A double-stage heuristic algorithm for the antibandwidth maximization problem via the linear bottleneck assignment optimization

Jintong Ren[a], Jin-Kao Hao[b], Eduardo Rodriguez-Tello[c], Zequn Wei[d,*]

[a]*Business School, Hohai University, 1 Xikang Road, 210098, Nanjing, China*
[b]*LERIA, Université d'Angers, 2 boulevard Lavoisier, 49045, Angers, France*
[c]*Cinvestav Unidad Tamaulipas, Km. 5.5 Carretera Victoria - Soto La Marina, 87130, Victoria, Mexico*
[d]*School of Economics and Management, Beijing University of Posts and Telecommunications, 10 Xitucheng Road, 100876, Beijing, China*

Computer and Operations Research 195: 107632, 2026

Abstract

The antibandwidth maximization problem is a classic graph layout problem arising in applications such as multiprocessor scheduling, radio frequency assignment, and obnoxious facility location. It involves assigning distinct integer labels to the vertices of a graph such that the minimum absolute difference between the labels of adjacent vertices is maximized. To solve this NP-hard problem, we propose a double-stage heuristic algorithm integrating a local optimization stage with a dedicated reduced neighborhood, and a greedy-optimized perturbation stage including a greedy assignment step as well as an optimized assignment step. Crucially, we demonstrate that the optimized assignment step can be transformed into solving the linear bottleneck assignment problem, which is solvable in polynomial time via a threshold algorithm. Experimental results across 256 benchmark instances show that our algorithm outperforms state-of-the-art methods by achieving 42 improved lower bounds while matching the best-known results on the majority of the remaining instances. Additional analysis confirms the critical contribution of the algorithm's key components to its overall performance.

Keywords: Combinatorial optimization, Heuristics, Antibandwidth maximization problem, Linear bottleneck assignment problem.

1. Introduction

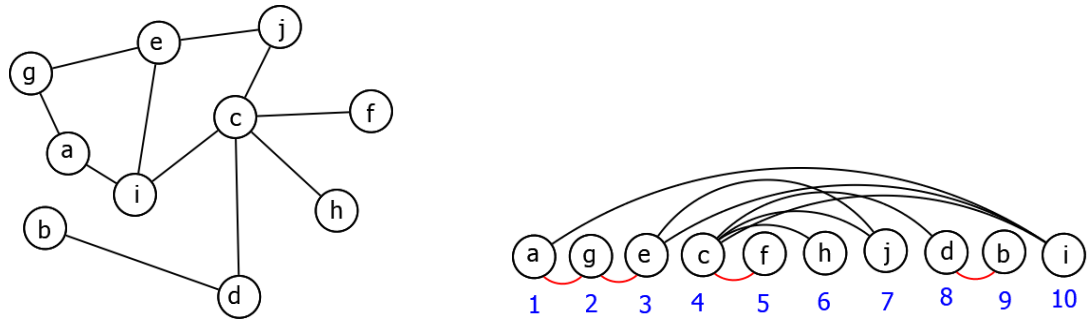
The antibandwidth maximization problem (AMP), proven NP-hard in 1984 (Leung et al., 1984), is a variant of the well-known bandwidth problem (Harper, 1964). Its formal definition is presented as follows. Given an n-vertex undirected graph $G = (V, E)$, the task requires finding a bijective embedding $\varphi : V \rightarrow \{1, 2, \ldots, n\}$ (termed a *solution* or an *arrangement*)

*Corresponding author.
Email addresses: jintong.ren@hhu.edu.cn (Jintong Ren), jin-kao.hao@univ-angers.fr (Jin-Kao Hao), ertello@cinvestav.mx (Eduardo Rodriguez-Tello), zequn.wei@bupt.edu.cn (Zequn Wei)

that places vertices at distinct integer positions. The antibandwidth of φ for G is given by

$$AB(G, E, \varphi) = \min_{\{u,v\} \in E} \{|\varphi(u) - \varphi(v)|\}, \tag{1}$$

where $\varphi(u)$ denotes the label assigned to vertex u. The objective of AMP is to find an embedding φ^* such that the antibandwidth is maximized. Figure 1 illustrates a typical AMP instance. To provide a formal baseline model, clarify the combinatorial structure of AMP, and strengthen the connection with exact methods and optimality bounds, we include in Appendix A an integer programming formulation of the AMP adopted from Duarte et al. (2011).



(a) A graph G of order $n = 10$.

(b) An embedding φ of G.

Figure 1: An illustrative example: (a) A graph G of order $n = 10$ with its vertices named by a to j; (b) An embedding where the graph G is embedded in a line according to the label (from 1 to 10 are marked in blue) assigned to each vertex. The antibandwidth of the embedding φ shown is $AB(G, E, \varphi) = 1$, which is determined by the edges $\{a, g\}$, $\{g, e\}$, $\{c, f\}$, and $\{d, b\}$, which are marked in red.

Recognized as a fundamental graph layout challenge (Díaz et al., 2002; Leung et al., 1984), AMP finds significant applications in multiprocessor scheduling, radio frequency assignment, and obnoxious facility location (Leung et al., 1984; Hale, 1980; Burkard et al., 2001). This broad applicability has motivated substantial theoretical and algorithmic investigations over decades.

Theoretically, researchers have established precise bounds for various graph classes: optimal solutions for paths and cycles were obtained by Lin and Yuan (2003); exact values for complete k-ary trees (with comprehensive results for even k and estimations for odd k) were proven by Calamoneri et al. (2009); two-dimensional meshes and tori were resolved by Raspaud et al. (2009); while hypercubes received explicit solutions from Wang et al. (2009). For higher dimensions, Török and Vrt'o (2009) derived asymptotic bounds up to third-order terms for d-dimensional meshes, with Török and Vrt'o (2010) providing near-optimal results for 3d meshes. Subsequent works established bounds for Hamming graphs (Dobrev et al., 2013) and itchy caterpillars (Rahaman et al., 2014). Bekos et al. (2014) presented tight bounds for

regular caterpillars and spider graphs.

Algorithmically, diverse solution methods have been developed for this NP-hard problem. Duarte et al. (2011) pioneered the first integer programming formulation and proposed GRASP with path relinking heuristics, introducing mesh, Hamming, and Harwell-Boeing benchmark instances for algorithm evaluation. Memetic algorithms were significantly enhanced through innovative label assignment techniques (Bansal and Srivastava, 2010) and later hybridized with intensive local search procedures (Rodriguez-Tello and Betancourt, 2011), demonstrating improved solution quality and convergence speed. After that, variable neighborhood search incorporating ejection chains was proposed by Lozano et al. (2012), which was rigorously validated on 236 diverse instances. Scott and Hu (2014) proposed a level-based heuristic combined with hill-climbing local search and showed effectiveness on several Harwell-Boeing instances. Recent exact methods have achieved significant advances, although solving the largest instances to optimality remains challenging. Sinnl (2021) developed new mixed-integer programming (MIP) formulations for the AMP, combined with branch and cut, iterative MIP, and constraint programming techniques. Using CPLEX 12.9 and CP Optimizer under a 1800-second time limit, the proposed approach proved optimality for eight previously open Harwell-Boeing instances, improved seven best-known values, and reduced the largest reported optimality gap from 577.4% to 46.3%. These results represent a significant advance in exact solution methods for the AMP and provide an important reference for the Harwell-Boeing benchmark set. Similarly, Fazekas et al. (2020) created specialized SAT encodings using binary decision diagrams with dual variable orders, establishing new performance benchmarks through linear-size conjunctive normal form formulations.

Connections to assignment problems provide profound methodological synergies for graph layout optimization. Esposito et al. (1998) established that bandwidth minimization reduces to a linear bottleneck assignment problem during the numbering phase of their enhanced Cuthill-McKee algorithm (Liu and Sherman, 1976), creating a two-phase paradigm (graph partitioning followed by polynomial-time bottleneck assignment) for solving NP-hard layout problems; complementing this, Glover and Rego (2017) revealed fundamental connections between linear assignment and minimum linear arrangement problems, introducing set-based neighborhoods where optimal solutions are findable in polynomial time despite factorial size complexity; conversely, Kellerer and Wirsching (1998) applied bandwidth-derived lower bounds to solve bottleneck quadratic assignment problem, establishing a bidirectional theoretical bridge. Collectively, these works confirm the deep interconnections between graph layout and assignment formulations, suggesting significant untapped potential for AMP in providing exact subproblem solutions within heuristic frameworks through polynomial-time optimization. Collectively, these works confirm the close connections between graph layout problems and assignment formulations. Accordingly, the proposed method is designed as a time-bounded AMP metaheuristic that embeds a polynomial-time assignment procedure for exactly solving the selected

subproblem.

We observe that existing heuristic approaches for AMP could be further enhanced by leveraging theoretical insights. In this work, we bridge AMP with linear bottleneck assignment optimization (Pferschy, 1997) for the first time and propose a double-stage heuristic algorithm (DSHA) to advance the state of the art. Our main contributions are listed as follows.

- By establishing the connection between AMP and linear bottleneck assignment problems, we propose DSHA, a novel double-stage heuristic algorithm. It features a computationally efficient reduced neighborhood for local optimization, and a perturbation stage combining a greedy assignment with an optimized assignment. We show that the optimized assignment can be reformulated as a linear bottleneck assignment problem that is solvable in polynomial time, thereby yielding an optimal solution to the corresponding subproblem.

- Extensive experiments on 256 benchmark instances show that DSHA achieves 42 improved lower bounds, thereby providing tighter reference bounds for subsequent studies. We will make the source code of our algorithm and all benchmark instances publicly available to facilitate future research. We further conduct insightful analysis of the algorithm's key components to elucidate their performance contributions.

The remainder of this paper is structured as follows. Section 2 details the proposed algorithm DSHA, including its core framework and key components. Section 3 presents experimental results comparing DSHA with state-of-the-art methods across benchmark instances. In Section 4, we analyze the impact of DSHA's key components on overall performance. Finally, Section 5 summarizes our findings and suggests future research directions.

2. Double-stage heuristic algorithm

The proposed double-stage heuristic algorithm is based on the iterated local search (ILS) framework (Lourenço et al., 2003), which often includes a local search phase to find local optimal solutions and several perturbation phases to escape local optimum traps. This framework has been successfully applied to solve the bandwidth minimization problem (Abreu and Gonzaga de Oliveira, 2024), two dimensional bandwidth problem (Cavero et al., 2023) and cyclic bandwidth problem (Ren et al., 2019, 2020), demonstrating its effectiveness in graph layout problems. In this work, the term "double-stage" refers to the two complementary stages of the proposed framework, namely local optimization and perturbation. Accordingly, our algorithm adopts a two-stage framework, relying on four components: a greedy construction (GreedyConstruction, See Section 2.1) to generate high-quality initial solutions, an extended evaluation function (See Section 2.2) to distinguish the solutions with the same objective

value, a dedicated tabu search stage with a reduced neighborhood structure (DTSS, See Section 2.3) to perform local optimization and a greedy-optimized perturbation stage (GOPS, See Section 2.4) to diversify the search from the local optimum.

Algorithm 1 Double-stage heuristic algorithm for AMP

1: **Input**: Extended evaluation function f_e, tabu search depth L, sampling percentage of random vertices p and cutoff time T_{max}
2: **Output**: The best found solution φ^*
3: /*GreedyConstruction aims to generate high-quality solutions, See Section 2.1.*/
$\varphi \leftarrow$ GreedyConstruction()
4: $\varphi^* \leftarrow \varphi$
5: **while** the cutoff time limit T_{max} is not reached **do**
6: /*φ_{lb} is the local best found solution.*/
$\varphi_{lb} \leftarrow \varphi$
7: /*DTSS is designed for performing local optimization, See Section 2.3.*/
$(\varphi, \varphi_{lb}) \leftarrow$ DTSS$(\varphi, \varphi_{lb}, f_e, L)$
8: /*GOPS is for escaping from the local optimum, See Section 2.4.*/
$(\varphi, \varphi_{lb}) \leftarrow$ GOPS$(\varphi, \varphi_{lb}, f_e, p)$
9: **if** $f_e(\varphi_{lb}) > f_e(\varphi^*)$ **then**
10: $\varphi^* \leftarrow \varphi_{lb}$
11: **end if**
12: **end while**
13: **return** φ^*

Algorithm 1 details the workflow of DSHA. The process commences with a greedy construction phase initializing solution φ (line 3), which is immediately designated as the best found solution φ^* (line 4). The core optimization loop then iterates until reaching the cutoff time T_{max} (line 5). At each iteration, the current solution φ updates φ_{lb} (line 6), followed by the dedicated tabu search stage (DTSS) executing local optimization (line 7). Subsequently, the greedy-optimized perturbation stage (GOPS) activates to escape local optima (line 8), after which the extended evaluation function updates φ^* based on solution quality (lines 9-11). Upon loop termination, the algorithm returns φ^* as the final solution (line 13).

2.1. Greedy construction

We directly employ the greedy construction procedure from Bansal and Srivastava (2010), a method widely used for AMP and cyclic antibandwidth problem (Bansal and Srivastava, 2011; Lozano et al., 2013; Sundar, 2019; Cavero et al., 2022).

