a). \tag{5}$$

This ensures that the algorithm explores arms that have not been played often (since the uncertainty term $\sqrt{\frac{2 \ln t}{N_t(a)}}$ is large when $N_t(a)$ is small) but also exploits arms that have a high average reward. Note that as the initial value of N_t is 0 for all arms, the previous formula would provoke a situation where there is a division by zero. To avoid this drawback, initially all arms are played once in a random order to have starting information. The logarithmic term ensures that all arms will be explored infinitely often as t goes to infinity, but the rate of exploration decreases as the total number of turns increases. This allows the algorithm to automatically converge to the optimal arm while still maintaining some level of exploration. Therefore, the MAB could be seen as an optimization problem itself, which aims for minimizing a regret value over a certain number of decisions.

In this paper, the MAB framework is applied to the field of heuristic optimization (Burke et al., 2013; Drake et al., 2020), to operate as a hyperheuristic by providing a systematic approach to address the uncertainty inherent in heuristic selection.

Algorithmic proposal

In this paper, the proposed MAB is configured with multiple heuristics, each corresponding to an arm of the method, and the best one is chosen during the runtime. A visual representation showing the behavior of this proposal is shown in Fig. 4. This flowchart follows the activity diagram standard described in the Unified Modeling Language (UML) (Jacobson et al., 1999). Consequently, each rectangular element denotes a distinct task or activity within the process, while diamonds represent decision points depending on specific conditions. A rectangle with the upper right corner bent represents a comment, and it is related to specific diagram components using a dotted line.

As it is shown in Fig. 4, three different arms are proposed for tackling the CMinSA. Each arm consists of a construction phase and an improvement phase. The construction phase of each arm uses a different constructive algorithm, denoted as Constructive 1, 2, and 3, to generate a new initial solution constructed from scratch at each iteration. The solutions generated by arms 1 and 2 then undergo a subsequent local search. Finally, the resulting solutions from all three arms are improved using a shared improvement procedure. It is important to note that only after the improvement phase is the result used to update the LCB values, which will determine the selection of the arm in the next iteration.

We complement the previous description with the pseudocode for the UCB1 method reported in Algorithm 1. The algorithm starts by setting the running time ($time$) and the number of turns (t) to 0 (steps 1 and 2). Then it initializes the rewards (step 3) and the LCBs (step 4) for all arms by running each once, increasing the value of t in one unit for each arm run. Then, the best solution is set to an empty solution (step 5).

In the main loop (steps 6–16), the algorithm selects the arm with the lowest LCB (step 7). Then, it constructs a solution using the selected arm (step 8) and improves the solution using the improving phase (step 9). Specifically, the construction phase (step 8) is the one that differs among arms, differing on how solution construction is done. On the other hand, the improvement phase (step 9) is the same for all arms. If the improved solution is better than the best solution, the best solution is updated (steps 10 and 11). Finally, the average reward of that arm and the LCBs for all arms are updated, and the number of played turns is incremented (steps 13, 14 and 15).

Finally, the main loop ends when the elapsed time reaches the maximum time (step 16). The best solution is then returned (step 17).

Algorithm 1 Multi-Armed Bandit — UCB1

Input: Maximum time T, input graph G

1: $time = 0$
2: $t = 0$ {Set current turn}
3: Initialize rewards $X_t(a)$ for all arms
4: Initialize $LCB_t(a)$ for all arms
5: $\varphi^* = \emptyset$ {Initialize best solution}
6: **while** $time < T$ **do**
7: $a = \arg\min_a LCB_t(a)$ {Select an arm a}
8: $\varphi' = \texttt{ConstructivePhase}(G, a)$ {Construction of solution φ' using arm a}
9: $\varphi'' = \texttt{ImprovementPhase}(\varphi')$ {Improvement of solution, common to all arms}
10: **if** $\mathcal{E}(\varphi'') < \mathcal{E}(\varphi^*)$ **then**
11: $\varphi^* = \varphi''$
12: **end if**
13: Update reward $X_t(a)$ for arm a
14: Update lower confidence bounds $LCB_t(a)$ for all arms
15: $t = t + 1$
16: **end while**
17: **return** φ^* {Best solution}

As a final remark, the method proposed in this paper is a selection hyperheuristic (Dokeroglu et al., 2024) since, at each step, it dynamically selects a heuristic, generates a solution and evaluates its performance. This frequent use of the best heuristic results in high-quality solutions. Specifically, the UCB1 algorithm chooses the best arm (constructive or constructive followed by local search) among those available. Note that this strategy could also be extended to the improvement phase. However, in this case, the experiment performed has shown it to be unnecessary considering the local search procedures proposed.

In Sections 4.2 and 4.3, we provide a detailed explanation of the constructive and improvement heuristics, respectively, of the UCB1 algorithm.

4.2. Construction phase

This section introduces a construction phase for providing initial solutions to our MAB algorithm. Particularly, the constructive procedure introduced in this proposal has been inspired by the GRASP methodology (Feo and Resende, 1995; Resende and Ribeiro, 2010), which can be easily tailored for the integration within MAB. Next, we review the methodological background of GRASP and then, we describe our specific proposal.

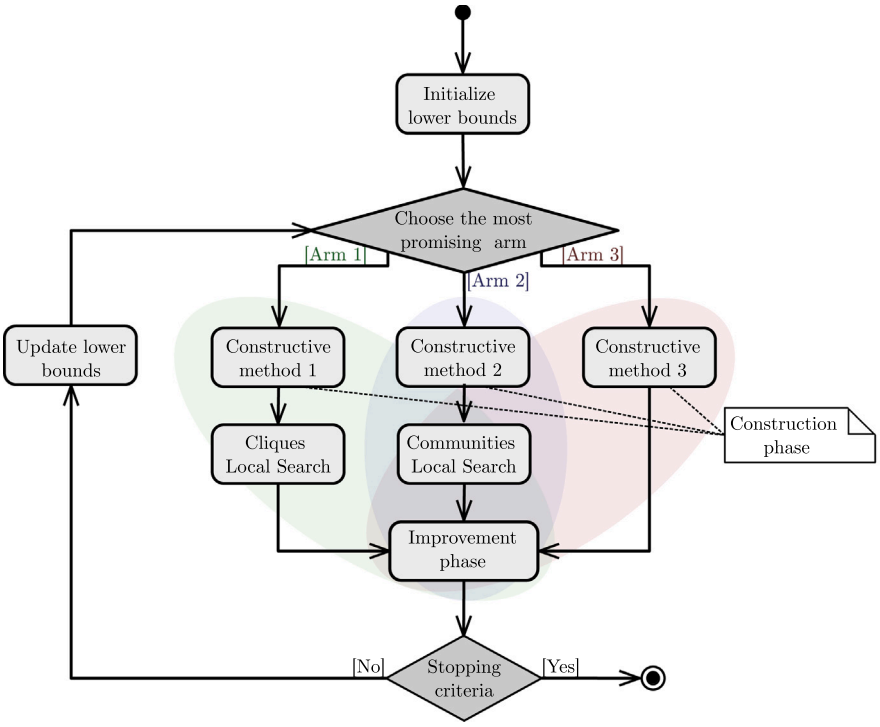


Fig. 4. UML activity diagram representing the MAB algorithm.

Methodological background

We focus our attention in the construction phase of GRASP methodology, which introduces a balance between greediness and randomness in the construction phase. Specifically, solutions are constructed from scratch by adding elements to the solution one by one. In order to select the next element to be added to the solution, a greedy criterion g is used to evaluate the candidate elements, which are stored in a Candidate List (CL). However, instead of adding the element which maximizes the greedy criterion, a percentage of the best elements, determined by a search parameter α, are selected conforming a Restricted Candidate List (RCL), and one of them is chosen at random from the list. Therefore, the parameter $\alpha \in [0, 1]$, which must be tuned experimentally, balances the greediness/randomness of the procedure, i.e., in this case, higher values of α increase randomness, while lower values of α favor greediness.

Algorithm 2 outlines the constructive algorithm proposed. It initializes an empty solution (step 1) and generates the CL of elements from the input graph G (step 2). Then it evaluates the candidates using a greedy criterion g to obtain the maximum and the minimum values (steps 4 and 5) respectively. These values are used to compute a threshold (th, in step 6) using α. Based on this threshold, it generates an RCL, which includes only those candidates v, whose evaluation of g, denoted as $g(v)$, met the condition $g_{\min} \leq g(v) \leq th$ (step 7). Finally, a random element is selected from the RCL (step 8), and is added to the solution being constructed (step 9). The process continues until a solution is completed and then returned (step 12).

Algorithm 2 Constructive algorithm

Input: Parameter α, greedy criteria g, input graph G
1: $\varphi' = \emptyset$ {Initialize empty solution}
2: $CL = \texttt{GenerateCandidateList}(G)$
3: **while** φ' is not complete **do**
4: $g_{\max} = \arg\max_v g(v)$
5: $g_{\min} = \arg\min_v g(v)$
6: $th = g_{\min} + \alpha \cdot (g_{\max} - g_{\min})$
7: $RCL = \texttt{GenerateRestrictedCandidateList}(CL, th)$
8: $v = \texttt{GetRandomCandidate}(RCL)$
9: $\varphi' = \varphi' \cup \{v\}$ {Add a random candidate from RCL}
10: $CL = \texttt{UpdateCandidateList}(CL, v)$
11: **end while**
12: **return** φ'

Algorithmic proposal

In this paper, we propose three different constructive procedures based on GRASP. The main difference between each constructive is the greedy criterion g used to evaluate the candidates from the input graph, to be added to the solution at each step (see steps 4 and 5 of Algorithm 2). The specific greedy criterion g, denoted as g_1, g_2, and g_3, will be further described in the following sections.

Given a vertex from the input graph selected with any of the proposed criteria, then it has to be assigned to a vertex of the host graph. The strategy we follow to perform this assignment is shared by all the proposed constructive methods. In particular, in the first assignment, we randomly select a host vertex. Then, each candidate vertex from the input graph is assigned to a host vertex considering only the vertices which are adjacent to an already assigned host vertex. Therefore, since the host graph is a cycle, there are two possible host vertices at each step, except for the first iteration (all host vertices are available) and for the last iteration (there will be only one remaining host vertex available). For the rest of the iterations, we try the embedding in both possible host vertices and select the one which produces fewer error, with ties broken at random. Next, we review the three constructive procedures proposed, which compose the MAB hyperheuristic.

Constructive 1: Cliques generation. This approach is based on the methodology initially proposed in Pardo et al. (2020). Particularly, it uses the concept of cliques (i.e., a subset of vertices within a graph where every pair of vertices is adjacent) as candidate elements for the constructive procedure in Algorithm 2. This updated method tries to generate as many cliques as possible from the input graph, within a specified time limit, considering only the vertices connected by positive edges. To that end, we use the Bron–Kerbosch algorithm (Conte,

2013). Regarding the time limit, it is worth mentioning that, for large instances constructing all possible cliques is not reasonable due to time constraints. Also, the main memory of the computer would run out due to the large number of cliques, thus stopping the execution. To solve these problems, we apply a time limit to the Bron–Kerbosch algorithm. In this case, since all vertices in a clique are connected by positive edges, they can be embedded in consecutive adjacent host vertices in the solution, without producing any error among them. Therefore, in this constructive procedure, the candidate elements forming the CL (see step 2 and step 10 in Algorithm 2) are groups of vertices (i.e., cliques) instead of just simple vertices. Notice that, when performing multiple iterations that involve different constructions, it is not necessary to calculate the cliques again, since they can be computed once *a priori* and stored for future iterations.

The greedy criterion g_1, provided to this first constructive procedure as an input parameter, evaluates each candidate clique according to its cardinality, with the objective of adding the cliques to the solution in descending order of size. Formally, let q be a clique, and let $g_1(q) = -|q|$. Notably, if a vertex is part of multiple cliques, when a clique containing it is added to the solution, the vertex has to be removed from the other cliques in the CL. This implies the update of the CL (see step 10 in Algorithm 2) after each insertion. Additionally, it is important to remark that the vertices within each clique are embedded in the host graph following a random order. Regarding the algorithmic complexity, this method has a worst-case scenario of $O(B_K + C \cdot |V_G|^3)$, where B_K is the complexity of the initial Bron–Kerbosch algorithm execution and C represents the maximum number of cliques of the graph for any input graph. $|V_G|^3$ is the complexity of the steps required for adding each selected clique to the partial solution under construction and evaluate the associated objective function. The initial Bron–Kerbosch algorithm execution complexity is $O(3^{|V_G|/3})$ (Tomita et al., 2006) and the number of cliques is also $3^{|V_G|/3}$ (Moon and Moser, 1965), therefore, the resulting complexity would be $O(3^{|V_G|/3} + 3^{|V_G|/3} \cdot |V_G|^3)$, which can be simplified to just $O(3^{|V_G|/3} \cdot |V_G|^3)$. However, it is important to note that, in practice, we let the Bron–Kerbosch algorithm run for a short period of time and, therefore the number of cliques obtained is relatively small, which makes this method suitable to be used in a reduced time frame.

Subsequently, once a feasible solution has been constructed, a local search procedure is conducted. This search involves swapping entire cliques within the solution while preserving the internal order of the embedded vertices.

Constructive 2: Community detection. This method refines Constructive 1, aiming to reduce the computational overhead linked to the Bron–Kerbosch algorithm for clique generation. Instead of searching for cliques, this approach adopts a community detection method to cluster vertices within the input graph. We employed a tailored variant of the Louvain method (Que et al., 2015), known for its rapid and scalable identification of communities in large networks. In this case, a previous implementation used to generate the communities has been adapted for signed graphs (Esmailian and Jalili, 2015).

This algorithm diverges from the previous constructive method by selecting the community that introduces the fewest errors in the solution when embedding its vertices in the host graph. In formal terms, the evaluation function g_2 assesses a candidate community, designated as o, in relation to a given incomplete solution, represented by the function $g_2(\varphi'_1, o) = \mathcal{E}(\varphi'_2)$ wherein φ'_2 represents the solution after the addition of community o to the solution φ'_1, and the $\mathcal{E}$ function is defined in Eq. (1). As with Constructive 1, the vertices within each selected community are embedded in the host graph in a random order.

Again, to select the host vertex to be assigned in each iteration, we used the same strategy explained in the description of Constructive 1.

The algorithmic complexity for this constructive method is $O(L + C \cdot |V_G|^3)$, where L is the complexity of the Louvain method, used to extract the communities, C represents the number of communities, and $|V_G|^3$ represents the complexity of evaluating the objective function for each community. The complexity of the Louvain method is $O(|V_G| \cdot log(|V_G|))$ based on the efficiency shown on large networks (Blondel et al., 2008), and the number of communities is at most $|V_G|$ non-overlapping sets of vertices. Therefore, the resulting complexity is $O(|V_G| \cdot log(|V_G|) + |V_G| \cdot |V_G|^3)$, which is simplified to $O(|V_G|^4)$. In practice, as the number of communities C increases, the size of each community decreases, reducing the computational time. If we compare the algorithmic complexity of Constructive 1 and Constructive 2, we observe that the complexity of Constructive 2 is smaller than the complexity of Constructive 1. This is in part due to the fact that the number of cliques greatly exceeds the number of communities.

Subsequently, once a feasible solution has been constructed, a local search procedure is conducted. This search involves alternating movements of swap and insert involving entire communities within the solution, while preserving the internal order of the vertices embedded.

Constructive 3: Edge-based vertex selection. This constructive method distinguishes itself from the previous ones in two key aspects: the composition of the CL and the greedy function used to evaluate the candidate elements. Unlike the previous variants, where the CL comprised groups of vertices (cliques or communities) in this variant, in this case the CL consists of individual vertices. Initially, it is composed of all vertices of the input graph ($v \in V_G$). Consequently, the solution is constructed by adding candidates to the solution one by one, and a candidate is a vertex awaiting to be embedded in the host graph.

In this case, the greedy function used to evaluate candidate vertices, denoted as g_3, is based on the idea of quantifying the priority of incorporating a vertex into the solution under construction. This concept was initially proposed by McAllister for the Minimum Linear Arrangement problem (Mcallister, 1999) and has since been used to tackle various other GLP variants (Cavero et al., 2022b, 2021). In this work, we have adapted it to consider a signed input graph.

Before defining g_3, let us introduce the following notation: given a vertex $u \in$ CL, we denote as $A^+(u)$ and $A^-(u)$ to the sets of vertices containing those adjacent vertices to u, already assigned to the solution, which are connected by a positive or a negative edge, respectively. Similarly, $U^+(u)$ and $U^-(u)$ contain those adjacent vertices to u awaiting to be assigned to the solution, connected by a positive or negative edge, respectively. Formally,

- $A^+(u) = \{v \in V_G : (u,v) \in E_G^+ \wedge v \notin \text{CL}\}$,
- $U^+(u) = \{v \in V_G : (u,v) \in E_G^+ \wedge v \in \text{CL}\}$,
- $A^-(u) = \{v \in V_G : (u,v) \in E_G^- \wedge v \notin \text{CL}\}$,
- $U^-(u) = \{v \in V_G : (u,v) \in E_G^- \wedge v \in \text{CL}\}$.

Then, the function g_3 is defined as a linear combination of the four criteria as follows:

$$g_3(\varphi', u) = -w_1 * |A^+(u)| + w_2 * |U^+(u)| + w_3 * |A^-(u)| - w_4 * |U^-(u)|, \tag{6}$$

where w_1, w_2, w_3, w_4 are search parameters that weight the importance of having more/less positive/negative adjacent vertices in the current solution. The value of each parameter needs to be tuned experimentally within the range $[0, 1]$, such that $w_1 + w_2 + w_3 + w_4 = 1$.

Notice that in this case no specific local search is applied after the construction step since, in contrast to the previous constructive methods, it is based on single vertices instead of group of vertices.

In terms of time complexity, this procedure has a complexity of $O(|V_G|^3)$. The selected vertex is evaluated with the objective function twice, resulting in a complexity of $O(|V_G|^2)$, and all vertices must be selected once, which has a complexity of $O(|V_G|)$. Comparing the complexity of the three proposed constructive procedures, we observe that the complexity of Constructive 3 is smaller than the other two proposals.