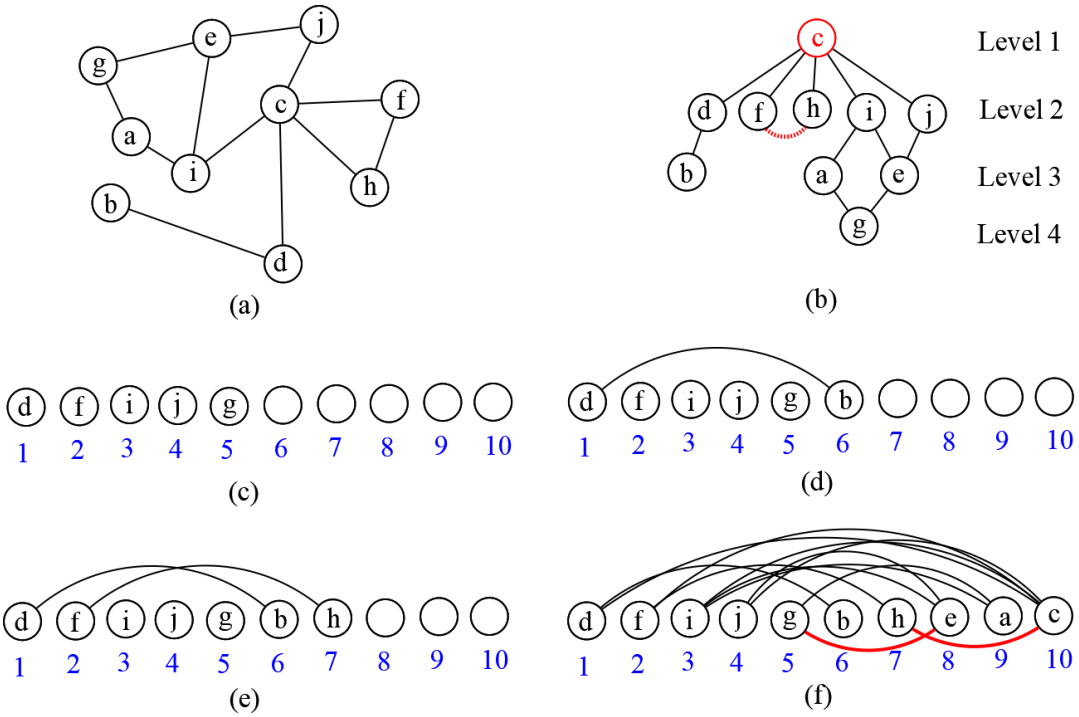


Figure 2: Greedy construction process ($n = 10$): (a) Original graph; (b) BFST rooted at c (red) with intra-level edge (f, h) (dashed); (c) Even-level labeling (labels 1-5), excluding h; (d) Label 6 to b; (e) Label 7 to h; (f) Complete solution.

To make the paper self-contained, we briefly summarize its steps, as implemented in Figure 2. Starting from a randomly chosen root (vertex c in Figure 2(b)), we build a breadth-first search tree (BFST) that partitions the vertices into levels, with adjacent vertices placed in consecutive levels (Díaz et al., 2002). Labels are then assigned to non-adjacent vertices drawn from either the even or the odd levels, skipping any vertex adjacent to one that has already been labelled (vertex h is skipped in Figure 2(c) because of edge (f, h)). Finally, the remaining labels are allocated greedily to limit the increase in the objective: for example, label 6 is given to vertex b (Figure 2(d)) and label 7 to vertex h (Figure 2(e)), yielding the complete solution in Figure 2(f).

2.2. Extended evaluation function

To escape the vast plateaus in the search space, we adopt an extended evaluation function, a technique commonly used in graph layout problems (Ren et al., 2020; Rodriguez-Tello et al., 2015, 2019). For an n-vertex graph, AMP admits $n!/2$ permutations but only n distinct objective values, $1, \ldots, n$; hence large sets of solutions share the same fitness. We therefore refine the objective by adding a tie-breaking term that favors solutions with fewer critical edges

whose length equals the antibandwidth value. Formally, the extended evaluation function is

$$f_e(\varphi) = AB(G, E, \varphi) + \frac{1}{2\,N_c}, \tag{2}$$

where N_c is the number of critical edges, computed as

$$N_c = \sum_{\{u,v\}\in E} X_{u,v}, \quad X_{u,v} = \begin{cases} 1 & \text{if } |\varphi(u) - \varphi(v)| = AB(G, E, \varphi), \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

2.3. Dedicated tabu search stage

To perform local optimization, we employ a dedicated tabu search stage operating in a reduced neighborhood, with pseudocode provided in Algorithm 2. The algorithm begins by initializing the consecutive non-improvement counter Ct to zero (line 3). An outer loop (lines 4-24) executes the bounded local optimization process. Within each iteration, the algorithm first identifies the set of critical vertices S_u (line 5), where critical vertices are defined as endpoints of critical edges in the current solution. After initializing an improvement flag $FlagImp$ to *False* (line 6), the inner refinement loop (lines 7-18) then commences during which a vertex u is randomly selected from S_u (line 8) and removed from the set (line 9). Since critical vertices may lose their status during refinement iterations, only those retaining this attribute are processed. For each qualifying vertex u, the best neighboring solution within the reduced neighborhood is selected (line 11). Following each move, the tabu list receives a systematic update (line 12). When the modified solution φ yields improved cost according to Eq. (2), both φ_{lb} and $FlagImp$ are updated (lines 13-16). This vertex processing continues until S_u is depleted (line 18). After that, Ct undergoes conditional updating (lines 19-23): reset to zero if $FlagImp$ is *True*, otherwise incremented by one. The outer loop persists until Ct reaches the predefined tabu search depth L (line 4). Finally, the algorithm returns both φ and φ_{lb} (line 25). It should be noted that a vertex v initially identified as critical and included in set S_u (line 5 of Algorithm 2) may become non-critical during the iteration. This occurs because the algorithm updates the current solution after exploring the reduced neighborhood of each vertex (lines 7–18). Such an update can alter the status of other vertices in S_u causing them to transition from critical to non-critical. Thus, a reassessment step is incorporated in line 10 to dynamically update the critical status during each iteration.

2.3.1. Reduced neighborhood

The reduced neighborhood employs a *Swap* operator for vertex label exchange, specifically designed to enhance search efficiency by strategically targeting critical vertices (identified via critical edges) and suitable vertices. This approach contrasts with traditional swapping neighborhoods that exhibit high exploration complexity, which often require evaluating an imprac-

Algorithm 2 Dedicated tabu search stage (DTSS)

Input: Input solution φ, local best found solution φ_{lb}, extended evaluation function f_e and tabu search depth L
Output: The current solution φ and the local best found solution φ_{lb}
$Ct \leftarrow 0$
while $Ct < L$ **do**
  $S_u \leftarrow$ IdentifyCriticalSet(φ)
  $FlagImp \leftarrow False$
  repeat
    $u \leftarrow$ Randomly choose a vertex from S_u
    $S_u \leftarrow S_u \setminus \{u\}$
    if u is a critical vertex **then**
      $\varphi \leftarrow$ FindBestNeighbor(ReducedNF(φ, u))
      $UpdateTabuList()$
      if $f_e(\varphi) > f_e(\varphi_{lb})$ **then**
        $\varphi_{lb} \leftarrow \varphi$
        $FlagImp \leftarrow True$
      end if
    end if
  until S_u is empty
  if $FlagImp$ **then**
    $Ct \leftarrow 0$
  else
    $Ct \leftarrow Ct + 1$
  end if
end while
return φ, φ_{lb}

tically large number of candidate moves. By focusing only on the most promising regions of the solution space, this reduction allows for a more intelligent and computationally efficient search process. For each critical vertex u with k adjacent vertices, the solution space is partitioned into $k + 1$ distinct components. Suitable vertices are then determined per component based on their structural properties and the current antibandwidth value $AB(G, E, \varphi)$. The formally defined reduced neighborhood operates through three sequential steps:

1) *Critical vertex identification.* Given the graph $G = (V, E)$ and the current solution φ, the set of critical vertices S_u with respect to φ is determined by:

$$S_u = \{u \in V : ab(u, \varphi) = AB(G, E, \varphi)\} \ , \tag{4}$$

where the vertex-specific antibandwidth $ab(u, \varphi)$ is defined as:

$$ab(u, \varphi) = \min_{v \in A(u)} |\varphi(u) - \varphi(v)| \ , \tag{5}$$

with $A(u)$ denoting the adjacency set of vertex u.

2) *Candidate vertex construction.* For each critical vertex $u \in S_u$, we create the label set $L(u) = \{l_1, \ldots, l_{|A(u)|}\}$ of its adjacent vertices through three sequential substeps:

   i) Sort the labels in $L(u)$ in ascending order to construct the ordered sequence $B(u) = (\beta_1, \ldots, \beta_{|A(u)|})$.

   ii) Generate the consecutive pair set:

$$D(u) = \{(\beta_i, \beta_{i+1}) \mid 1 \leq i \leq |A(u)| - 1\} . \tag{6}$$

   iii) Determine the suitable vertex sets based on $D(u)$, the minimum label β_1, and the maximum label $\beta_{|A(u)|}$:

$$S_1(u, \varphi) = \{v \in V : 1 \leq \varphi(v) \leq \beta_1 - AB(G, E, \varphi)\} , \tag{7}$$

$$S_2(u, \varphi) = \{v \in V : l_1 + AB(G, E, \varphi) \leq \varphi(v) \leq l_2 - AB(G, E, \varphi),\ (l_1, l_2) \in D(u)\} , \tag{8}$$

$$S_3(u, \varphi) = \left\{v \in V : \beta_{|A(u)|} + AB(G, E, \varphi) \leq \varphi(v) \leq n\right\} . \tag{9}$$

   The comprehensive set of suitable vertices for u is then:

$$S_v(u, \varphi) = S_1(u, \varphi) \cup S_2(u, \varphi) \cup S_3(u, \varphi). \tag{10}$$

3) *Restricted neighborhood definition.* For each critical vertex $u \in S_u$, the reduced neighborhood relative to solution φ is defined as:

$$N_1(\varphi, u) = \left\{\varphi' = \varphi \oplus \mathrm{Swap}(u, v) \mid v \in S_v(u, \varphi),\ \mathrm{Swap}(u, v) \notin TL_1\right\} \ , \tag{11}$$

where $\varphi' = \varphi \oplus \mathrm{Swap}(u, v)$ denotes the solution transformation via the swap operator, and TL_1 maintains the tabu status of recently executed swap moves.

Figure 3 illustrates the reduced neighborhood mechanism on a 10-vertex graph. Critical

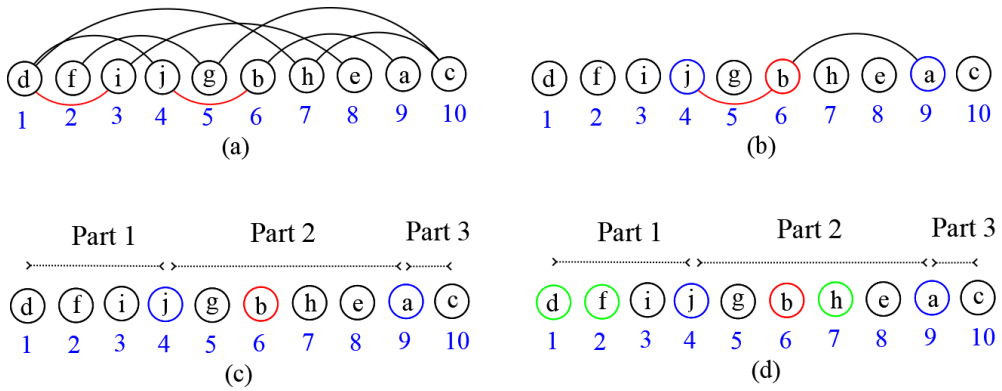


Figure 3: Reduced neighborhood demonstration ($n = 10$): (a) Critical vertices $S_u = \{d, i, j, b\}$ identified via critical edges (red); (b) Processing b with adjacent vertices $A(b) = \{a, j\}$ (blue); (c) Solution partitioning induced by adjacent vertex labels; (d) Candidate vertex set $S_v(b, \varphi) = \{d, f, h\}$ (green).

vertices $S_u = \{d, i, j, b\}$ are identified by critical edges (Figure 3(a)). Processing vertex $b \in S_u$ reveals its adjacent vertices $A(b) = \{a, j\}$ with sorted labels $B(b) = (4, 9)$ (Figure 3(b)). This induces a three-part solution partition (Figure 3(c)), yielding the candidate vertex set $S_v(b, \varphi) = \{d, f, h\}$ (Figure 3(d)). It is important to clarify that the configuration illustrated in Figure 3 demonstrates a specific case where the selected vertex b has two adjacent vertices, resulting in the permutation being divided into three segments. This example generalizes to a key principle of our method: for any selected vertex v with k adjacent vertices, the entire permutation is partitioned into $k + 1$ parts. This dynamic partitioning mechanism, which depends on the degree of the selected vertex, distinguishes our approach from the related reduction technique presented in Duarte et al. (2011).

2.3.2. Tabu list

To strategically accept deteriorating solutions during the search process, we employ a tabu list mechanism designed to forbid the reversal of recently performed operations. This mechanism specifically controls swap operations by systematically recording the pairs of vertices (u, v) involved in each executed *Swap* move. Following every swap operation, the corresponding vertex pair is added to the tabu list and prohibited for a dynamically adjusted number of subsequent iterations, known as the *tabu tenure*.

The tabu tenure undergoes dynamic adaptation throughout the search process, implementing established methodologies from graph layout optimization literature (Ren et al., 2019, 2020). This adaptive approach serves two critical purposes: preventing solution cycling by avoiding immediate reversals of recent moves, while simultaneously promoting search diversity through systematic exploration of new solution regions.

The adaptation mechanism employs a periodic step function (PS) (Galinier et al., 2011; Wu and Hao, 2013) defined over the iteration count ($iter$). This function operates with a fixed

period of 1500 iterations, which is partitioned into 15 consecutive intervals each spanning 100 iterations. For any given iteration $iter$, the tabu tenure value is computed according to the following precise mathematical formulation:

$$\text{Tenure}(iter) = 10 \times a_j + (\text{rand}() \bmod 10), \tag{12}$$

where the interval index j is determined by the expression $j = \lfloor \frac{iter \bmod 1500}{100} \rfloor + 1$, the coefficient sequence is explicitly defined as $(a_j)_{j=1}^{15} = (1, 2, 1, 4, 1, 2, 1, 8, 1, 2, 1, 4, 1, 2, 1)$.

2.4. Greedy-optimized perturbation stage

To escape local optima through search diversification, we introduce a greedy-optimized perturbation stage based on a destroy-and-repair framework, which achieves an effective trade-off between diversification and solution quality. The stage begins by destroying a complete solution φ: we remove all critical vertices and a random subset of non-critical vertices to obtain a partial solution $\bar{\varphi}$, thereby promoting diversification through reassignment across these vertices. Crucially, this partial solution preserves fixed labels for vertices not removed. We then perform repair through a dual-phase procedure: (1) a greedy assignment step minimizes immediate quality degradation during each label reassignment (Section 2.4.2); (2) an optimized assignment step maximizes solution quality by transforming the problem into a linear bottleneck formulation and employing a polynomial-time threshold algorithm for optimal label reassignment within independent sets (Section 2.4.3). The entire process targets the constrained partial antibandwidth maximization problem (CPAMP, formalized in Section 2.4.1), ensuring that quality improvements directly benefit the original AMP objective.

The pseudocode of this stage is presented in Algorithm 3. The procedure starts by setting V_{remove} to contain all critical vertices and $p \times |V|$ randomly selected non-critical vertices (line 3), then initializes V_{ind} and V_{greedy} (lines 4–5). The set of free labels L_{free} is obtained according to V_{remove} and the current solution φ (line 6). We remove the vertices of V_{remove} from the current solution φ to obtain a partial solution $\bar{\varphi}$ (line 7). A while-loop partitions V_{remove} into an independent set V_{ind} and its complement V_{greedy} (lines 8–14). In each iteration, a random vertex $x \in V_{remove}$ is added to V_{ind} (lines 9-10), and all adjacent vertices of x in V_{remove} stored in S' are added to V_{greedy} (lines 11–12). Vertex x and its neighbors in S' are then removed from V_{remove} (line 13). The above operations repeat until V_{remove} is empty. Next, GreedyAssignment repairs solution $\bar{\varphi}$ by reassigning V_{greedy} vertices (line 15), while OptimizedAssignment reassigns V_{ind} vertices to yield a complete solution φ (line 16). The local best solution φ_{lb} is updated using the extended evaluation function f_e (lines 17–19). Finally, both the current solution φ and local best solution φ_{lb} are returned (line 20). Notably, line 9 of Algorithm 3 corresponds to the step of constructing an independent set from the selected vertices. In addition to the random selection strategy, we also consider

Algorithm 3 Greedy-optimized perturbation stage (GOPS)

Input: Input solution φ, local best solution φ_{lb}, extended evaluation function f_e and sampling percentage of random vertices p

Output: The current solution φ and the local best solution φ_{lb}

/*V_{remove} is the vertex set of all the critical vertices and $p \times |V|$ random selected non-critical vertices.*/
$V_{remove} \leftarrow FetchCritical(\varphi) \cup SelNonCrit(\varphi, p)$

$V_{ind} \leftarrow \emptyset$ /*V_{ind} is an independent set of V_{remove}.*/

$V_{greedy} \leftarrow \emptyset$ /*V_{greedy} is the complement of V_{ind} with respect to V_{remove}.*/

$L_{free} \leftarrow FetchFreeLabels(\varphi, V_{remove})$

$\bar{\varphi} \leftarrow RemoveV(V_{remove}, \varphi)$

while $V_{remove} \neq \emptyset$ **do**

    $x \leftarrow RandomSelV(V_{remove})$

    $V_{ind} \leftarrow V_{ind} \cup \{x\}$

    /*S' is the set of all vertices in V_{remove} that are adjacent to vertex x.*/
    $S' \leftarrow VertexAdj(V_{remove}, x)$

    $V_{greedy} \leftarrow V_{greedy} \cup S'$

    $V_{remove} \leftarrow V_{remove} \setminus (\{x\} \cup S')$

end while

/*GreedyAssignment is designed for reassigning the vertices in the set V_{greedy} following a greedy method, See Section 2.4.2.*/
$(\bar{\varphi}, L_{free}) \leftarrow GreedyAssignment(\bar{\varphi}, V_{greedy}, L_{free})$

/*OptimizedAssignment is to reassign the vertices in the independent set V_{ind} by a polynomial threshold algorithm, See Section 2.4.3.*/
$\varphi \leftarrow OptimizedAssignment(\bar{\varphi}, V_{ind}, L_{free})$

if $f_e(\varphi) > f_e(\varphi_{lb})$ **then**

    $\varphi_{lb} \leftarrow \varphi$

end if

return φ, φ_{lb}

several greedy heuristics, such as prioritizing critical vertices, or vertices with higher degrees. Our preliminary experiments indicate no significant performance difference between random selection and these alternative strategies. Therefore, to maintain simplicity and clarity in the algorithm description, we adopted random selection for constructing the independent set.

2.4.1. Constrained partial AMP

The complete repair procedure is formalized as a constrained partial assignment maximization problem (CPAMP). Formally, given an undirected graph $G = (V, E)$ and a partial solution $\bar{\varphi} : V_{fix} \to \{1, 2, \ldots, n\}$ that fixes labels for vertex subset $V_{fix} \subseteq V$, the CPAMP seeks a complete embedding $\varphi : V \to \{1, 2, \ldots, n\}$ extending $\bar{\varphi}$ such that: $\varphi(v) = \bar{\varphi}(v), \forall v \in V_{fix}$. The edge set decomposes into $E_{fix} = \{\{u, v\} \in E \mid u, v \in V_{fix}\}$ and $E_{free} = E \setminus E_{fix}$. The objective function maximizes the antibandwidth for E_{free}:

$$AB(G, E_{free}, \varphi) = \min_{\{u,v\} \in E_{free}} |\varphi(u) - \varphi(v)|. \tag{13}$$

Thus, CPAMP aims to find an embedding φ^* maximizing $AB(G, E_{free}, \varphi)$ given G and $\bar{\varphi}$.

Remark 1. *The AMP is a relaxation of the constrained partial AMP (CPAMP), as the latter introduces additional constraints through fixed labels. A fundamental property of this relaxation is that the optimal value of the AMP is an upper bound for the optimal value of the CPAMP. Specifically, for any solution φ, the objective value of the AMP is never less than that of the CPAMP.*

Proof. By Eq. (1) and Eq. (13), the global antibandwidth satisfies:

$$AB(G, E, \varphi) = \min\left(AB(G, E_{fix}, \varphi),\ AB(G, E_{free}, \varphi)\right).$$

The inequality $AB(G, E_{free}, \varphi) \geq AB(G, E, \varphi)$ holds universally, since the minimum function value cannot exceed either component. □

The perturbation stage includes all critical vertices in V_{free} (denoted V_{remove} in Algorithm 3). Crucially, improvements to $AB(G, E_{free}, \varphi)$ directly enhance the bottleneck edges governing $AB(G, E, \varphi)$. Therefore, maximizing $AB(G, E_{free}, \varphi)$ during repair simultaneously optimizes both CPAMP and AMP objectives.

2.4.2. Greedy assignment step

The greedy assignment step repairs the partial solution $\bar{\varphi}$ by relabelling vertices in set V_{greedy} using labels from L_{free}, as specified in Algorithm 4. This iterative procedure executes a while-loop until V_{greedy} becomes empty. During each iteration, the algorithm first selects a random vertex $u \in V_{greedy}$ (line 4). It then evaluates every label in L_{free} against the partial solution $\bar{\varphi}$ using the objective function defined in Eq. (13), identifying the label l that maximizes the objective value (line 5). The selected label l is assigned to vertex u (line 6), followed by removing vertex u from V_{greedy} and label l from L_{free} (lines 7-8). Finally, the procedure returns the updated solution $\bar{\varphi}$ and the remaining label set L_{free} (line 10).

Algorithm 4 Greedy assignment step (GreedyAssignment)

Input: Input partial solution $\bar{\varphi}$, vertex set V_{greedy} and the set of free labels L_{free}
Output: The updated solution $\bar{\varphi}$ and the free label set L_{free}
while $V_{greedy} \neq \emptyset$ **do**
    $u \leftarrow RandomSel(V_{greedy})$
    $l \leftarrow FindBestFreeLabel(u, \bar{\varphi}, L_{free})$
    $\bar{\varphi}(u) \leftarrow l$
    $V_{greedy} \leftarrow V_{greedy} \setminus \{u\}$
    $L_{free} \leftarrow L_{free} \setminus \{l\}$
end while
return $\bar{\varphi}$, L_{free}

2.4.3. Optimized assignment step

The optimized assignment step repairs the partial solution $\bar{\varphi}$ by relabeling vertices in the independent set V_{ind} using available labels from L_{free}. While exhaustive search over all $k!$ possible assignments (for $|V_{ind}| = k$) yields $O(k!)$ complexity, and conventional greedy methods (Section 2.4.2) use heuristic approaches, our method optimally solves this CPAMP subproblem in polynomial time.

This approach is based on a fundamental reduction: when $V_{free} \subseteq V$ is an independent set, the CPAMP is equivalent to the linear bottleneck assignment problem (LBAP) (Pferschy, 1997), as formalized in Proposition 1. The polynomial-time solvability follows from a threshold algorithm implementation (Garfinkel, 1971) presented in Appendix B.

Proposition 1. *For independent set $V_{free} \subseteq V$, the CPAMP reduces to the linear bottleneck assignment problem and admits an $O(|V_{free}|^3)$ solution.*

Proof. We establish equivalence by constructing an LBAP instance where optimal solutions correspond directly to optimal CPAMP assignments. Complete proof appears in Appendix B. □

2.5. Discussion

This section discusses the differences between this work and related studies. It also provides further insights into the proposed algorithm.

A key distinction between the proposed algorithm, DSHA, and existing methods is the design of its neighborhood search, specifically the implementation of a reduced neighborhood, as detailed in Section 2.3.1. The neighborhood function is a fundamental component of any local search algorithm. For the AMP, the prevalent approach to constructing neighborhoods is the swap operation, which our work also adopts. Existing literature on this topic can

be categorized into three primary methods. The first employs a full swap neighborhood, as in Rodriguez-Tello and Betancourt (2011), which executes an exhaustive evaluation of all possible swaps. The second strategy uses a neighborhood portfolio to broaden the search scope and avoid premature convergence to local optima. A representative example is the variable neighborhood search framework in Lozano et al. (2012), which alternates among three distinct swap-based neighborhoods, including single-swap, consecutive-swap, and ejection-chain swaps. The third method filters the most promising neighboring solutions based on graph structure to accelerate evaluation, exemplified by Duarte et al. (2011), where vertices are partitioned into three groups relative to a critical vertex to guide swap selection. A common feature across these prior methods is their reliance on a local descent paradigm, selecting the best neighboring solution at each step and halting when no improving moves exist. In contrast, our method introduces a more sophisticated reduction technique embedded within a tabu search metaheuristic. The core advance lies in a dynamic partitioning scheme based on the degree of the critical vertex v, which divides all vertices into $k+1$ segments, where k denotes the degree of v. This structure enables the algorithm to focus on the most promising swaps while substantially reducing the neighborhood size. This reduction significantly accelerates the local optimization phase, particularly for instances with large size or high graph density. Furthermore, by operating within the tabu search framework, our algorithm gains enhanced exploratory power through the conditional acceptance of non-improving moves, enabling it to escape local optima and navigate a broader region of the solution space.

A further key difference of our work from the literature is the design of a greedy-optimized perturbation stage. Crafting an effective perturbation mechanism to escape local optima remains a central challenge in local search methodology. Common strategies in the literature include straightforward solution reinitialization after each local optimization phase (Lozano et al., 2012). Population-based algorithms, for their part, typically generate new offspring through search-space exploration (Duarte et al., 2011), employ random mutation or reinitialization (Bansal and Srivastava, 2010), or exploit structural similarities in parent solutions (Rodriguez-Tello and Betancourt, 2011). Departing from these approaches, our proposed DSHA is grounded in the inherent properties of the AMP and its theoretical link to the LBAP. We formally prove that a constrained form of the AMP can be reduced to an LBAP, a problem solvable in polynomial time. Building on this foundation, we developed a dedicated perturbation stage. This component is designed to strategically balance diversification and intensification, systematically generating new, high-quality starting points for local search rather than relying on random moves.

Besides that, the AMP exhibits a symmetry property in its solution structure. For instance, the permutations (1, 2, 3, 4, 5) and (5, 4, 3, 2, 1) represent the same solution (a reversal) and consequently yield the same objective value. Therefore, for a graph with n vertices, the effective search space consists of $n!/2$ unique solutions rather than $n!$. This symmetry characteristic can

notably affect population-based metaheuristics, such as evolutionary algorithms. A population may inadvertently contain pairs of symmetric solutions that are identical in quality, which reduces genetic diversity and can hinder the search process. Symmetry can also complicate algorithms that rely on operations binding labels to vertices. Our proposed DSHA algorithm inherently addresses this challenge in two key ways. First, as a trajectory search method operating on a single solution, it does not maintain a population and therefore avoids diversity issues arising from symmetric solutions in the pool. Second, DSHA guides its search by tracking the number of edges that attain the current minimum absolute label difference. This metric is sensitive to the permutation structure and enables the algorithm to distinguish between symmetric solutions that share the same objective value but differ in their configuration across edges, thereby improving navigation within the symmetric search space.

2.6. Complexity analysis

Given an AMP instance $G = (V, E)$ with $n = |V|$ vertices, $m = |E|$ edges, and maximum degree Δ, we analyze the complexity of DSHA by considering its construction phase, DTSS, GOPS, and the overall time-bounded framework. For GOPS, let $s = |V_{remove}|$, $r = |V_{greedy}|$, and $k = |V_{ind}|$. The construction phase builds the BFS structure, assigns labels greedily, and evaluates the resulting solution, yielding a total cost of $O(n(n + m))$.

In DTSS, one outer iteration identifies critical vertices and explores the reduced neighborhood of each retained critical vertex. If $r_u = |S_v(u, \varphi)|$ is the number of candidate swap partners for a critical vertex u, the reduced-neighborhood construction, candidate evaluation, and move update lead to $O\big(n + \sum_{u \in S_u} \big(n + \deg(u)^2 + r_u(n + \Delta) + \Delta^2\big)\big)$, which is $O(n^3)$ in the worst case. Let I_{DTSS} be the number of outer iterations executed in one DTSS call. Since this number depends on the search trajectory and the cutoff time, one DTSS call costs $O(I_{DTSS}n^3)$.