4.3. Improvement phase

This section presents the improvement phase within our MAB algorithm. Particularly, the improvement procedure introduced in this proposal has been inspired by the Variable Neighborhood Descent (VND) methodology (Hansen et al., 2019), which can be easily tailored for the integration within MAB. Next, we review the methodological background of VND and describe our specific proposal.

Methodological background

The improvement phase of our proposal is based on the concept of local search procedure, which starts the search from an initial feasible solution, previously obtained from the construction phase. A local search method explores a particular neighborhood looking for a local optimum, which is the best solution in that neighborhood. During the search, the procedure performs deterministic movements in the current solution which conduct to a neighboring solution. Then, that solution is evaluated and, when an improvement is found, the method chooses between immediate restart of the search process from the new solution found or defer the start of a new local search upon finding the best solution in the current iteration. This leads the method to the two well-known exploration strategies: *first improvement* and *best improvement*, respectively.

One of the most popular and effective metaheuristics used to combine multiple local search procedures is VND (Hansen et al., 2019; Mladenović and Hansen, 1997). This methodology systematically changes the neighborhood structure within a local search algorithm. The basic idea is to explore the increasingly distant neighborhoods of the current solution, in the hope of finding an improving solution. The VND algorithm alternates between different neighborhood structures, allowing the method to escape from local optima. The procedure starts from a feasible solution and a set of predefined neighborhood structures. Iteratively, VND selects a solution in the current neighborhood. If an improving solution is found, the algorithm updates the current solution and starts the search again with the first neighborhood. Otherwise, it moves to the next neighborhood in the sequence. The process continues until no further improvement can be found in any of the predefined neighborhoods.

In Algorithm 3, we detail the proposed VND, which takes an initial solution φ and multiple neighborhoods as inputs. The neighborhoods are denoted as N_k, where in this case, $k = \{1, 2\}$. At the start of the execution, the best current solution, φ^*, is set to the initial input solution φ (step 1) and the current neighborhood k is initialized by setting $k = 1$ (step 2). Then, an iterative process is started, in which all the provided neighborhoods are explored with the objective of enhancing the quality of the solution (steps 3 to 11). Within this process, in the first iteration, the algorithm examines the neighborhood N_1 for the solution φ^* by running a local search method (step 4). The local search method returns a local optimum denoted as φ'. Then, if the new local optimum is better than the best solution found, the latter is updated, starting the subsequent explorations from this solution, and the neighborhood k is restarted to the first one (steps 5 to 7). However, when no improvement is found, the procedure moves to the next neighborhood to be explored by increasing the value of k (step 9). The algorithm continues exploring neighborhoods across those provided in N_k, iterating through them, and updating φ^* whenever a better solution is discovered in any neighborhood. This iterative process persists until no further enhancements can be made in any neighborhood, indicating the termination of the algorithm (step 11). The best solution found throughout this process, φ^*, is a local optimum with respect to all the neighborhood structures explored. Then, this solution is returned by the method (step 12).

Algorithm 3 Improvement Phase: VND

Input: Starting solution φ, multiple neighborhoods N_k
1: $\varphi^* = \varphi$ {Assign the best solution}
2: $k = 1$
3: **while** $k \leq |N_k|$ **do**
4: $\varphi' = \texttt{LocalSearch}(N_k, \varphi^*)$
5: **if** $\mathcal{E}(\varphi') < \mathcal{E}(\varphi^*)$ **then**
6: $\varphi^* = \varphi'$ {Update the best solution}
7: $k = 1$
8: **else**
9: $k = k + 1$
10: **end if**
11: **end while**
12: **return** φ^*

Algorithmic proposal

In this paper, we propose a VND procedure which explores two neighborhoods: the swap neighborhood, denoted as N_S, and the insert neighborhood, denoted as N_I.

Notice that, the decisions of which neighborhood explore first (N_S or N_I) and the selection of the improvement strategy (*first* or *best improvement*) has been empirically determined (see Section 5.2). Particularly, the method is configured to explore N_S first and N_I later, both using a *first improvement* strategy.

The **swap neighborhood**, denoted by $N_S(\varphi)$, comprises the set of solutions achievable from φ through the use of the swap move. The size of the swap neighborhood is $\frac{n\cdot(n-1)}{2}$, being n the number of vertices of the input graph.

The swap move involves exchanging the assignments of two input vertices in the solution. Formally, given $u, v \in V_G$ and $x, y \in V_H$ such that $\varphi(u) = x$, and $\varphi(v) = y$, the embedding after the swap operation results in $\varphi(u) = y$, and $\varphi(v) = x$.

Let us illustrate the swap move with an example. Taking into account the embedding φ'_1 from Fig. 5, where $\varphi'_1(\text{A}) = 1$, $\varphi'_1(\text{B}) = 2$, $\varphi'_1(\text{C}) = 3$, $\varphi'_1(\text{D}) = 4$ and $\varphi'_1(\text{E}) = 5$, a swap between vertices A and C will update the assignations $\varphi'_2(\text{A}) = 3$ and $\varphi'_2(\text{C}) = 1$. Notice that only the values of $\varphi'_1(\text{A})$ and $\varphi'_1(\text{C})$ have been modified in φ'_2, remaining the rest of assignments unchanged.

The **insert neighborhood**, denoted as $N_I(\varphi)$, encompasses all the solutions achievable from the solution φ by applying an insert move. The size of the insert neighborhood is $n \cdot (n-1)$, being n the number of vertices of the input graph.

The insert move involves reassigning an input vertex embedded in the solution φ into a different host vertex. This might imply a chain of reassignment moves of some of the input vertices to make room for the new assignation. Particularly, this can be performed following a clockwise or counterclockwise criterion, depending on the number of reassignments needed with each criterion (i.e., the fewer, the better, with ties broken at random).

Let us illustrate the insert move with an example. Considering the embedding φ'_1 from Fig. 6, with $\varphi'_1(\text{A}) = 1$, $\varphi'_1(\text{B}) = 2$, $\varphi'_1(\text{C}) = 3$, $\varphi'_1(\text{D}) = 4$, and $\varphi'_1(\text{E}) = 5$, inserting vertex A at host vertex 3 involves a chain of swap moves starting with a swap between A and B, and followed by a swap between A and C. Therefore, the obtained solution φ'_3 after the move, would update only the assignations $\varphi'_3(\text{A}) = 3$, $\varphi'_3(\text{B}) = 1$, and $\varphi'_3(\text{C}) = 2$, remaining the rest of the assignations unchanged.

Notice, that the swap and the insert moves have been defined using vertices, but they can also be used over groups of vertices (i.e., cliques or communities). In fact, this adaptation has been used after the constructive methods 1 and 2, as a post-processing improvement phase.

To complement the previous procedure we propose two additional strategies: a reduction of the neighborhood traversed by the local

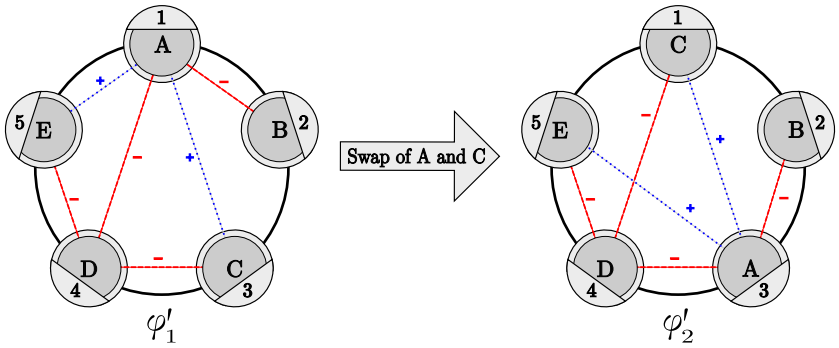


Fig. 5. Example of an embedding φ'_1 and the resultant embedding φ'_2 obtained after swap of vertices A and C.

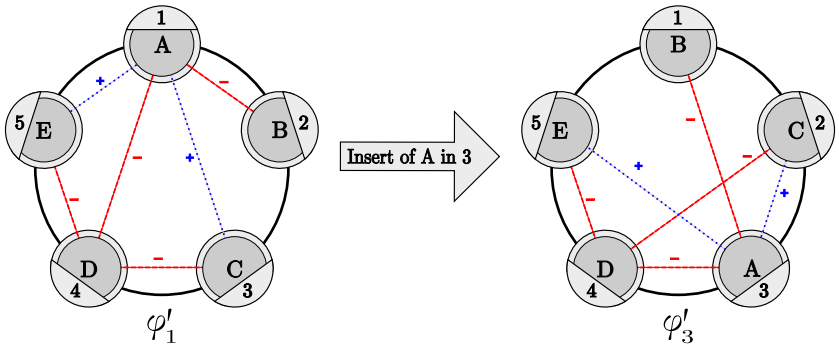


Fig. 6. Example of an embedding φ'_1 and the resultant embedding φ'_3 obtained after insert of vertex A in 3.

search method, exploring only the most promising moves, and a factorization for the evaluation of the objective function, which avoids making unnecessary calculations.