For GOPS, selecting and removing vertices and constructing V_{ind} require $O(n + s\Delta)$ time. The greedy assignment step costs $O(r(n+s\Delta+\Delta^2))$. The optimized assignment step constructs an LBAP instance in $O(n + k^2\Delta)$ time, solves it by the threshold algorithm in $O(k^3)$ time (Garfinkel, 1971; Pferschy, 1997), and applies the assignment in $O(n + k\Delta^2)$ time. Hence, one GOPS call is bounded by $O(n^2\Delta + k^3)$, which becomes $O(n^3)$ in the worst case. In practice, k is much smaller than n because only the critical vertices and a small fraction of non-critical vertices are selected, with $p = 0.06$ after parameter tuning (see Section 3.2).

Let H be the number of completed main-loop iterations before the cutoff time T_{max}. The overall running time is $O\big(n(n+m) + H(I_{DTSS}n^3 + n^2\Delta + k^3)\big)$. This expression represents the work performed by the time-bounded heuristic, since H and I_{DTSS} are governed by the search trajectory and the prescribed time limit rather than by n alone. For a completed main-loop iteration in the worst case, the cost is $O((I_{DTSS} + 1)n^3)$, which simplifies to $O(I_{DTSS}n^3)$.

3. Experimental results

We conduct rigorous benchmark experiments to evaluate DSHA, presenting the experimental configuration (Section 3.1), the parameter calibration methodology (Section 3.2), and the comparative results against reference algorithms (Section 3.3).

3.1. Experimental setup

Table 1: Composition of 256 benchmark instances across 11 sets. The first 10 sets are from Lozano et al. (2012), while the last set was introduced by Scott and Hu (2014). Optimal solutions are known for path, cycle, mesh, toroidalmesh, hypercube, cbt, and hamming sets (Lin and Yuan, 2003; Wang et al., 2009; Calamoneri et al., 2009; Raspaud et al., 2009), while unknown for caterpillar, 3dmesh, Harwell-Boeing and Sparse-Matrix sets.

Set	Num.	Vertices	Optimum
path	24	50-1000	Known
cycle	24	50-1000	Known
mesh	24	81-1170	Known
toroidalmesh	37	16-1600	Known
hypercube	7	16-1024	Known
cbt	24	30-950	Known
hamming	24	80-1152	Known
caterpillar	40	20-1000	Unknown
3dmesh	8	12-3600	Unknown
Harwell-Boeing	24	30-900	Unknown
Sparse-Matrix	20	234-504855	Unknown

We utilize the established benchmark suite from Lozano et al. (2012) and Scott and Hu (2014). The whole suite comprises 256 instances categorized into 11 distinct sets. Optimal solutions are known for the first seven data sets, but not for the remaining ones. Table 1 provides a comprehensive description of these benchmark sets, with the 'Set' column specifying each category name, 'Num.' indicating the instance count per category, 'Vertices' detailing the vertex count range, and 'Optimum' reporting solution status (Known/Unknown). Among them, 24 instances are from the Harwell-Boeing sparse matrix collection[1], and 20 are 'Sparse-Matrix' instances sourced from the University of Florida sparse matrix collection[2], both representing real-world engineering applications. Notably, seven 'Sparse-Matrix' instances contain over

[1] https://math.nist.gov/MatrixMarket/data/Harwell-Boeing
[2] https://sparse.tamu.edu/

30000 vertices, constituting hard cases for evaluating algorithm performance[3].

The DSHA algorithm was implemented in C++ using g++ 7.5.0 compiler with -O2 optimization flag. All experiments were executed on a Linux system equipped with 2.1 GHz Intel Xeon 5318Y processors.

Since the source code of the baseline algorithms was unavailable, we compared our results against the results reported in the respective literature. To ensure a fair comparison of computational time across different hardware, we applied a standard CPU performance conversion method[4] widely adopted in benchmarking studies (Pecin et al., 2017). The detailed conversion factors are provided in Table 2. The experimental time limits were set with reference to prior work. For the first ten instance sets, the stopping condition in Lozano et al. (2012) was 150 seconds. Accordingly, we adopted a primary comparison time limit of 78 seconds, which is our hardware-calibrated equivalent. While Sinnl (2021) and Fazekas et al. (2020) used a 1800-second limit, our converted equivalent time far exceeds 150 seconds. Therefore, to maintain consistency and focus the performance comparison, the results at the 78-second limit serve as our main basis for comparison against the literature. We also provide results under a 150-second limit, however, this is primarily to illustrate the convergence behavior of our algorithm over a longer runtime, rather than for core performance benchmarking. For the Sparse-Matrix instances, following the 3600-second limit set in Scott and Hu (2014), we set our maximum running time to 1968 seconds after CPU time conversion. Adhering to standard benchmarking practices, each instance was solved once. We acknowledge that hardware differences (e.g., memory configuration and other system-specific factors) beyond CPU performance may introduce additional variability, and thus this comparison is only relatively fair. To facilitate future research, the source code of our algorithm, the benchmark instances, and the solution certificates are publicly available at `https://github.com/REN-Jintong/DSHA-AMP`. Moreover, to further evaluate the performance of the proposed algorithm, we use the latest version (22.1.2) of the CPLEX optimizer to solve all 256 benchmark instances of the AMP under a time limit of 1800 seconds. The computational results show that the proposed algorithm achieves clear advantages over CPLEX on these benchmark instances, and the corresponding detailed results are also reported in our GitHub repository.

3.2. Tuning of parameters

This section details the methodology for determining DSHA's two input parameters. Parameter tuning employs the *irace* automatic configuration toolkit (López-Ibáñez et al., 2016), a

[3]It is worth noting that among the 24 instances introduced by Scott and Hu (2014), six (curtis54, can_445, 662_bus, nos6, dwt_234, and sherman4) also appear in the Harwell-Boeing set. Four of these (curtis54, can_445, 662_bus, and nos6) are identical in both collections, while the remaining two (dwt_234 and sherman4) are entirely different instances. Therefore, the Sparse-Matrix category effectively contains 20 unique new instances.

[4]`https://www.cpubenchmark.net/singleThread.html`

Table 2: CPU Time Conversion for Benchmark Instances. Lozano et al. (2012) employed a 2.4 GHz Intel Core2 Quad CPU Q8300. Sinnl (2021) ran his experiments in a 2.5 GHz Intel Xeon E5v4. Fazekas et al. (2020) carried out their experiments in a 2.10 GHz Intel Xeon E5-2620 v4. Scott and Hu (2014) used a 2.4 GHz Intel Xeon E5620. This work was run in a 2.1 GHz Intel Xeon 5318Y.

Instances	Experimental settings in the literature	This work
First 9 sets	Lozano et al. (2012): 150 seconds	78 seconds + 150 seconds
Harwell-Boeing	Lozano et al. (2012): 150 seconds Fazekas et al. (2020) and Sinnl (2021): 1800 seconds	78 seconds + 150 seconds
Sparse-Matrix	Scott and Hu (2014): 3600 seconds	1968 seconds

state-of-the-art framework leveraging iterated racing procedures and Friedman's non-parametric rank-based ANOVA (MacFarland and Yates, 2016). Its operation requires three inputs: the parameter space (Table 3), the algorithm's executable, and representative training instances.

Table 3: Parameter space for DSHA configured via *irace* (López-Ibáñez et al., 2016).

Parameter	Type	Range/Values	Description
L	Categorical	{20,40,50,100,200,250,350,400,600,700,800}	Tabu search depth
p	Float	(0.01, 1.00)	Vertex sampling percentage

The tuning process allocated a computational budget of 3000 runs. Training utilized 12 strategically selected instances from the 256-instance benchmark, chosen for their large size and solution difficulty. The optimized parameter configuration yielded $L = 600$ and $p = 0.06$. It is worth noting that parameter tuning is a common practice in heuristic algorithm development. In our experiments, this parameter configuration is applied to all instances, and no fine-tuning is performed for individual instances.

3.3. Comparisons with state-of-the-art algorithms

This section provides a comprehensive comparative evaluation of the proposed DSHA algorithm against state-of-the-art reference methods across the 256 benchmark instances. The main algorithms used for comparison are: the memetic algorithm (MA from Bansal and Srivastava (2010)), the GRASP with evolutionary path relinking algorithm (EvPR from Duarte et al. (2011)), the variable neighborhood search algorithm (VNS from Lozano et al. (2012)), the level-based algorithm (LBA from Scott and Hu (2014)), the GSpectral algorithm (GSpectral from Scott and Hu (2014)), the duplex encoding method (Duplex from Fazekas et al. (2020)), and the mixed integer programming approach (MIP from Sinnl (2021)). Performance on the first seven instance sets (164 instances with known optima) is summarized in Table 4, while detailed results for 3dmesh and caterpillar sets are provided in Table 5. Harwell-Boeing

instances (Table 6) include additional comparisons with Duplex, MIP and LBA.In Table 7, we presented the comparative results for Sparse-Matrix instances, comparing with the algorithms in Scott and Hu (2014), including GSpectral, EvPR and LBA. For Tables 4 to 6, following the CPU time conversion described in Table 2, the results obtained within 78 seconds, denoted by DSHA (78s), are used as the primary basis for comparison. The results obtained within 150 seconds, denoted by DSHA (150s), are also reported to illustrate the convergence behavior of the proposed algorithm. The results in Table 4 for the first seven instance sets are presented in aggregated form, with statistics summarized at the set level. This format is adopted to align with the comparative baseline established in the primary reference (Lozano et al., 2012). For the remaining instance sets, a detailed comparison of the objective value for each instance against other algorithms is provided. The complete set of detailed results for every individual instance across all benchmark sets, along with the corresponding solution certificates, is publicly available at `https://github.com/REN-Jintong/DSHA-AMP` to ensure full reproducibility and to facilitate further analysis.

Table 4 presents comparative results between the three reference algorithms and the proposed DSHA algorithm across the first seven benchmark sets with known optimal solutions. The columns are structured as follows. The 'Set' column identifies the instance family, 'Num.' specifies the number of instances per set, 'OPT(%)' indicates the percentage of instances where each algorithm found optimal solutions (with superior values highlighted in bold), and 'GAP(%)' represents the average gap to the optimality calculated as $GAP(\%) = \frac{(Opt-Obj)\times 100}{Opt}$ to quantify solution quality (smaller values denote better quality). The subsequent eight columns provide identical metrics for EvPR, VNS, and DSHA (78s) and DSHA (150s) respectively. The final row ('Avg') reports aggregate statistics including column-wise averages across all 164 instances.

Table 4: Overall results obtained by 3 reference algorithms and our algorithm for 7 sets of instances (164 instances) with known optimal solutions.

Set	Num.	MA		EvPR		VNS		DSHA (78s)		DSHA (150s)	
		OPT(%)	GAP(%)	OPT(%)	GAP(%)	OPT(%)	GAP(%)	OPT(%)	GAP(%)	OPT(%)	GAP(%)
path	24	87.50	0.04	16.67	1.88	**100.00**	0.00	50.00	0.45	58.33	0.28
cycle	24	66.67	0.88	16.67	2.46	66.67	0.84	**87.50**	0.12	**87.50**	0.12
mesh	24	45.83	0.13	12.50	3.02	20.83	0.92	**54.17**	0.59	**54.17**	0.58
toroidal	37	29.73	38.59	56.76	5.48	54.05	6.49	75.68	0.83	**78.38**	0.49
hypercube	7	**100.00**	0.00	57.14	1.35	71.43	1.15	**100.00**	0.00	**100.00**	0.00
cbt	24	0.00	18.79	**4.17**	3.94	**4.17**	3.92	**4.17**	3.43	**4.17**	3.02
hamming	24	0.00	57.99	0.00	30.80	4.17	17.06	0.00	17.24	0.00	16.30
All	164	40.24	20.10	22.56	7.45	43.90	4.84	50.00	3.38	**51.83**	3.08

Comparing DSHA (78s) with the three main reference algorithms from the literature,

analysis of Table 4 reveals three key performance patterns. First, DSHA (78s) demonstrates better performance on cycle, mesh, and toroidal sets, achieving the highest OPT(%) values (87.50%, 54.17%, and 75.68% respectively). Second, on hypercube instances, DSHA (78s) matches MA's perfect 100% optimal solution rate while significantly outperforming other references. Third, for cbt instances, although two reference algorithms achieve identical OPT(%) (4.17%), DSHA (78s) delivers superior solution quality with a lower GAP(%) (3.43 vs. 3.92 and 3.94). Globally, DSHA (78s) outperforms all references with a 50.00% optimal solution rate - 13.90% higher than the next-best (VNS at 43.90%) and 30.17% lower average deviation (3.38 vs. VNS's 4.84). However, performance limitations are noted on path instances where VNS achieves 100% optimal solutions versus DSHA's 50.00%, and on hamming sets where DSHA fails to find optimal solutions (0% vs. VNS's 4.17%). These areas represent promising directions for future algorithmic improvements. Comparing DSHA (78s) with DSHA (150s) shows that the extended runtime leads to improvements in certain metrics. Specifically, DSHA (150s) achieves higher OPT(%) values on path and toroidal instance sets. It also yields slight improvements in GAP(%) for mesh, cbt, and hamming sets. Overall, our proposed algorithm demonstrates strong performance across the seven standard benchmark sets compared to the three main reference algorithms, and its results show further potential when allowed a longer running time.

Table 5 presents comparative results on 48 instances from 3dmesh and caterpillar sets where optimal solutions are unknown. The columns are defined as follows. 'Ins' identifies the instance name, 'UB' indicates the best-known upper bound, 'BestFind' reports the best result from the three reference algorithms, 'DSHA (78s)' and 'DSHA (150s)' shows the solution from our proposed algorithm obtained in 78 and 150 seconds respectively, and 'δ' quantifies the improvement (DSHA (150s) - BestFind). Superior values are highlighted in bold. The table is divided into left and right partitions for space efficiency. From Table 5, one notices that DSHA (78s) matches or exceeds reference performance on 46 instances, achieving 20 strict improvements and 26 ties. For caterpillar instances, DSHA (78s) reaches the upper bound in 31 cases compared to just 19 attained by the best reference algorithm. When the running time is extended to 150 seconds, DSHA (150s) demonstrates the capability to match or exceed the reference algorithm in all instances.

Table 6 presents a comprehensive comparison of the proposed DSHA algorithm against five state-of-the-art reference methods on 24 the Harwell-Boeing instances. The column structure includes: 'Ins' for instance identification, 'Vertex' specifying vertex counts, 'UB' indicating best-known upper bounds (Duarte et al., 2011; Sinnl, 2021), 'BestFind' reporting the best-known values from the literature, 'HeuristicAlgo' showing the best composite result from MA, EvPR, VNS and LBA (Duarte et al., 2011; Lozano et al., 2012; Scott and Hu, 2014), 'Duplex' and 'MIP' displaying results from Fazekas et al. (2020) and Sinnl (2021), and DSHA (78s) and DSHA (150s) presenting our results produced in 78 seconds and 150 seconds, respectively.

Table 5: Performance comparison on 48 instances with unknown optima (3dmesh and caterpillar sets). The upper bounds (UB) come from Lozano et al. (2012). 'BestFind' denotes the best result among MA, EvPR, and VNS.

Ins	UB	BestFind	DSHA (78s)	DSHA (150s)	δ	Ins	UB	BestFind	DSHA (78s)	DSHA (150s)	δ
3dmesh_3	12	**9**	**9**	**9**	0	3dmesh_4	31	**25**	**25**	**25**	0
3dmesh_5	61	**51**	**51**	**51**	0	3dmesh_6	107	**92**	**92**	**92**	0
3dmesh_7	170	149	**150**	**150**	1	3dmesh_8	255	**228**	**228**	**228**	0
3dmesh_9	363	328	328	**330**	2	3dmesh_10	499	454	451	**456**	2
caterpillar_5_4	10	**10**	**10**	**10**	0	caterpillar_5_5	12	**12**	**12**	**12**	0
caterpillar_5_6	15	**15**	**15**	**15**	0	caterpillar_5_7	17	**17**	**17**	**17**	0
caterpillar_9_4	18	**18**	**18**	**18**	0	caterpillar_10_4	20	**20**	**20**	**20**	0
caterpillar_9_5	22	**22**	**22**	**22**	0	caterpillar_10_5	25	**25**	**25**	**25**	0
caterpillar_9_6	27	**27**	**27**	**27**	0	caterpillar_10_6	30	**30**	**30**	**30**	0
caterpillar_15_4	30	**30**	**30**	**30**	0	caterpillar_9_7	31	**31**	**31**	**31**	0
caterpillar_10_7	35	**35**	**35**	**35**	0	caterpillar_15_5	37	**37**	**37**	**37**	0
caterpillar_20_4	40	**40**	**40**	**40**	0	caterpillar_15_6	45	**45**	**45**	**45**	0
caterpillar_20_5	50	**50**	**50**	**50**	0	caterpillar_15_7	52	**52**	**52**	**52**	0
caterpillar_20_6	60	**60**	**60**	**60**	0	caterpillar_20_7	70	69	**70**	**70**	1
caterpillar_20_10	100	99	**100**	**100**	1	caterpillar_25_10	125	124	**125**	**125**	1
caterpillar_20_15	150	148	**150**	**150**	2	caterpillar_30_10	150	149	**150**	**150**	1
caterpillar_35_10	175	174	**175**	**175**	1	caterpillar_25_15	187	185	**187**	**187**	2
caterpillar_20_20	200	197	**199**	**199**	2	caterpillar_40_10	200	199	**200**	**200**	1
caterpillar_30_15	225	222	**225**	**225**	3	caterpillar_20_25	250	247	**249**	**249**	2
caterpillar_25_20	250	247	**250**	**250**	3	caterpillar_35_15	262	260	**261**	**261**	1
caterpillar_30_20	300	297	**300**	**300**	3	caterpillar_40_15	300	297	**300**	**300**	3
caterpillar_25_25	312	**309**	**309**	**309**	0	caterpillar_35_20	350	347	**349**	**349**	2
caterpillar_30_25	375	371	**373**	**373**	2	caterpillar_40_20	400	397	**399**	**399**	2
caterpillar_35_25	437	433	435	**436**	3	caterpillar_40_25	500	496	493	**498**	2
Compare DSHA (78s) with reference algorithms: Win: 20, Match: 26, Lose: 2											
Compare DSHA (150s) with reference algorithms: Win: 23, Match: 25, Lose: 0											

The last column quantifies improvements over BestFind, with '-' denoting unavailable data and asterisks (*) marking 11 instances with proven optimality (Sinnl, 2021). The summary row provides aggregate performance metrics: Win (new records), Match (ties with best), and Lose (inferior results). Notably, while Duplex and MIP employ exact methods, their 1800-second runtime limit yields lower bounds rather than guaranteed optima for several challenging instances.

A comparative analysis of DSHA (78s) against all reference algorithms yields three key findings: (1) DSHA (78s) successfully verifies optimal solutions for all 11 provably optimal instances; (2) On the remaining 13 instances, it establishes 8 new records and matches 4 best-known solutions, with only one marginally inferior result (22 vs Duplex's 23); (3) Substantial lower-bound improvements are achieved: 10.91% for impcol_d, 12.94% for can__445, and 15.92% for dwt__592, demonstrating significant advancements in solution quality for computationally demanding cases. Extending the runtime to 150 seconds results in only minor improvements for three instances (can__445, dwt__503, and dwt__592). This indicates that DSHA achieves most of its performance gains rapidly, demonstrating fast convergence.

Table 6: Comparative results on 24 Harwell-Boeing instances: HeuristicAlgo (best of MA, EvPR, VNS, LBA), Duplex and MIP (exact methods with 1800s runtime yielding lower bounds). Asterisk (*) denotes instances with proven optimal solutions.

Ins	Vertex	UB	BestFind	HeuristicAlgo	Duplex	MIP	DSHA (78s)	DSHA (150s)	δ
pores_1*	30	6	6	**6**	**6**	**6**	**6**	**6**	0
ibm32*	32	9	9	**9**	**9**	**9**	**9**	**9**	0
bcspwr01*	39	17	17	**17**	**17**	**17**	**17**	**17**	0
bcsstk01*	48	9	9	8	**9**	**9**	**9**	**9**	0
bcspwr02*	49	21	21	**21**	**21**	**21**	**21**	**21**	0
curtis54*	54	13	13	**13**	**13**	**13**	**13**	**13**	0
will57*	57	13	13	**13**	**13**	**13**	**13**	**13**	0
impcol_b*	59	8	8	**8**	**8**	**8**	**8**	**8**	0
ash85	85	27	23	22	**23**	22	22	22	-1
nos4	100	40	35	**35**	**35**	34	**35**	**35**	0
dwt__234	117	57	51	**51**	49	**51**	**51**	**51**	0
bcspwr03*	118	39	39	**39**	**39**	**39**	**39**	**39**	0
bcsstk06	420	38	34	33	34	33	**36**	**36**	2
bcsstk07	420	38	34	33	34	33	**36**	**36**	2
impcol_d	425	141	110	105	99	110	**122**	**122**	12
can__445	445	120	85	85	-	78	96	**97**	12
494_bus	494	246	228	**228**	-	219	**228**	**228**	0
dwt__503	503	71	62	58	62	56	65	**66**	4
sherman4	546	272	261	**261**	-	256	**261**	**261**	0
dwt__592	592	150	113	113	-	103	131	**133**	20
662_bus*	662	220	220	**220**	**220**	**220**	**220**	**220**	0
nos6	675	337	329	329	-	326	**330**	**330**	1
685_bus*	685	136	136	**136**	**136**	**136**	**136**	**136**	0
can__715	715	142	121	121	-	112	**123**	**123**	2
Compare DSHA (78s) with reference algorithms: Win: 8, Match: 15, Lose: 1									
Compare DSHA (150s) with reference algorithms: Win: 8, Match: 15, Lose: 1									

Table 7 presents a comprehensive comparison between the proposed DSHA algorithm and three state-of-the-art reference methods across the 20 large Sparse-Matrix instances. The columns denote instance identification (Ins), vertex count (Vertex), edge count (Edge), best-known literature solutions (BestFind), results from three algorithms in Scott and Hu (2014) run up to 3600 seconds, our DSHA results under a 1968-second limit (after CPU time conversion using single-thread PassMark scores), and a last column quantifying the improvement over BestFind. The symbol '-' indicates unavailable data. A summary row aggregates performance via Win (new records), Match (ties), and Lose (inferior results). Notably, the last seven instances (big_dual, ship_001, brack2, shipsec8, fcondp2, msdoor, af_shell1) have over 30000

Table 7: Comparative results on 20 Sparse-Matrix instances: GSpectral, EvPR and LBA (with a maximum running time of 3600 seconds). '-' indicates unavailable data. Considering conversion of the CPU performance, our algorithm DSHA were run during 1968 seconds.

Ins	Vertex	Edge	BestFind	GSpectral	EvPR	LBA	DSHA	δ
dwt_234	234	300	80	77	-	80	**109**	29
saylr1	238	445	111	72	107	111	**112**	1
grid1	252	476	116	77	114	116	**117**	1
nos7	729	1944	330	177	324	**330**	**330**	0
nos5	468	2352	66	64	**66**	49	**66**	0
saylr3	1000	1375	627	249	-	633	**641**	14
sherman4	1104	1341	815	274	-	815	**816**	1
netz4504	1961	2578	671	652	-	671	**958**	287
lshp2614	2614	7683	512	512	-	337	**650**	138
grid2	3296	6432	1626	823	-	**1626**	1623	-3
saylr4	3564	9376	1726	873	-	1726	**1727**	1
sherman3	5005	7514	3509	1246	-	**3509**	1666	-1843
ukerbe1	5981	7852	2054	1990	-	2054	**2532**	478
big_dual	30269	44929	10031	10031	-	6645	**12549**	2518
ship_001	34920	2304655	319	-	-	319	**406**	87
brack2	62631	366559	4984	-	-	4984	**6919**	1935
shipsec8	114919	3269240	2246	-	-	2246	**2632**	386
fcondp2	201822	5546247	4020	-	-	4020	**4101**	81
msdoor	415863	9912536	8137	-	-	**8137**	4356	-3781
af_shell1	504855	8542010	14973	-	-	**14973**	4766	-10207
Compare DSHA with literature algorithms: Win: 14, Match: 2, Lose: 4								

vertices, with the largest exceeding 500000. Given that the standard algorithm configuration employs a tabu matrix with $O(n^2)$ space complexity, along with a time-consuming construction phase and a computationally intensive neighborhood visiting strategy, these extreme-scale instances would incur prohibitive memory and time costs. To address this, three targeted minor adaptations were implemented: (1) the tabu matrix in Section 2.3.2 was reduced to a vector where u and v are independently forbidden after each swap operation; (2) the construction procedure in Section 2.1 was simplified by randomly assigning labels to remaining vertices after the initial placement of non-adjacent ones; and (3) exhaustive visits to the reduced neighborhood in Section 2.3.1 were replaced by randomly sampling 20 candidate solutions per iteration. Collectively, these modifications enable the algorithm to handle extremely large-scale instances within practical computational limits.

According to Table 7, the proposed DSHA algorithm delivers strong overall performance across the 20 large Sparse-Matrix instances, outperforming the reference results in 14 instances

and matching the best-known solutions in 2 others. Notably, DSHA achieves significant improvements on five challenging instances: 42.77% for netz4504, 26.95% for lshp2614, 23.27% for ukerbe1, 25.10% for big_dual, and 38.82% for brack2, which clearly demonstrates its effectiveness compared to the reference algorithms. However, DSHA yields inferior results on 4 instances, with notably poorer performance on 3 of them (sherman3, msdoor, and af_shell1). The instance sherman3 originates from the Harwell-Boeing collection and possesses a dedicated graph structure; our algorithm, which primarily relies on a single type of neighborhood move, is less effective in addressing this specific graph type. Designing a more diverse set of move operators is thus a promising direction for future work. The other two instances, msdoor and af_shell1, are both extremely large in scale. Our analysis indicates that the initial solutions generated for these instances were of low quality, which subsequently constrained the final solution quality. In contrast, the level-based algorithms presented in Scott and Hu (2014) are reported to provide high-quality initial solutions for massive instances. Specifically, the exact solution of the bottleneck assignment problem incurs $O(n^3)$ complexity, and perturbing merely 6% of the nodes in *af_shell1* (504,855 nodes) still requires processing over 30,000 nodes per call, imposing severe demands on both computational time and memory space. To mitigate these scalability limitations, future work will explore partitioning the selected node set into bounded subsets to alleviate the $O(n^3)$ bottleneck, developing approximate assignment solvers to further reduce time complexity, and optimizing data structures and parameters for improved memory efficiency at the million-node scale.

In summary, the proposed DSHA algorithm demonstrates superior performance over the reference algorithms across the benchmark instances tested. For the seven instance sets with known optimal solutions, DSHA achieves significantly higher success rates in locating optimal solutions. Among the remaining 92 benchmark graphs with unknown optima, it establishes improved lower bounds for 42 instances and matches best-known solutions for 43 instances. The algorithm also shows strong results on specialized instance collections: in Harwell-Boeing instances, it sets 8 new records with substantial improvements of 10.91–15.92% in three challenging instances; in Sparse-Matrix instances, it finds 14 better results and achieves improvements of 23.27–42.77% in five challenging instances. These outcomes collectively highlight the algorithm's robustness across diverse test suites structures.