The improving phase is usually the most time-consuming optimization procedure. In order to perform an efficient exploration of the search space, heuristic algorithms can be designed so that the size of a particular neighborhood is affordable in terms of the time needed to explore it. Inspired by the strategy followed by the Constructive 1, those input vertices connected by positive edges are usually close to each other in the embedding, in high-quality solutions. Therefore, for any input vertex u, we can reduce the neighborhood of either swap or insert operations, by considering only its positively connected adjacent vertices. Particularly, for each input vertex v, such that $(u, v) \in E_G^+$, two host vertices (i.e., those adjacent to the host vertex where v is embedded) are candidates for the improvement operation. Therefore, the use of the reduced neighborhood is based on evaluating only a few promising movements which results in a faster convergence to a local minimum. The main inconvenience is that by evaluating only the promising movements we skip movements that may not look promising at the current step of the heuristic procedure but might result in better solutions in latter iterations. This inconvenience is outweighed by the great reduction of the neighborhood and the improvement in the convergence speed.

The second proposed strategy is an efficient way to evaluate the objective function of a solution after a move. This is, instead of evaluating a solution from scratch, an update of the value of the objective function is performed. To do so, we exclusively evaluate the vertices affected by the move, encompassing the moved vertex and its adjacent ones, while the error of the remaining vertices remains unaltered. Additionally, we avoid redundant evaluations by excluding vertices solely possessing positive or negative edges, as they cannot contribute to the total number of errors of the solution. Notice that, this is valid for either the swap or the insert moves. For instance, in the embedding illustrated in Fig. 3, the number of errors of C does not change (i.e., it is zero) after any move since it is only connected by a positive edge. Similarly, this situation also happens in the case of vertices B and D, which are only connected by negative edges. In the case of vertex E, if it is embedded into another vertex, then its number of errors should be updated, together with its adjacent ones (D and A). Since D does not change, then only errors from E and A need to be recomputed. In order to efficiently update the value of the objective function after a move, we store the contribution to the objective function of each vertex separately. Therefore, only the contribution of the vertices affected by the move is updated. It is worth mentioning that from a theoretical perspective, the complexity of exploring a neighborhood based on either the swap or the insert movements remains unchanged whether the evaluation of the objective function is fully performed or just updated with the advanced strategy. However, in practice, due the sparsity of many instances, performing the advanced evaluation results in a significant reduction in computational time for both swap and insert moves with respect to perform a fully evaluation, as we illustrate in the experimental Section 5.2.

5. Experimental results

In this section, we present our experimental results, evaluating the performance of our algorithmic proposals and comparing it to the state-of-the-art methods. It is important to note that all algorithms, including those in the state of the art, have been implemented in Java 17 and all the experiments have been run in an Intel Xeon Gold 6226R 2.90 GHz with 119 GB RAM.

Particularly, in Section 5.1, we outline the instances used in the experiments. In Section 5.2, we detail the preliminary experiments devoted to tune the parameters of our algorithms and determining our best algorithmic configuration. Finally, in Section 5.3 we present the comparisons between our best proposal and the best previous state-of-the-art method.

Table 1
Characteristics of the sets of instances.

Set name	#instances	#vertices	#edges
Complete	108	[10, 230]	[45, 26 335]
Interval	117	[10, 250]	[6, 26 281]
Random	117	[10, 250]	[9, 24 900]
Real cases	14	[16, 5000]	[58, 29 630]

5.1. Instances

In this study, we used three instance sets derived from the literature and a new instance set derived from real scenarios, making a total of 356 instances. The instance sets from the literature consist of three different types of graphs (complete graphs, interval graphs, and random graphs) and they have been drawn from previous research on MinSA and CMinSA (Pardo et al., 2020). These groups of instances complement each other by exhibiting diverse graph structures alongside varying edge density and proportions of negative edges, totaling 342 instances. In addition, we propose the inclusion of a new set of instances derived from real-world instances in social network graphs and related fields, totaling 14 new instances. In Table 1, we summarize the main characteristics of each group of instances as well as the number of instances per group.

From the previous instances available, we have selected a subset of 81 representative instances for preliminary experiments, using the selection method proposed in Martín-Santamaría et al. (2024).

All the instances, including those from the preliminary set have been made available at GitHub.[1]

5.2. Preliminary experiments

This section presents the preliminary tests conducted to fine-tune the parameters of our algorithms, but also to determine the best variant among our proposals, and to illustrate the performance of some of the advanced strategies.

Parameter setting

We start our preliminary experimentation by adjusting the parameters of the constructive procedures proposed in Section 4.2. Particularly:

- The alpha parameter (α), which is common to the three constructive procedures, determines the balance between greediness and randomness of each constructive method.
- The time limit parameter of the Bron–Kerbosch algorithm, which generates the cliques used in Constructive 1.
- The resolution parameter, relevant to Constructive 2, which regulates the size of the clusters such that the larger the value, the larger the size of the resulting clusters.
- Finally, the parameters w_1, w_2, w_3, and w_4, which are relevant weights for Constructive 3, determine the percentages of positive or negative adjacent vertices that are already in the solution or waiting to be embedded.

All these parameters have been tailored using the optimization capabilities of *irace*, except for the time limit of the Bron–Kerbosch algorithm. This parameter has been empirically set to 2 seconds to find a balance between solution quality, CPU time, and RAM consumption.

Specifically, in the case of the Constructive 1 (Cliques), the parameter α was empirically set by *irace* to 0.30. In the Constructive 2 (Community-Based), *irace* determined a value of 0.05 for α, indicating a predominantly greedy approach with minimal randomness, accompanied by a resolution of 0.8. Finally, in the case of Constructive 3 (Edge-Based Vertex Selection), *irace* determined the best value for α to 0.4 and $w_1 = 0$, $w_2 = 0$, $w_3 = 0.5$, and $w_4 = 0.5$. Therefore, the resulting greedy function for this variant is defined as: $g_3(\varphi', u) = |A^-(u)| - |U^-(u)|$. This new expression avoids the vertices with negatives adjacent vertices already assigned in the solution, and instead prioritizes those which have a big amount of negative adjacent vertices that have not been assigned yet. The final adjusted parameter values are summarized in Table 2.

Table 2
Final value adjusted for the parameters of each constructive.

Constructive	Value for each parameter
Constructive 1	α: 0.3 Time limit for cliques search: 2 s
Constructive 2	α: 0.05 Resolution: 0.8
Constructive 3	α: 0.4 w_1: 0.0 w_2: 0.0 w_3: 0.5 w_4: 0.5

Subsequently, we employed *irace* to determine the configuration of the improvement phase. This phase is performed using VND and therefore the order for the exploration of the neighborhoods must be determined together with the exploration strategy (*first* or *best improvement*). Specifically, a random solution was provided to *irace* to start the search. It concluded that the best order entailed to explore the swap neighborhood first, and then the insert neighborhood. Moreover, both neighborhoods should be traversed utilizing a first-improving strategy.

Comparison of the performance of MAB arms

With both the constructive and the improvement phases already configured, we can now combine them into a complete arm under the MAB methodology, which first generates a greedy-randomized solution with a constructive procedure based on GRASP and then improves it with a VND method. First, we evaluate the individual performance of each configuration for different sets of instances. To that aim we run each configuration independently for 100 iterations with a time limit fixed to 300 seconds. Furthermore, we have also included an additional row, to observe the performance of the MAB. The results obtained are reported in Table 3, segmented by group of instances (Complete, Interval, Random, and all the instances in the preliminary set). In this table we report three metrics: the average objective function value (Avg.); the average total execution runtime in seconds required by each method (CPUt (s)); and the number of best solutions of the experiment (#Best). Notice that the maximum CPUt(s) per instance is bounded. However, if the search process finds a solution with zero errors it halts the execution since the solution found is already optimal.

In view of the results reported in Table 3, we can identify that no constructive method clearly stands out as the best for all groups of instances, which motivates the use of a combination of them. When analyzing and comparing each arm with respect to the MAB for all preliminary instances, we observe that MAB-based approach achieves the best average objective function value (71 825.52) while requiring slightly more computational time. This is mainly due to the fact that in the first iteration it needs to run all arms once. Notice that the MAB reaches the best solution in 55 out of 81 instances (68% of cases), demonstrating superior performance in solution quality.

It is worth noting that although the MAB demonstrates superior performance compared to the individual methods, this behavior can be partially attributed to the non-deterministic nature of the constructive procedures. From a theoretical perspective, given sufficient computational time and considering average behavior, the MAB should consistently achieve solutions at least as good as those obtained by each of its arms when executed independently. This is because the

[1] https://github.com/GRAFO-URJC/MAB-CMinSA/tree/main/instances.

Table 3
Results obtained by different configurations of GRASP on the preliminary instances.