4. Analysis of the key components

This section investigates two fundamental components governing algorithmic performance: the reduction technique in the neighborhood and the greedy-optimized perturbation stage. Through rigorous ablation studies on standard benchmarks, we systematically quantify their individual contributions. Section 4.1 demonstrates how neighborhood reduction enhances local search efficiency, while Section 4.2 quantitatively assesses the perturbation stage's impact on

solution quality. All experiments in this section follow the settings outlined in Section 3.1, with maximum running times set to 150 seconds for the first 10 instance sets and 1968 seconds for the Sparse-Matrix one.

4.1. Influence of the reduction technique in the neighborhood

This section analyzes the impact of neighborhood reduction techniques on algorithmic performance. The proposed method employs a dedicated reduced swap neighborhood to enhance local search efficiency. To isolate this component's effect, we created a variant named *FullSwap* that utilizes a full swap neighborhood while retaining all other features. Both algorithms were evaluated under identical experimental conditions (Section 3.1) across benchmark instances.

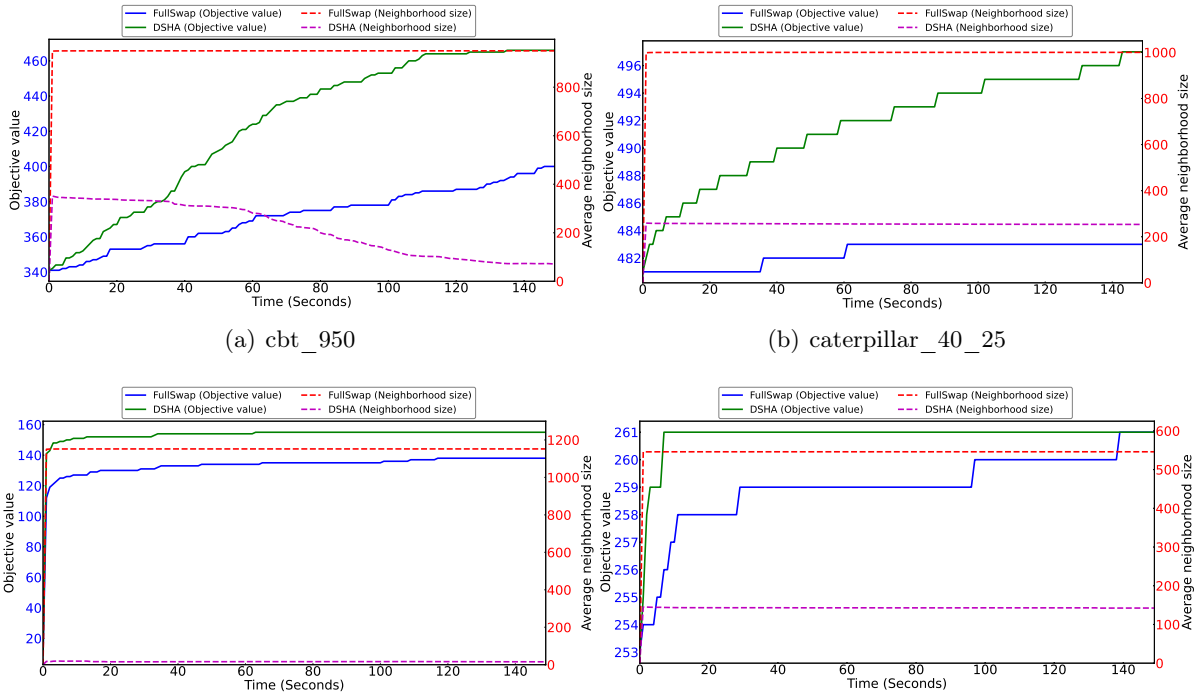
Table 8 presents comparative results with the following metrics per instance set: 'Set' (family name), 'Num.' (instance count), 'Obj' (average objective value), and 'OPT(%)' (success rate for optimal/upper-bound solutions). The last three columns quantify performance relationships: 'Win' (DSHA superior), 'Match' (equal), 'Lose' (DSHA inferior). Superior objective values are bolded.

Table 8: Overall results obtained by *FullSwap* and our algorithm for all sets of instances using the experimental settings described in Section 3.1.

Set	Num.	*FullSwap*		DSHA		*Win*	*Match*	*Lose*
		Obj	OPT(%)	Obj	OPT(%)			
path	24	108.46	29.17	**108.75**	58.33	7	17	0
cycle	24	107.42	70.83	**107.58**	87.50	4	20	0
mesh	24	270.50	41.67	**271.08**	54.17	8	16	0
toroidal	37	242.41	64.86	**278.81**	78.38	13	24	0
hypercube	7	**100.00**	100.00	**100.00**	100.00	0	7	0
cbt	24	171.79	4.17	**180.67**	4.17	12	12	0
hamming	24	65.79	0.00	**70.50**	0.00	12	5	7
3dmesh	8	165.88	0.00	**167.62**	0.00	3	5	0
caterpillar	40	144.88	60.00	**147.03**	77.50	16	24	0
Harwell-Boeing	24	81.96	41.67	**84.62**	45.83	9	15	0
Sparse-Matrix	20	2190.05	0.00	**2353.80**	0.00	12	5	3
Total Instance: 256, Win: 96, Match: 150, Lose: 10, p-value: 1.05×10^{-14}								

From Table 8, one can conclude that the DSHA with reduced neighborhood technique demonstrates significantly better performance in 10 of 11 instance sets, achieving higher average objective values than *FullSwap*. For the hypercube set, both algorithms achieve optimal solutions with identical success rates. Overall, DSHA outperforms *FullSwap* with 96 superior results, 150 ties, and only 10 inferior outcomes. This performance advantage is statistically significant according to the Wilcoxon signed-rank test ($p = 1.05 \times 10^{-14} < 0.05$), supporting

the effectiveness of the neighborhood reduction. The reported p-values are based on paired results obtained from single runs on each instance and are intended to provide complementary statistical evidence for the observed performance differences. Note that DSHA incorporates a neighborhood reduction technique in its local optimization phase. This technique accelerates the search by focusing on the most promising immediate improvements at each iteration, rather than exhaustively evaluating all possible moves. While this leads to faster convergence, it may overlook solutions that are beneficial in the long run, particularly for graphs with dense edge structures (such as the hamming instances where DSHA shows more losses). This observation provides a clear direction for future work: to develop strategies to better identify and retain solutions with long-term potential during the reduced search process.



(a) cbt_950 (b) caterpillar_40_25

(c) hamming3x4x4x4x6 (d) sherman4

Figure 4: Comparative runtime profiles of FullSwap and DSHA on four representative instances over 150 seconds. X-axis: elapsed time (seconds). Left Y-axis: solution quality (objective value). Right Y-axis: average neighborhood size. Solid curves: objective value (blue: FullSwap, green: DSHA). Dashed curves: neighborhood size evolution (red: FullSwap, purple: DSHA).

To rigorously quantify efficiency gains from neighborhood reduction, we monitored solution quality and neighborhood size dynamics during 150-second executions of FullSwap and DSHA on four representative instances (cbt_950, caterpillar_40_25, hamming3x4x4x4x6, sherman4). Figure 4 presents comparative runtime profiles with dual ordinates: left for solution quality (objective value), right for average neighborhood size, both against elapsed time

(abscissa). Solution trajectories appear as continuous solid curves (blue: FullSwap, green: DSHA), while neighborhood size evolution is shown as continuous dashed curves (red: FullSwap, purple: DSHA).

Analysis reveals two critical patterns: (1) DSHA consistently achieves superior solution quality with accelerated convergence, exemplified by the green solid curves dominating blue counterparts across all subfigures; (2) Neighborhood reduction yields much smaller search spaces (purple vs. red). Specially, there is notable inverse correlation between solution quality and neighborhood size as demonstrated in Figure 4(a), where solution improvement directly corresponds to neighborhood contraction. This efficiency arises directly from our designed negative correlation mechanism between solution quality and neighborhood size in the reduced NF framework (Section 2.3.1), where higher-quality solutions intrinsically enable more aggressive neighborhood pruning. Moreover, Figure 4 also provides empirical evidence regarding the runtime behavior and scalability of DSHA. The runtime trajectories show that DSHA consistently reaches high-quality solutions substantially earlier than FullSwap across the tested instances. This advantage becomes more evident on relatively large instances, such as hamming3x4x4x4x6 and sherman4, where the reduced neighborhood mechanism effectively controls the growth of the search space throughout the search process. These observations suggest that the proposed neighborhood reduction strategy not only improves solution quality, but also enhances search efficiency and scalability by maintaining manageable neighborhood sizes as problem complexity increases.

Collectively, Table 8 and Figure 4 provide dual validation of neighborhood reduction's efficacy: quantitatively demonstrating performance superiority across benchmark sets and revealing real-time efficiency gains during execution. This reduction technique constitutes a fundamental optimization mechanism within our algorithm, systematically enhancing search efficiency through solution-space pruning. Notably, the proposed reduction framework exhibits significant transfer potential, as its design principles can be seamlessly adapted to other graph layout problems with minimal adaptation.

4.2. Influence of the greedy-optimized perturbation stage

This section conducts a comprehensive analysis of the greedy-optimized perturbation stage's influence on algorithmic performance, specifically examining two key aspects: the holistic impact of the entire perturbation stage and the distinct contribution of the optimized assignment step. To isolate these effects, we developed two algorithmic variants: 'DSHA-NoRecons' which disables the entire perturbation stage, and 'DSHA-OnlyGreedy' which employs the greedy assignment to repair all the vertices in V_{free} while disabling the optimized assignment step. Both variants were executed under identical experimental conditions detailed in Section 3.1, across all 11 benchmark instance sets. The comparative results presented in Table 9 maintain consistent metric definitions with Table 8, facilitating direct component efficacy evaluation.

Table 9: Comparative performance of perturbation stage variants: DSHA-NoRecons (disabled reconstruction), DSHA-OnlyGreedy (greedy-only assignment) and proposed DSHA across 11 benchmark sets. Metrics include solution quality (Obj) and success rate (OPT(%)).

Set	Num.	DSHA-NoRecons		DSHA-OnlyGreedy		DSHA	
		Obj	OPT(%)	Obj	OPT(%)	Obj	OPT(%)
path	24	108.67	50.00	108.71	54.17	**108.75**	58.33
cycle	24	**107.71**	100.00	107.58	87.50	107.58	87.50
mesh	24	270.88	50.00	270.88	50.00	**271.08**	54.17
toroidal	37	238.73	54.05	260.05	72.97	**278.81**	78.38
hypercube	7	**100.00**	100.00	**100.00**	100.00	**100.00**	100.00
cbt	24	152.04	0.00	**180.71**	4.17	180.67	4.17
hamming	24	63.21	0.00	69.71	0.00	**70.50**	0.00
3dmesh	8	164.75	0.00	167.38	0.00	**167.62**	0.00
caterpillar	40	**147.28**	95.00	146.88	75.00	147.03	77.50
Harwell-Boeing	24	72.79	8.33	84.21	37.50	**84.62**	45.83
Sparse-Matrix	20	2039.40	0.00	2139.15	0.00	**2353.80**	0.00
W/M/L		102/137/17		52/195/9			
p-value		9.36×10^{-16}		4.04×10^{-7}			

Table 9 enables systematic component impact analysis through pairwise variant comparisons. Against the DSHA-NoRecons variant, DSHA demonstrates superior solution quality in 8 of 11 instance sets (Obj column), with 102 superior outcomes versus only 17 inferior results. This difference is statistically significant (Wilcoxon signed-rank test, $p = 9.36 \times 10^{-16} < 0.05$). This confirms the perturbation stage's critical role in solution enhancement. Compared to DSHA-OnlyGreedy, DSHA achieves better solution quality in 8 instance sets while matching performance on cycle and hypercube sets. DSHA-OnlyGreedy shows marginal advantage only on cbt instances. Quantitatively, the DSHA with optimized assignment step yields 52 superior solutions and 195 ties against DSHA-OnlyGreedy, with merely 9 inferior outcomes, underscoring its consistent value across diverse problem structures. This difference is also statistically significant, with a p-value of 4.04×10^{-7}. It is worth noting that while the difference in the average objective value (column 'Obj') between DSHA-NoRecons and DSHA is minor, their success rates in attaining the known optimal solutions differ significantly (95% vs. 77.5%). This apparent discrepancy stems from two inherent properties of the AMP. First, the optimal objective value for a given graph is confined to a narrow integer range (from 1 to n - 1). For graphs with dense edge sets, this feasible range is especially constrained. Consequently, competing algorithms often yield final solutions with very similar values, which directly reflects this 'max-min' problem characteristic. Second, although the solution space is vast (containing n!/2 distinct permutations considering symmetry), the number of permutations that share

the same objective value increases substantially as the value itself increases. This means that achieving a higher objective value, even by a single unit, represents a considerably more difficult optimization challenge. Notably, DSHA-NoRecons demonstrates superior performance on the cycle (100% optimal) and caterpillar (95% optimal) instance sets. This result suggests that the current perturbation stage may not be fully suitable to these specific topologies, indicating a clear avenue for improvement. A promising direction for future work is to incorporate graph topology into the vertex selection strategy during the perturbation phase.

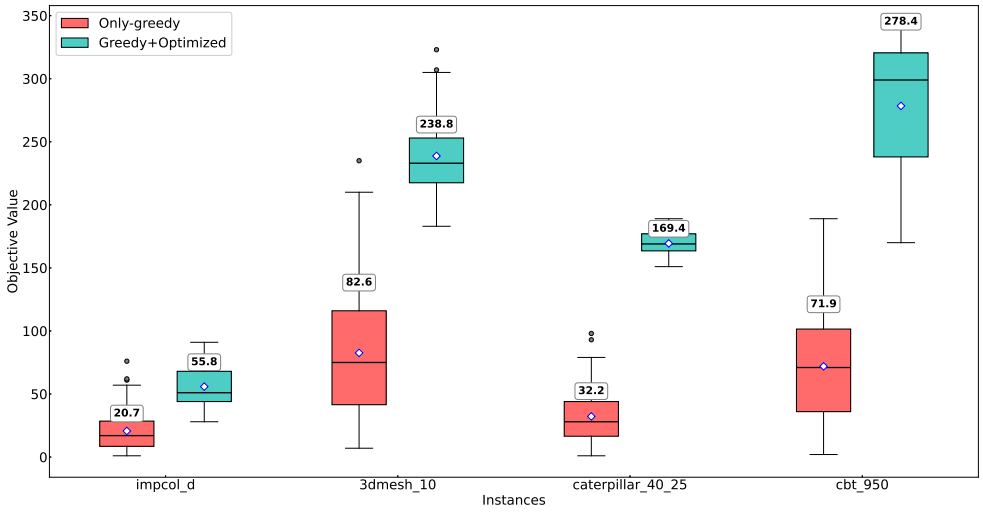


Figure 5: Solution quality distributions after perturbation: Greedy+Optimized vs. Only-Greedy from local optima (4 instances $\times$ 100 runs).