Instance set	Configuration	Avg.	CPUt (s)	#Best
Complete (27)	Constructive 1 + VND	166 940.96	208.73	17
	Constructive 2 + VND	167 550.74	261.03	4
	Constructive 3 + VND	167 202.33	216.20	11
	MAB	166 858.56	309.44	16
Interval (27)	Constructive 1 + VND	590.07	112.52	23
	Constructive 2 + VND	118.04	99.44	24
	Constructive 3 + VND	468.15	135.78	19
	MAB	237.26	101.15	22
Random (27)	Constructive 1 + VND	48 348.48	208.75	10
	Constructive 2 + VND	48 676.81	273.40	4
	Constructive 3 + VND	48 459.85	210.54	8
	MAB	48 380.74	282.96	17
All instances (81)	Constructive 1 + VND	71 959.84	176.67	50
	Constructive 2 + VND	72 115.20	211.29	32
	Constructive 3 + VND	72 043.44	187.51	38
	MAB	71 825.52	231.18	55

learning mechanism of MAB should ideally identify and exploit the most effective method for each instance type while maintaining the ability to explore alternative strategies.

To further validate our proposal, we conducted an analysis of the contribution of each arm and the associated coordination mechanism within MAB algorithm during the search process. The experiment consisted of executing the MAB for a maximum of 25 iterations while recording three key elements: the selected arm, the quality of the solution obtained at each iteration, and the quality of the best solution found during the process.

Fig. 7 illustrates the results obtained during the search process for three different and representative instances, one per set of instances. In particular, the solid green line depicts the MAB convergence trajectory, representing the best solution found up to that iteration. The blue dashed line shows the actual solution quality obtained at each specific iteration. To distinguish between arms, we employed different markers: circles for arm 1, squares for arm 2, and triangles for arm 3.

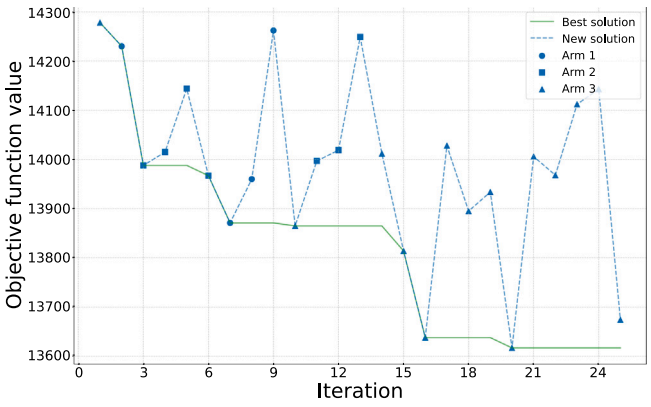
In Fig. 7(a), the experimental results reveal an interesting behavioral pattern of the MAB algorithm. During the initial phase (iterations 1 to 3), the algorithm exhibits a clear exploration strategy by testing different arms. Arm 2 (square) demonstrates superior performance early on, leading to its selection in the subsequent three iterations. The algorithm then diversifies the search by activating arm 1, which discovers an improved solution and consequently remains active for the following two iterations. This alternating pattern between exploration (switching arms) and exploitation (maintaining the same arm) persists until the algorithm ultimately converges on arm 3, which proves to be the most effective choice for obtaining a high-quality solution.

A similar coordination pattern is observed in Fig. 7(b), which presents results for another representative instance from our preliminary test set.

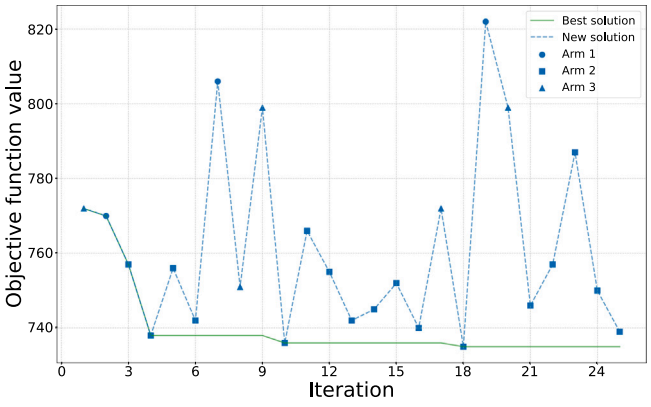
Finally, Fig. 7(c) illustrates a different behavioral pattern, highlighting the adaptability of MAB to vary depending on the characteristics of the instance. In this case, the algorithm quickly identifies arm 3 as ineffective for solving the given instance and excludes it from further consideration in the following iterations. The resulting search process alternates between arms 1 and 2, as both demonstrate comparable effectiveness without showing either a clear dominance.

Contribution of the advanced strategies

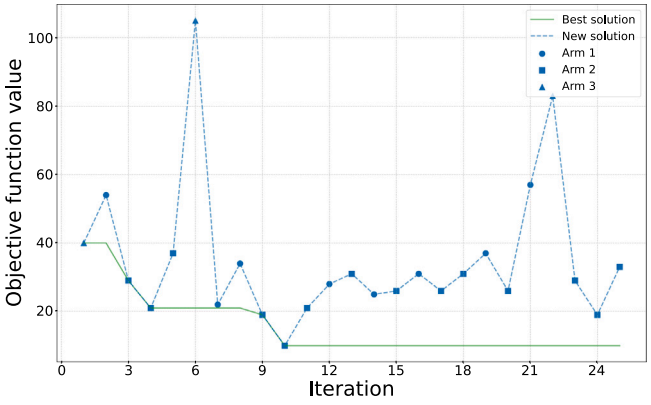
Regarding the techniques proposed to improve the efficiency of the search, it has been observed that avoiding re-evaluating the objective function from scratch reduces the execution time by 59%. Additionally, the use of a reduced neighborhood, achieves an extra reduction of 56% of the execution time. Globally, the combination of both techniques,



(a) Instance name: "C_035_70x2415_100_50" (Complete set).



(b) Instance name: "R_023_50x612_50_50" (Random set).



(c) Instance name: "I_048_110x1212_20_80" (Interval set).

Fig. 7. Behavior of MAB algorithm for three representative instances over 25 iterations.

totals a 76% of time reduction, compared to the original approach. However, the use of the reduced neighborhood has the inconvenience that it leads to solutions that are, on average, 0.2% worse than those obtained when searching in the complete neighborhood. In this case, considering the significant time saved, the trade-off between time and quality is highly favorable, making the reduced neighborhood an essential part of this proposal.

5.3. Comparison with the state of the art

In this section, we compare our MAB proposal with the previous state-of-the-art method for the CMinSA problem. Particularly, we adapted the BVNS method proposed in Pardo et al. (2020) for the

Table 4
Comparison of the state-of-the-art method with the MAB proposal using the complete set of instances.

Instance set	Method	Avg.	CPUt (s)	#Best
Complete (108)	BVNS	126 469.44	296.90	54
	MAB	**125 262.46**	314.09	**73**
Interval (117)	BVNS	975.97	174.44	68
	MAB	**334.15**	121.77	**110**
Random (117)	BVNS	47 439.16	271.47	52
	MAB	**46 615.20**	302.55	**83**
All (342)	BVNS	56 500.79	246.31	174
	MAB	**55 618.19**	244.35	**266**

linear variant of the CMinSA to the problem under study. To test both methods, we used the whole set of 342 instances proposed in the related literature. For this comparison, both methods have been bounded to a maximum running time of 300 seconds, starting new iterations until that time is reached or a solution with O.F. of zero is found. In order to determine whether our proposal was competitive or not, we compared the average objective function value (Avg.), the average CPU time (CPUt (s)), and the number of best solutions found by each method (#Best). Results are presented in Table 4. Since the provided results in these tables are reported on average, we refer the reader to Appendix, where it is possible to find the individual results per instance.

It can be observed that for all instance sets, our MAB method obtains better results than the BVNS method for every metric considered. In particular, our method achieves a significant improvement in the Interval group, where it finds the best solution for 110 instances, while the BVNS method only finds 68 out of 117. The significance of the differences between both methods has been confirmed by the Wilcoxon statistical test, which yields positive results for each instance set, $p = 0.00044$ for Complete, $p = 0.00006$ for Interval, and $p < 0.00001$ for Random, as well as $p < 0.00001$ for the whole dataset. These results show that our MAB method is a competitive and efficient algorithm for the CMinSA problem, outperforming the state-of-the-art method in terms of solution quality when run in a similar computational time.

In our final experiment, we test the BVNS and the MAB algorithms using the new set of instances, not previously reported in the MinSA literature, which have been derived from real-world scenarios. The rationale behind using these instances is two-fold. Firstly, they provide a more accurate representation of real-world problems, thereby serving as a more relevant benchmark for our algorithms. Secondly, since these instances are new and distinct from the subset used for tuning the algorithms, they offer a completely trustworthy and unbiased test bed. The unique properties and structure of these instances ensure that the performance of the algorithms is evaluated in a diverse range of scenarios, further solidifying the validity of our results.

The results obtained are presented in Table 5. In this table, the deviation (Dev. (%)) has also been included because there are no instances with zero errors, which would not make it possible to calculate the deviation. Again, the proposed MAB method obtained overall better results on all metrics. The most important difference can be found in the CPUt (s). The constructive method of the BVNS requires to make a number of evaluations in each step which grows exponentially as the construction progresses. This makes the use of this method unreliable for large instances, such as those used in this comparison. The state-of-the-art instances have a maximum of 250 vertices, while the real-world scenarios instances range from 16 vertices to 5000. As a consequence, our MAB proposal reduces the run time required by the state-of-the-art BVNS method by an order of 225 times. This significant execution time difference can be attributed to two key factors. First, the constructive method of the BVNS algorithm has inherently high time complexity. Second, the original BVNS implementation lacks the efficiency optimizations introduced in our work.

Table 5
Comparison of the state-of-the-art method (BVNS) with the MAB proposal, using a set of 14 real-world instances.

Method	Avg.	CPUt (s)	Dev. (%)	#Best
BVNS	3319.00	69 363.89	75.33	6
MAB	**2835.64**	**307.50**	**8.72**	**11**

6. Conclusions

In this paper, we address the CMinSA problem which aims to find an optimal cyclic arrangement of a given signed graph that minimizes the number of errors. To approach this problem, we propose several novel heuristic algorithms combined into a MAB procedure. Specifically, we combine three constructive methods that are based on different strategies to select vertices from the input graph and assign them to vertices in the cycle host graph, such as generating cliques, detecting communities, and exploiting edge information. Then, we apply a VND method to locally optimize the generated solution within a GRASP framework.

By comparing our constructive methods, each of them provides better or worse results depending on the properties of the specific instance. Even after applying the improvement phase, the results are still different for each constructive. This means that each constructive method generates a different solution depending on the internal structure of the input graph, making it impossible to select only one constructive method as the best one. Instead, we take advantage of this situation by applying a higher-level method, known as hyperheuristic. Specifically, we employ a reinforced learning algorithm, which chooses the best method in a dynamic way for each instance at each iteration.

With respect to the MAB algorithm, it is a great approach in situations in which there are multiple methods with diverse performance for different instances. Specifically, the strength of the MAB lies in those cases in which we are unable to make a clear distinction on which method to select for the final configuration of an algorithm. This case is common in the context of the GLPs, since the structure of the input graph greatly affects the results of the algorithms. This could also be extended to situations where there are some instances that require special methods to solve, but loosing solution quality is not an option.

When configuring the MAB algorithm, we conducted an extensive experimental evaluation on three classes of previously studied instances: complete, interval, and random graphs. We compared our algorithm with the state-of-the-art method, showing that the proposed MAB procedure outperforms the previous best method in terms of solution quality. We also performed a statistical analysis to confirm the significance of our results and to identify the strengths and weaknesses of our variant. In view of the results obtained, we can conclude that our algorithm is a relevant contribution to the state of the art in CMinSA.

It has also been observed that the main negative point of the state-of-the-art BVNS algorithm is the inability to scale to larger instances. In real scenarios, the time needed by BVNS might be 225 times larger than that required by the MAB algorithm. This lack of efficiency renders the BVNS inviable for instances with a large number of vertices. Instead, our proposal employs a lighter constructive method and a neighborhood reduction strategy, making the algorithm significantly faster and allowing the computation of larger instances.

The last contribution of this work is a new set of signed graphs adapted from graphs derived from real cases, which can be used not only for future comparisons with other algorithms for this problem, but also to study other related problems that require a signed graph. This new group of instances solves the lack of variety found in the three groups of instances used in the MinSA literature. Particularly, we included much larger instances, which represent real-world cases, such as social media, relations between people, ranging from small groups to big communities, etc.

Table A.1

Real instance group		
Instance name	Alternative name	O. F.
Real_1_1007x4658_1_8	1000soc-sign-epinions.txt	101
Real_2_1049x6773_2_12	1000wikipedia_adminship_election_data.txt	2 956
Real_3_102x176_4_7	100out.soc-sign-bitcoinalpha.txt	0
Real_4_103x785_15_8	100soc-sign-epinions.txt	48
Real_5_119x549_8_15	100wikipedia_adminship_election_data.txt	41
Real_6_2501x4179_1_4	2500out.soc-sign-bitcoinalpha.txt	549
Real_7_2516x20986_1_8	2500soc-sign-epinions.txt	2 550
Real_8_5000x19525_1_10	5000wikipedia_adminship_election_data.txt	32 045
Real_9_500x1544_2_7	500out.soc-sign-bitcoinalpha.txt	130
Real_10_570x3749_3_6	500soc-sign-epinions.txt	93
Real_11_510x2960_3_15	500wikipedia_adminship_election_data.txt	917
Real_12_219x521_3_10	out.convote.txt	1
Real_13_18x126_83_49	out.moreno_sampson_sampson	34
Real_14_16x58_49_45	out.ucidata-gama	0

Table A.2

Complete instance group					
Instance name	O. F.	Instance name	O. F.	Instance name	O. F.
C-001_10x45_100_20	0	C-037_90x4005_100_20	17 610	C-073_170x14365_100_20	142 358
C-002_10x45_100_50	10	C-038_90x4005_100_50	31 184	C-074_170x14365_100_50	240 582
C-003_10x45_100_80	10	C-039_90x4005_100_80	18 746	C-075_170x14365_100_80	139 396
C-004_10x45_100_20	8	C-040_90x4005_100_20	18 292	C-076_170x14365_100_20	142 611
C-005_10x45_100_50	12	C-041_90x4005_100_50	31 012	C-077_170x14365_100_50	238 015
C-006_10x45_100_80	4	C-042_90x4005_100_80	17 390	C-078_170x14365_100_80	141 983
C-007_10x45_100_20	0	C-043_90x4005_100_20	17 611	C-079_170x14365_100_20	144 386
C-008_10x45_100_50	10	C-044_90x4005_100_50	31 628	C-080_170x14365_100_50	239 973
C-009_10x45_100_80	2	C-045_90x4005_100_80	17 878	C-081_170x14365_100_80	147 488
C-010_30x435_100_20	400	C-046_110x5995_100_20	34 069	C-082_190x17955_100_20	206 462
C-011_30x435_100_50	820	C-047_110x5995_100_50	59 509	C-083_190x17955_100_50	340 303
C-012_30x435_100_80	464	C-048_110x5995_100_80	36 120	C-084_190x17955_100_80	206 304
C-013_30x435_100_20	399	C-049_110x5995_100_20	33 514	C-085_190x17955_100_20	210 335
C-014_30x435_100_50	768	C-050_110x5995_100_50	59 784	C-086_190x17955_100_50	339 742
C-015_30x435_100_80	386	C-051_110x5995_100_80	34 320	C-087_190x17955_100_80	210 514
C-016_30x435_100_20	388	C-052_110x5995_100_20	34 984	C-088_190x17955_100_20	204 426
C-017_30x435_100_50	755	C-053_110x5995_100_50	59 467	C-089_190x17955_100_50	340 435
C-018_30x435_100_80	406	C-054_110x5995_100_80	34 871	C-090_190x17955_100_80	201 066
C-019_50x1225_100_20	2 486	C-055_130x8385_100_20	59 516	C-091_210x21945_100_20	275 590
C-020_50x1225_100_50	4 541	C-056_130x8385_100_50	102 716	C-092_210x21945_100_50	467 692
C-021_50x1225_100_80	2 715	C-057_130x8385_100_80	61 218	C-093_210x21945_100_80	280 909
C-022_50x1225_100_20	2 311	C-058_130x8385_100_20	59 048	C-094_210x21945_100_20	280 313
C-023_50x1225_100_50	4 406	C-059_130x8385_100_50	101 865	C-095_210x21945_100_50	465 176
C-024_50x1225_100_80	2 408	C-060_130x8385_100_80	60 018	C-096_210x21945_100_80	285 211
C-025_50x1225_100_20	2 446	C-061_130x8385_100_20	60 759	C-097_210x21945_100_20	281 712
C-026_50x1225_100_50	4 580	C-062_130x8385_100_50	102 295	C-098_210x21945_100_50	463 135
C-027_50x1225_100_80	2 526	C-063_130x8385_100_80	59 942	C-099_210x21945_100_80	280 172
C-028_70x2415_100_20	8 202	C-064_150x11175_100_20	98 290	C-100_230x26335_100_20	382 186
C-029_70x2415_100_50	13 753	C-065_150x11175_100_50	160 874	C-101_230x26335_100_50	620 394
C-030_70x2415_100_80	7 628	C-066_150x11175_100_80	95 884	C-102_230x26335_100_80	371 890
C-031_70x2415_100_20	7 195	C-067_150x11175_100_20	93 297	C-103_230x26335_100_20	374 877
C-032_70x2415_100_50	13 574	C-068_150x11175_100_50	158 477	C-104_230x26335_100_50	619 067
C-033_70x2415_100_80	8 011	C-069_150x11175_100_80	96 860	C-105_230x26335_100_80	371 868
C-034_70x2415_100_20	7 914	C-070_150x11175_100_20	95 585	C-106_230x26335_100_20	372 148
C-035_70x2415_100_50	13 541	C-071_150x11175_100_50	162 437	C-107_230x26335_100_50	616 490
C-036_70x2415_100_80	7 627	C-072_150x11175_100_80	97 655	C-108_230x26335_100_80	369 065

CRediT authorship contribution statement

Marcos Robles: Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization. **Sergio Cavero:** Writing – original draft, Supervision, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Eduardo G. Pardo:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Oscar Cordón:** Writing – review & editing, Writing – original draft, Validation, Investigation, Formal analysis, Conceptualization.