To examine the behavior of the independent set extracted during the perturbation stage, we analyzed its size relative to the total pool of selected vertices across different benchmark sets. For each set, we selected the largest instance for this investigation. As shown in Figure 6, the independent set size (blue bars) and the total number of selected vertices (orange bars) are displayed alongside their ratio (red solid curve). The results indicate that the independent set typically comprises more than 50 vertices (except for can_ _715, which has 36), and the ratio remains above 0.8 for most sets (except for the hamming3x4x4x4x6, where it is 0.65). This consistently high ratio demonstrates that the perturbation stage reliably obtains a substantial independent set, which serves as a high-quality starting point for the subsequent greedy-optimized repair step. This empirical evidence further corroborates the effectiveness of the perturbation mechanism, as discussed in relation to Figure 5. Furthermore, we observed that the sizes of both the independent set and the selected vertex pool remain largely stable throughout the algorithm's execution. This stability is anticipated, as the perturbation stage is activated only after an intensive local optimization phase. The solution entering the first perturbation is already of high quality, typically attaining over 90% of the final objective value.

Since the initial quality is maintained in subsequent iterations, the composition of the critical vertex set, and consequently the size of the selected vertex pool, remains relatively consistent, leading to the observed stability in independent set size across the run. It is important

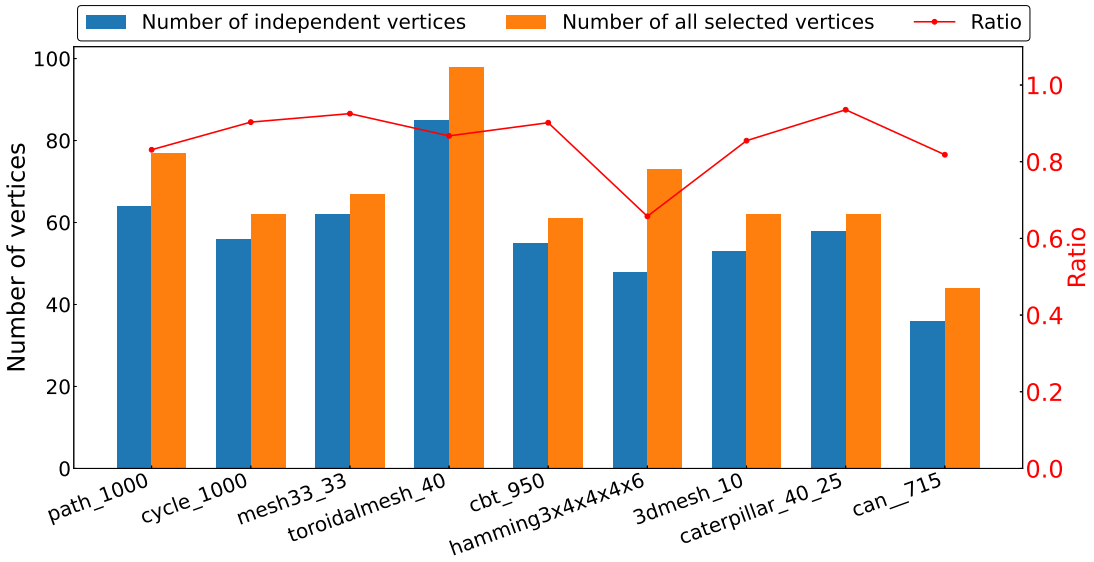


Figure 6: Comparison of independent set size versus selected vertex pool size across different instance types. The blue bars indicate the number of vertices in the independent set, the orange bars represent the total number of selected vertices, and the red solid curve shows the ratio of the independent set size to the selected vertex pool size (right Y-axis). Instance names are shown along the X-axis.

to note that the parameter p is directly linked to the phenomenon described above. This parameter, which ranges from 0.01 to 1.00, determines the fraction of non-critical vertices to be randomly selected for the perturbation stage. The empirically tuned value for our algorithm is $p = 0.06$. This relatively small value reflects a deliberate trade-off between exploration diversity and solution quality preservation. A larger value of p introduces more vertices into the perturbation, which increases the diversity of the search but also elevates the risk of generating a lower-quality intermediate solution. This is because a greater number of vertices then undergo the subsequent, less-controlled greedy reassignment step. An extreme setting of $p = 1.00$ essentially corresponds to a complete reinitialization of the solution. Conversely, a smaller p yields a perturbed solution that is less diversified but typically of higher initial quality. The other extreme, p=0, would limit the perturbation exclusively to the set of critical vertices. Empirically, for most standard instances under 300 nodes, the local optimization procedure reaches near-optimal solutions before the perturbation phase is ever invoked. Given that perturbation is rarely triggered for the majority of our test instances, we adopted a fixed value for p to keep the algorithm simple. As our parameter training prioritized large-scale instances, the final tuned value naturally leans toward a conservative setting (closer to zero),

favoring solution quality and stability over excessive diversification.

Collectively, the experimental results in Table 9, Figure 5 and Figure 6 demonstrate the critical efficacy of the greedy-optimized perturbation stage, particularly its optimized assignment component, in enhancing algorithmic performance. The perturbation framework featuring polynomial-time bottleneck optimization (Section 2.4.3) exhibits significant transfer potential as a general methodology for graph layout problems. This adaptability positions it as a promising foundation for extending research to related domains such as bandwidth minimization and linear arrangement optimization.

5. Conclusions

This work presents a double-stage heuristic algorithm for the antibandwidth maximization problem, called DSHA. Within an iterated local search framework, the method incorporates three core components: a greedy initialization phase, a tabu search mechanism with reduced neighborhood for local refinement, and a greedy-optimized perturbation strategy for diversification. The entire search process is guided by an extended evaluation function. Key contributions also include the formulation of the perturbation stage as a constrained partial antibandwidth maximization problem, along with a proof that it reduces to a linear bottleneck assignment problem over independent sets, thus admitting optimal solutions in polynomial time.

Experimental evaluation on 256 benchmark instances demonstrates the effectiveness of the proposed algorithm. On the first seven instance sets (164 instances) with known optima, it achieves a higher success rate in locating optimal solutions. For the remaining instances (92 instances), the algorithm establishes 42 improved lower bounds and matches 43 best-known results. On the Harwell–Boeing instances specifically, it obtains 8 improved lower bounds, including three substantial improvements ranging from 10.91% to 17.70%. On the larger-scale Sparse-Matrix instances, it delivers strong performance, achieving 14 better results with improvements of up to 42.77% on five challenging cases. Ablation studies confirm the critical roles of both the neighborhood reduction technique and the greedy-optimized perturbation stage used in DSHA.

From the perspective of future work, four directions could be pursued. First, more effective move operators (e.g. rotation and segment-based moves) could further improve the algorithm. Second, because the fixed parameter setting becomes a computational bottleneck on extremely large instances, future work will explore a dynamically adaptive p that scales with instance size or partitions the perturbed node set into smaller subsets. Third, the new lower bounds identified in this work may serve as a foundation for more powerful exact algorithms. Finally, the optimized assignment technique, which reformulates the subproblem as a polynomially solvable linear bottleneck assignment problem, could be extended to other graph

layout problems such as bandwidth minimization (Papadimitriou, 1976) and minimum linear arrangement (Rodriguez-Tello et al., 2008).

Acknowledgment

We are grateful to the reviewers for their useful comments and suggestions which helped us to significantly improve the paper. We would like to thank Professor Jennifer Scott for her valuable support. This work was partially supported by the National Natural Science Foundation of China (No. 72401088, 72301036), the Ministry of Education Humanities and Social Sciences Research Project (No. 24YJC630171) and the Jiangsu Natural Science Foundation (No. BK20241504). Support from the High Performance Computing Platform of Hohai University is also acknowledged.

References

Abreu, A.A.A.M.d., Gonzaga de Oliveira, S.L., 2024. Iterated local search with tabu search for the bandwidth reduction problem in graphs, in: International Conference on Computational Science and Its Applications, Springer. pp. 125–136.

Bansal, R., Srivastava, K., 2010. Memetic algorithm for the antibandwidth maximization problem. Journal of Heuristics 17, 39–60.

Bansal, R., Srivastava, K., 2011. A memetic algorithm for the cyclic antibandwidth maximization problem. Soft Computing 15, 397–412.

Bekos, M.A., Kaufmann, M., Kobourov, S., Veeramoni, S., 2014. A note on maximum differential coloring of planar graphs. Journal of Discrete Algorithms 29, 1–7.

Burkard, R.E., Dollani, H., Lin, Y., Rote, G., 2001. The obnoxious center problem on a tree. SIAM Journal on Discrete Mathematics 14, 498–509.

Calamoneri, T., Massini, A., Török, L., Vrt'o, I., 2009. Antibandwidth of complete k-ary trees. Discrete Mathematics 309, 6408–6414.

Cavero, S., Pardo, E.G., Duarte, A., 2022. A general variable neighborhood search for the cyclic antibandwidth problem. Computational Optimization and Applications 81, 657–687.

Cavero, S., Pardo, E.G., Duarte, A., 2023. Efficient iterated greedy for the two-dimensional bandwidth minimization problem. European Journal of Operational Research 306, 1126–1139.

Dobrev, S., Královič, R., Pardubská, D., Török, L., Vrt'o, I., 2013. Antibandwidth and cyclic antibandwidth of hamming graphs. Discrete Applied Mathematics 161, 1402–1408.

Duarte, A., Martí, R., Resende, M.G., Silva, R.M., 2011. GRASP with path relinking heuristics for the antibandwidth problem. Networks 58, 171–189.

Díaz, J., Petit, J., Serna, M., 2002. A survey of graph layout problems. ACM Computing Surveys 34, 313–356.

Esposito, A., Catalano, M.F., Malucelli, F., Tarricone, L., 1998. A new matrix bandwidth reduction algorithm. Operations Research Letters 23, 99–107.

Fazekas, K., Sinnl, M., Biere, A., Parragh, S., 2020. Duplex encoding of staircase at-most-one constraints for the antibandwidth problem, in: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer. pp. 186–204.

Galinier, P., Boujbel, Z., Coutinho Fernandes, M., 2011. An efficient memetic algorithm for the graph partitioning problem. Annals of Operations Research 191, 1–22.

Garfinkel, R.S., 1971. An improved algorithm for the bottleneck assignment problem. Operations Research 19, 1747–1751.

Glover, F., Rego, C., 2017. New relationships for multi-neighborhood search for the minimum linear arrangement problem. Journal of Discrete Algorithms 46, 16–24.

Hale, W., 1980. Frequency assignment: Theory and applications. Proceedings of the IEEE 68, 1497–1514.

Harper, L.H., 1964. Optimal assignments of numbers to vertices. Journal of the Society for Industrial and Applied Mathematics 12, 131–135.

Hopcroft, J.E., Karp, R.M., 1973. An n^5/2 algorithm for maximum matchings in bipartite graphs. SIAM Journal on Computing 2, 225–231.

Kellerer, H., Wirsching, G., 1998. Bottleneck quadratic assignment problems and the bandwidth problem. Asia-Pacific Journal of Operational Research 15, 169–177.

Leung, J., Vornberger, O., Witthoff, J., 1984. On some variants of the bandwidth minimization problem. SIAM Journal on Computing 13, 650–667.

Lin, Y., Yuan, J., 2003. The dual bandwidth problem for graphs. Journal of Zhengzhou University Natural Science Edition 35, 1–5.

Liu, W.H., Sherman, A.H., 1976. Comparative analysis of the cuthill–mckee and the reverse cuthill–mckee ordering algorithms for sparse matrices. SIAM Journal on Numerical Analysis 13, 198–213.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search, in: Handbook of Metaheuristics. Springer, pp. 320–353.

Lozano, M., Duarte, A., Gortázar, F., Martí, R., 2012. Variable neighborhood search with ejection chains for the antibandwidth problem. Journal of Heuristics 18, 919–938.

Lozano, M., Duarte, A., Gortázar, F., Martí, R., 2013. A hybrid metaheuristic for the cyclic antibandwidth problem. Knowledge-Based Systems 54, 103–113.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

MacFarland, T.W., Yates, J.M., 2016. Friedman twoway analysis of variance (ANOVA) by ranks, in: Introduction to Nonparametric Statistics for the Biological Sciences Using R. Springer International Publishing, Cham, pp. 213–247.

Papadimitriou, C.H., 1976. The NP-completeness of the bandwidth minimization problem. Computing 16, 263–270.

Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., 2017. Improved branch-cut-and-price for capacitated vehicle routing. Mathematical Programming Computation 9, 61–100.

Pferschy, U., 1997. Solution methods and computational investigations for the linear bottleneck assignment problem. Computing 59, 237–258.

Rahaman, M.S., Eshan, T.A., Al Abdullah, S., Rahman, M.S., 2014. Antibandwidth problem for itchy caterpillars, in: 2014 International Conference on Informatics, Electronics & Vision, IEEE. pp. 1–6.

Raspaud, A., Schröder, H., Sýkora, O., Torok, L., Vrt'o, I., 2009. Antibandwidth and cyclic antibandwidth of meshes and hypercubes. Discrete Mathematics 309, 3541–3552.