Appendix. Best known results per instance

In the tables included in this Appendix (see Tables A.1–A.4), we present the value of the objective function (O. F.) of the best solution obtained for each instance found, by any of the evaluated methods, during our experimentation. These values might be useful as a baseline for future comparisons. The individual results per instance and algorithm, including the CPU time, are available in GitHub.[2]

[2] https://github.com/GRAFO-URJC/MAB-CMinSA/tree/main/results.

Table A.3

Interval instance group					
Instance name	O. F.	Instance name	O. F.	Instance name	O. F.
I-001_10x6_20_20	0	I-040_90x2014_50_20	0	I-079_170x11713_80_20	0
I-002_10x8_20_50	0	I-041_90x2073_50_50	0	I-080_170x12058_80_50	0
I-003_10x9_20_80	0	I-042_90x2106_50_80	0	I-081_170x11714_80_80	0
I-004_10x21_50_20	0	I-043_90x3088_80_20	24	I-082_190x3846_20_20	0
I-005_10x25_50_50	0	I-044_90x3303_80_50	0	I-083_190x3601_20_50	0
I-006_10x26_50_80	0	I-045_90x3226_80_80	0	I-084_190x3617_20_80	202
I-007_10x34_80_20	0	I-046_110x1257_20_20	0	I-085_190x9057_50_20	0
I-008_10x35_80_50	0	I-047_110x1244_20_50	0	I-086_190x9204_50_50	0
I-009_10x33_80_80	0	I-048_110x1212_20_80	2	I-087_190x9112_50_80	0
I-010_30x106_20_20	0	I-049_110x3180_50_20	0	I-088_190x14865_80_20	0
I-011_30x85_20_50	0	I-050_110x3020_50_50	0	I-089_190x14608_80_50	0
I-012_30x83_20_80	0	I-051_110x3016_50_80	0	I-090_190x14527_80_80	0
I-013_30x227_50_20	0	I-052_110x5110_80_20	0	I-091_210x4534_20_20	0
I-014_30x216_50_50	0	I-053_110x4928_80_50	0	I-092_210x4492_20_50	0
I-015_30x216_50_80	0	I-054_110x4844_80_80	0	I-093_210x4415_20_80	321
I-016_30x348_80_20	0	I-055_130x1785_20_20	0	I-094_210x11279_50_20	0
I-017_30x345_80_50	0	I-056_130x1731_20_50	0	I-095_210x11070_50_50	0
I-018_30x348_80_80	0	I-057_130x1727_20_80	0	I-096_210x10995_50_80	1
I-019_50x260_20_20	0	I-058_130x4261_50_20	0	I-097_210x18619_80_20	0
I-020_50x235_20_50	0	I-059_130x4354_50_50	0	I-098_210x17830_80_50	0
I-021_50x248_20_80	0	I-060_130x4240_50_80	0	I-099_210x17734_80_80	0
I-022_50x634_50_20	0	I-061_130x6933_80_20	0	I-100_230x5346_20_20	0
I-023_50x609_50_50	0	I-062_130x6740_80_50	0	I-101_230x5488_20_50	0
I-024_50x609_50_80	0	I-063_130x6733_80_80	0	I-102_230x5304_20_80	158
I-025_50x1014_80_20	148	I-064_150x2274_20_20	0	I-103_230x13610_50_20	0
I-026_50x1025_80_50	0	I-065_150x2359_20_50	0	I-104_230x13435_50_50	0
I-027_50x967_80_80	0	I-066_150x2237_20_80	46	I-105_230x13250_50_80	0
I-028_70x482_20_20	0	I-067_150x5626_50_20	0	I-106_230x22043_80_20	0
I-029_70x472_20_50	0	I-068_150x5719_50_50	0	I-107_230x21418_80_50	0
I-030_70x496_20_80	0	I-069_150x5626_50_80	0	I-108_230x21241_80_80	0
I-031_70x1230_50_20	0	I-070_150x9141_80_20	0	I-109_250x6488_20_20	0
I-032_70x1217_50_50	0	I-071_150x8873_80_50	0	I-110_250x6389_20_50	1053
I-033_70x1220_50_80	0	I-072_150x9103_80_80	0	I-111_250x6228_20_80	354
I-034_70x2008_80_20	19	I-073_170x2995_20_20	0	I-112_250x15967_50_20	0
I-035_70x1979_80_50	0	I-074_170x2889_20_50	0	I-113_250x15815_50_50	0
I-036_70x1938_80_80	0	I-075_170x2923_20_80	8	I-114_250x15729_50_80	5177
I-037_90x827_20_20	0	I-076_170x7396_50_20	0	I-115_250x26281_80_20	0
I-038_90x799_20_50	0	I-077_170x7304_50_50	0	I-116_250x25484_80_50	0
I-039_90x816_20_80	0	I-078_170x7275_50_80	0	I-117_250x25193_80_80	0

Table A.4

Random instance group					
Instance name	O. F.	Instance name	O. F.	Instance name	O. F.
R-001_10x9_20_20	0	R-040_90x2002_50_20	3 397	R-079_170x11492_80_20	87 468
R-002_10x9_20_50	0	R-041_90x2002_50_50	6 551	R-080_170x11492_80_50	148 390
R-003_10x9_20_80	0	R-042_90x2002_50_80	3 339	R-081_170x11492_80_80	89 085
R-004_10x22_50_20	0	R-043_90x3204_80_20	10 878	R-082_190x3591_20_20	4 944
R-005_10x22_50_50	0	R-044_90x3204_80_50	18 951	R-083_190x3591_20_50	9 327
R-006_10x22_50_80	0	R-045_90x3204_80_80	10 620	R-084_190x3591_20_80	4 942
R-007_10x36_80_20	0	R-046_110x1199_20_20	633	R-085_190x8977_50_20	45 024
R-008_10x36_80_50	0	R-047_110x1199_20_50	1 362	R-086_190x8977_50_50	76 881
R-009_10x36_80_80	0	R-048_110x1199_20_80	582	R-087_190x8977_50_80	44 348
R-010_30x87_20_20	0	R-049_110x2997_50_20	6 540	R-088_190x14364_80_20	124 665
R-011_30x87_20_50	0	R-050_110x2997_50_50	12 382	R-089_190x14364_80_50	210 956
R-012_30x87_20_80	0	R-051_110x2997_50_80	7 369	R-090_190x14364_80_80	124 920
R-013_30x217_50_20	56	R-052_110x4796_80_20	21 717	R-091_210x4389_20_20	7 543
R-014_30x217_50_50	119	R-053_110x4796_80_50	36 777	R-092_210x4389_20_50	13 550
R-015_30x217_50_80	77	R-054_110x4796_80_80	20 564	R-093_210x4389_20_80	6 603
R-016_30x348_80_20	253	R-055_130x1677_20_20	1 140	R-094_210x10972_50_20	61 603
R-017_30x348_80_50	419	R-056_130x1677_20_50	2 505	R-095_210x10972_50_50	104 390
R-018_30x348_80_80	223	R-057_130x1677_20_80	1 175	R-096_210x10972_50_80	63 759
R-019_50x245_20_20	4	R-058_130x4192_50_20	12 299	R-097_210x17556_80_20	174 259
R-020_50x245_20_50	41	R-059_130x4192_50_50	22 130	R-098_210x17556_80_50	288 943
R-021_50x245_20_80	9	R-060_130x4192_50_80	12 171	R-099_210x17556_80_80	175 587
R-022_50x612_50_20	358	R-061_130x6708_80_20	37 257	R-100_230x5267_20_20	10 098
R-023_50x612_50_50	728	R-062_130x6708_80_50	63 130	R-101_230x5267_20_50	17 821
R-024_50x612_50_80	357	R-063_130x6708_80_80	35 839	R-102_230x5267_20_80	9 793
R-025_50x980_80_20	1531	R-064_150x2235_20_20	2 094	R-103_230x13167_50_20	84 617
R-026_50x980_80_50	2582	R-065_150x2235_20_50	4 059	R-104_230x13167_50_50	139 474
R-027_50x980_80_80	1476	R-066_150x2235_20_80	2 025	R-105_230x13167_50_80	80 744
R-028_70x483_20_20	39	R-067_150x5587_50_20	19 916	R-106_230x21068_80_20	235 106
R-029_70x483_20_50	206	R-068_150x5587_50_50	35 582	R-107_230x21068_80_50	384 264
R-030_70x483_20_80	66	R-069_150x5587_50_80	19 876	R-108_230x21068_80_80	228 913
R-031_70x1207_50_20	1445	R-070_150x8940_80_20	56 960	R-109_250x6225_20_20	13 153
R-032_70x1207_50_50	2669	R-071_150x8940_80_50	99 538	R-110_250x6225_20_50	23 727
R-033_70x1207_50_80	1479	R-072_150x8940_80_80	58 244	R-111_250x6225_20_80	13 021
R-034_70x1932_80_20	4353	R-073_170x2873_20_20	3 171	R-112_250x15562_50_20	109 127
R-035_70x1932_80_50	8338	R-074_170x2873_20_50	6 370	R-113_250x15562_50_50	184 263
R-036_70x1932_80_80	4533	R-075_170x2873_20_80	2 983	R-114_250x15562_50_80	111 020
R-037_90x801_20_20	202	R-076_170x7182_50_20	29 342	R-115_250x24900_80_20	298 015
R-038_90x801_20_50	565	R-077_170x7182_50_50	53 585	R-116_250x24900_80_50	500 856
R-039_90x801_20_80	147	R-078_170x7182_50_80	29 084	R-117_250x24900_80_80	300 592

The instance names included in here indicate their specific properties and are formatted as T-ID_VxE_C_N where:

- T is the type of instance, abbreviated as C for Complete, I for Interval, R for Random, and Real for real-world instances.
- ID is a numeric identifier of the instance.
- V represents the number of vertices in the instance.
- E denotes the number of edges.
- C indicates the connectivity density, representing the average percentage of other vertices to which each vertex is directly connected.
- N signifies the percentage of edges that are negative.

Data availability

I have shared the data used in a link included in the paper.

References

Almeida, C., Gonçalves, R., Venske, S., Lüders, R., Delgado, M., 2020. Hyper-heuristics using multi-armed bandit models for multi-objective optimization. Appl. Soft Comput. 95, 106520.

Aracena, J., Thraves Caro, C., 2023. The weighted sitting closer to friends than enemies problem in the line. J. Comb. Optim. 45, 9.

Benítez, F., Aracena, J., Caro, C., 2018. The sitting closer to friends than enemies problem in the circumference. ArXiv Preprint arXiv:1811.02699.

Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E., 2008. Fast unfolding of communities in large networks. J. Stat. Mech. Theory Exp. 2008 (10), 10008.

Bouneffouf, D., Rish, I., Aggarwal, C., 2020. Survey on applications of multi-armed and contextual bandits. In: 2020 IEEE Congress on Evolutionary Computation. CEC, pp. 1–8.

Burke, E., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. 64, 1695–1724.

Cavero, S., Pardo, E., Duarte, A., 2022a. A general variable neighborhood search for the cyclic antibandwidth problem. Comput. Optim. Appl. 81, 657–687.

Cavero, S., Pardo, E., Duarte, A., Rodriguez-Tello, E., 2022b. A variable neighborhood search approach for cyclic bandwidth sum problem. Knowledge- Based Syst. 246, 108680.

Cavero, S., Pardo, E., Laguna, M., Duarte, A., 2021. Multistart search for the cyclic cutwidth minimization problem. Comput. Oper. Res. 126, 105116.

Chu, W., Li, L., Reyzin, L., Schapire, R., 2011. Contextual bandits with linear payoff functions. In: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics. pp. 208–214.

Chung, F., 1988. Labelings of Graphs, Selected Topics in Graph Theory 3 III.

Clotfelter, C., Cook, P., 1993. The gambler's fallacy in lottery play. Manag. Sci. 39, 1521–1525.

Conte, A., 2013. Review of the Bron-Kerbosch Algorithm and Variations. School Of Computing Science, University Of Glasgow., pp. 1–9.

Cygan, M., Fomin, F., Golovnev, A., Kulikov, A., Mihajlin, I., Pachocki, J., Socała, A., 2017. Tight lower bounds on graph embedding problems. J. ACM (JACM) 64, 1–22.

Cygan, M., Pilipczuk, M., Pilipczuk, M., Wojtaszczyk, J., 2012. Sitting closer to friends than enemies, revisited. In: International Symposium on Mathematical Foundations of Computer Science. pp. 296–307.

Díaz, J., Petit, J., Serna, M., 2002. A survey of graph layout problems. ACM Comput. Surv. (CSUR). 34, 313–356.

Dokeroglu, T., Kucukyilmaz, T., Talbi, E., 2024. Hyper-heuristics: A survey and taxonomy. Comput. Ind. Eng. 187, 109815.

Drake, J., Kheiri, A., Özcan, E., Burke, E., 2020. Recent advances in selection hyper-heuristics. European J. Oper. Res. 285, 405–428.

Dujmović, V., Wood, D., 2004. On linear layouts of graphs. Discrete Math. Theor. Comput. Sci. 6.

Esmailian, P., Jalili, M., 2015. Community detection in signed networks: the role of negative ties in different scales. Sci. Rep. 5, 14339.
Feo, T., Resende, M., 1995. Greedy randomized adaptive search procedures. J. Glob. Optim. 6, 109–133.
Hansen, P., Mladenović, N., Brimberg, J., Pérez, J., 2019. Variable Neighborhood Search. Springer.
Jacobson, I., Booch, G., Rumbaugh, J., 1999. The Unified Software Development Process. Addison-Wesley.
Kermarrec, A., Thraves, C., 2011. Can everybody sit closer to their friends than their enemies? In: International Symposium On Mathematical Foundations Of Computer Science. pp. 388–399.
Kuleshov, V., Precup, D., 2014. Algorithms for multi-armed bandit problems. ArXiv Preprint arXiv:1402.6028.
Lattimore, T., Szepesvári, C., 2020. Bandit algorithms. Cambridge University Press.
Li, L., Chu, W., Langford, J., Schapire, R., 2010. A contextual-bandit approach to personalized news article recommendation. In: Proceedings Of The 19th International Conference On World Wide Web. pp. 661–670.
Lin, Y., 1995. The cyclic bandwidth problem. Chin. Sci. Abstr. Ser. A. 2 (14), 14.
Lozano, M., Duarte, A., Gortázar, F., Martí, R., 2013. A hybrid metaheuristic for the cyclic antibandwidth problem. Knowledge- Based Syst. 54, 103–113.
Martín-Santamaría, R., Cavero, S., Herrán, A., Duarte, A., Colmenar, J., 2024. A practical methodology for reproducible experimentation: an application to the double-row facility layout problem. Evol. Comput. 32 (1), 69–104.
Mcallister, A., 1999. A New Heuristic Algorithm for the Linear Arrangement Problem. University of New Brunswick, New Brunswick, CA.
Meidani, K., Mirjalili, S., Farimani, A., 2022. MAB-OS: multi-armed bandits metaheuristic optimizer selection. Appl. Soft Comput. 128, 109452.
Mitchison, G., Durbin, R., 1986. Optimal numberings of an n*n array. SIAM J. Algebraic Discret. Methods 7, 571–582.
Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Oper. Res. 24, 1097–1100.
Moon, J.W., Moser, L., 1965. On cliques in graphs. Israel J. Math. 3, 23–28.
Newton, A., 1981. Computer-aided design of VLSI circuits. Proc. IEEE 69, 1189–1199.
Pardo, E., Martí, R., Duarte, A., 2018. Duarte, a linear layout problems. In: Handbook of Heuristics. pp. 1025–1049.
Pardo, E., a Sánchez, A.Garcí., Sevaux, M., Duarte, A., 2020. Basic variable neighborhood search for the minimum sitting arrangement problem. J. Heuristics 26, 249–268.
Pardo, E., Soto, M., Thraves, C., 2015. Embedding signed graphs in the line. J. Comb. Optim. 29, 451–471.
Petit, J., 2003. Experiments on the minimum linear arrangement problem. J. Exp. Algorithm. (JEA) 8.
Que, X., Checconi, F., Petrini, F., Gunnels, J., 2015. Scalable community detection with the louvain algorithm. In: 2015 IEEE International Parallel and Distributed Processing Symposium. pp. 28–37.
Ravi, R., Agrawal, A., Klein, P., 1991. Ordering problems approximated: single-processor scheduling and interval graph completion. In: Automata, Languages and Programming: 18th International Colloquium Madrid, Spain, July (1991) 8–12 Proceedings 18. pp. 751–762.
Resende, M., Ribeiro, C., 2010. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In: Handbook of Metaheuristics. pp. 283–319.
Resende, M., Ribeiro, C., 2016. Optimization By GRASP. Springer.
Robles, M., Cavero, S., Pardo, E., 2022. BVNS for the minimum sitting arrangement problem in a cycle. In: International Conference on Variable Neighborhood Search. pp. 82–96.
Sahhaf, S., Tavernier, W., Colle, D., Pickavet, M., 2015. Network service chaining with efficient network function mapping based on service decompositions. In: Proceedings of the 2015 1st IEEE Conference on Network Softwarization. NetSoft, pp. 1–5.
Shafaei, A., Saeedi, M., Pedram, M., 2013. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In: Proceedings of the 50th Annual Design Automation Conference. pp. 1–6.
Shiloach, Y., 1979. A minimum linear arrangement algorithm for undirected trees. SIAM J. Comput. 8, 15–32.
Slivkins, A., 2019. Introduction to multi-armed bandits. Found. Trends Mach. Learn. 12, 1–286.
Stützle, T., Ruiz, R., 2018. Iterated greedy. In: Handbook of Heuristics. pp. 547–577.
Sutton, R., Barto, A., 2018. Reinforcement Learning: An Introduction. MIT Press.
Tomita, E., Tanaka, A., Takahashi, H., 2006. The worst-case time complexity for generating all maximal cliques and computational experiments. Theoret. Comput. Sci. 363 (1), 28–42.