Ren, J., Hao, J.K., Rodriguez-Tello, E., 2019. An iterated three-phase search approach for solving the cyclic bandwidth problem. IEEE Access 7, 98436–98452.

Ren, J., Hao, J.K., Rodriguez-Tello, E., Li, L., He, K., 2020. A new iterated local search algorithm for the cyclic bandwidth problem. Knowledge-Based Systems 203, 106136.

Rodriguez-Tello, E., Betancourt, L.C., 2011. An improved memetic algorithm for the antibandwidth problem, in: Hao, J., Legrand, P., Collet, P., Monmarché, N., Lutton, E., Schoenauer, M. (Eds.), Artificial Evolution - 10th International Conference, Evolution Artificielle, EA 2011, Angers, France, October 24-26, 2011, Revised Selected Papers, Springer. pp. 121–132.

Rodriguez-Tello, E., Hao, J.K., Torres-Jimenez, J., 2008. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Computers & Operations Research 35, 3331–3346.

Rodriguez-Tello, E., Lardeux, F., Duarte, A., Narvaez-Teran, V., 2019. Alternative evaluation functions for the cyclic bandwidth sum problem. European Journal of Operational Research 273, 904–919.

Rodriguez-Tello, E., Romero-Monsivais, H., Ramirez-Torres, G., Lardeux, F., 2015. Tabu search for the cyclic bandwidth problem. Computers & Operations Research 57, 17–32.

Scott, J., Hu, Y., 2014. Level-based heuristics and hill climbing for the antibandwidth maximization problem. Numerical Linear Algebra with Applications 21, 51–67.

Sinnl, M., 2021. A note on computational approaches for the antibandwidth problem. Central European Journal of Operations Research 29, 1057–1077.

Sundar, S., 2019. A hybrid ant colony optimization approach for the cyclic antibandwidth problem, in: 6th International Conference on Control, Decision and Information Technologies, pp. 1289–1294.

Török, L., Vrt'o, I., 2009. Antibandwidth of d-dimensional meshes, in: International Workshop on Combinatorial Algorithms, Springer. pp. 471–477.

Török, L., Vrt'o, I., 2010. Antibandwidth of three-dimensional meshes. Discrete Mathematics 310, 505–510.

Wang, X., Wu, X., Dumitrescu, S., 2009. On explicit formulas for bandwidth and antibandwidth of hypercubes. Discrete Applied Mathematics 157, 1947–1952.

Wu, Q., Hao, J.K., 2013. Memetic search for the max-bisection problem. Computers & Operations Research 40, 166–179.

Appendix A.

Integer programming formulation of the antibandwidth maximization problem

For completeness, we present the linearized formulation of the antibandwidth maximization problem adopted from Duarte et al. (2011). Let x_{ik} be equal to 1 if vertex i is assigned label k, and 0 otherwise, and let l_i denote the label assigned to vertex i. The variable b represents the antibandwidth, while y_{ij} and z_{ij} are auxiliary binary variables for each edge $\{i, j\} \in E$. The formulation is given as follows.

$$
\begin{aligned}
\max \quad & b \\
\text{subject to} \quad & \sum_{i=1}^{n} x_{ik} = 1, && \forall k = 1, \ldots, n, && \text{(A.1)} \\
& \sum_{k=1}^{n} x_{ik} = 1, && \forall i = 1, \ldots, n, && \text{(A.2)} \\
& \sum_{k=1}^{n} k \cdot x_{ik} = l_i, && \forall i = 1, \ldots, n, && \text{(A.3)} \\
& b - (l_i - l_j) \leq 2(n-1) y_{ij}, && \forall \{i,j\} \in E, && \text{(A.4)} \\
& b + (l_i - l_j) \leq 2(n-1) z_{ij}, && \forall \{i,j\} \in E, && \text{(A.5)} \\
& y_{ij} + z_{ij} = 1, && \forall \{i,j\} \in E, && \text{(A.6)} \\
& b \geq 1, && && \text{(A.7)} \\
& x_{ik} \in \{0,1\}, && \forall i,k = 1, \ldots, n, && \text{(A.8)} \\
& y_{ij}, z_{ij} \in \{0,1\}, && \forall \{i,j\} \in E, && \text{(A.9)} \\
& l_i \in \{1, \ldots, n\}, && \forall i = 1, \ldots, n. && \text{(A.10)}
\end{aligned}
$$

Constraints (A.1) ensure that each label is assigned to exactly one vertex, whereas constraints (A.2) ensure that each vertex receives exactly one label. Constraints (A.3) determine the label l_i assigned to each vertex i. Constraints (A.4)–(A.6) provide a linearization of the absolute-value constraint. Constraint (A.7) guarantees the positivity of b. Therefore, together with the maximization objective, constraints (A.4)–(A.7) imply that $b = \min_{\{i,j\} \in E} |l_i - l_j|$. Finally, constraints (A.8)–(A.10) define the domains of the decision variables.

Appendix B.

Proof of Proposition 1

Proof. **The constrained partial antibandwidth assignment problem (CPAMP).** Formally, given an undirected graph $G = (V, E)$ and a partial solution $\bar{\varphi}$, let V_{fix} be the set of fixed vertices, L_{fix} be the set of the used labels, $V_{free} = V \setminus V_{fix}$ be the set of remaining vertices and L_{free} be the set of unused labels. The edge set E is decomposed into two parts $E_{fix} = \{\{u,v\} \in E \mid u,v \in V_{fix}\}$ and $E_{free} = E \setminus E_{fix}$. The antibandwidth for the edges in E_{free} is defined as $AB(G, E_{free}, \varphi) = \min_{\{u,v\} \in E_{free}} |\varphi(u) - \varphi(v)|$. The objective of the CPAMP is to find an optimal solution φ^* to maximize the antibandwidth for the edges in E_{free}, with respect to the input graph G and the partial solution $\bar{\varphi}$.

The linear bottleneck assignment problem (LBAP). The LBAP (Pferschy, 1997) can be naturally modeled using a complete bipartite graph. Given a complete bipartite graph

$G = (U, V, E)$ with bipartition $U = \{u_1, u_2, \ldots, u_k\}$ and $V = \{v_1, v_2, \ldots, v_k\}$ and a weight function $w : E \to \mathbb{R}$ where $w(u_i, v_j) = c_{ij}$ for each edge $\{u_i, v_j\} \in E$, the LBAP seeks a *perfect matching* $M \subseteq E$ that minimizes the maximum edge weight in the matching:

$$\min_{M \in \mathcal{M}} \max_{\{u_i, v_j\} \in M} w(u_i, v_j), \tag{B.1}$$

where $\mathcal{M}$ is the set of all perfect matchings in G.

Transformation of CPAMP to LBAP. If V_{free} is an independent set, the CPAMP can be transformed into the linear bottleneck assignment problem. We prove this proposition by constructing an instance of LBAP from the CPAMP such that its optimal solution corresponds to an optimal solution for CPAMP.

In our transformation for the CPAMP:

- U represents the set of free vertices V_{free}
- V represents the set of free labels L_{free}
- The edge weight $w(u, \ell)$ encodes the antibandwidth for the vertex $u \in U$ if we assign it with label $\ell \in V$:

$$w(u, \ell) = n + 1 - min_{v \in A(u)} |\ell - \bar{\varphi}(v)|, \tag{B.2}$$

where $v \in A(u)$ depicts the adjacent vertices of u in the original graph of CPAMP and n is the number of vertex in the original graph.

The optimal matching M^* in this bipartite graph directly corresponds to the optimal labeling φ^* for the CPAMP with respect to the partial solution $\bar{\varphi}$. It is worth noting that, all the adjacent vertices of u are already assigned labels in the CPAMP as the vertices in V_{free} are independent. The antibandwidth of u can be exactly determined. That is also the reason why the CPAMP can be transformed into the LBAP only when the V_{free} is an independent set.

Complexity. For LBAP with $|U| = k$, constructing the $k \times k$ cost matrix W requires $O(k^2)$. Solving the LBAP using a threshold algorithm needs a $O(k^3)$ time complexity (Garfinkel, 1971). Therefore, finding the optimal solution φ^* for CPAMP with respect to the independent set V_{free} and the partial solution $\bar{\varphi}$ admits a polynomial algorithm within $O(k^3)$.

Example. A simple illustration is shown in Figure Appendix B.1 and Table Appendix B.1. For the original graph with 10 vertices in Figure Appendix B.1(a), an independent set $V_{free} = \{i, j, g, b, h\}$ is identified and marked in red. And the related labels $l \in L_{free}$ are shown in Figure Appendix B.1(b). When we remove these vertices and the edges, whose endpoints are the vertices in V_{free}, the partial solution $\bar{\varphi}$ is shown in Figure Appendix B.1(c). At last, we transform the CPAMP into the LBAP by constructing U and V with the independent set V_{free} and the label set L_{free}. The weight cost for each edge between U and V is obtained by

Eq. (B.2), and the detail costs are shown in Table Appendix B.1.

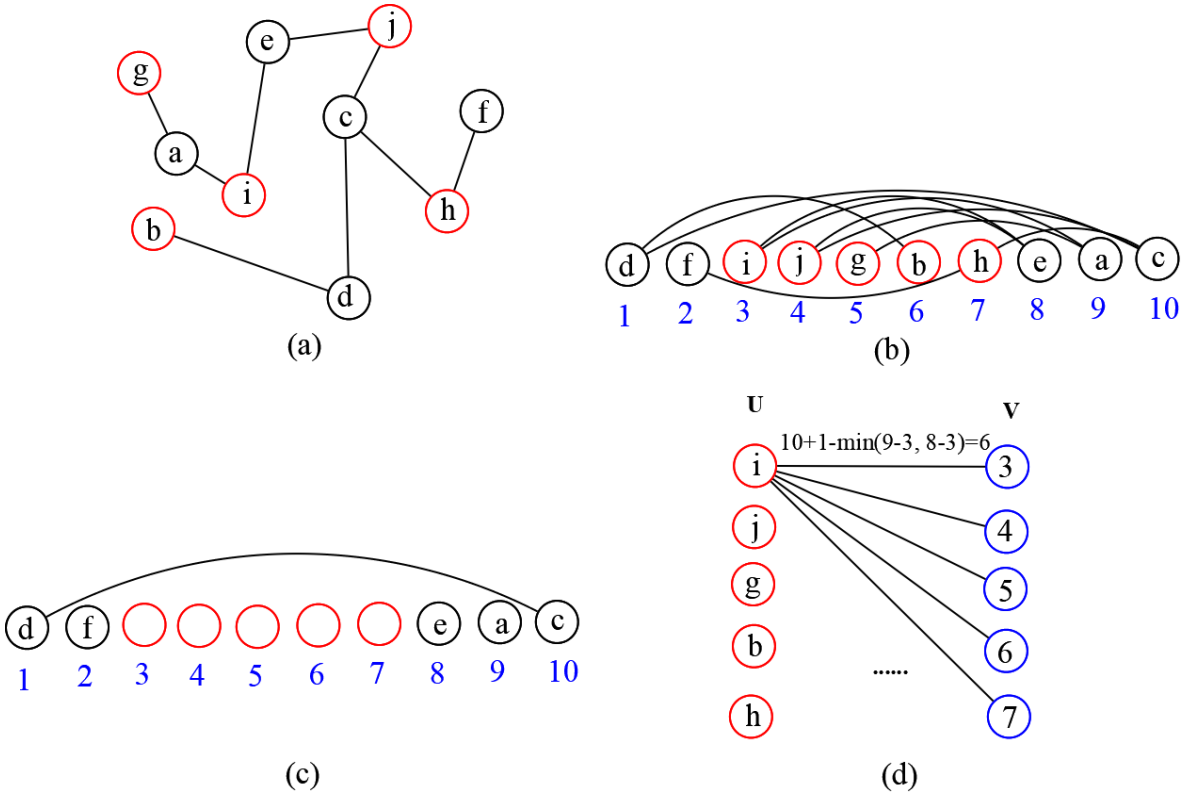


Figure Appendix B.1: An example of the transformation procedure from the CPAMP to the LBAP with an independent set V_{free} over a graph of order $n = 10$. (a) The original input graph of CPAMP with an independent set $V_{free} = \{i, j, g, b, h\}$ marked in red. (b) A complete solution with the set of free labels $L_{free} = \{3, 4, 5, 6, 7\}$. (c) The partial solution $\bar{\varphi}$ with removing every vertex $u \in V_{free}$ as well as the related edges. (d) The converted LBAP in a bipartite graph, where U is the free vertex set V_{free} and V is the free label set L_{free} in the CPAMP. The weight of each edge between U and V can be uniquely determined by Eq. (B.2).

Table Appendix B.1: Weight cost matrix for the LBAP in Figure Appendix B.1 with $U = \{i, j, g, b, h\}$ and $V = \{3, 4, 5, 6, 7\}$: $w(u, l)$ in the matrix ($u \in U, l \in V$) is obtained by Eq. (B.2) (n is 10 because the orignial graph of CPAMP has 10 vertices).

U	V				
	3	4	5	6	7
i	6	7	8	9	10
j	6	7	8	9	10
g	5	6	7	8	9
b	9	8	7	6	5
h	10	9	8	7	8

Threshold algorithm. Algorithm 5 implements the threshold algorithm for our opti-

mized assignment step via a two-phase process. First, it constructs the LBAP bipartite graph $(U_{lbap}, V_{lbap}, E_{lbap})$ from $\bar{\varphi}$, V_{ind}, and L_{free} (line 3). Second, it executes a threshold procedure: after partitioning E_{lbap} into cost-sorted groups $GrpEdge$ (line 4), the algorithm initializes edge length Len, candidate edge set E_p, and termination flag $Flag$. The threshold search proceeds iteratively. Each cycle adds to E_p all edges with cost equal to current Len (line 9), builds bipartite graph $G_p = (U_{lbap}, V_{lbap}, E_p)$ (line 10), and computes a maximum-cardinality matching M using the Hopcroft-Karp algorithm (Hopcroft and Karp, 1973) (line 11). When M becomes a perfect matching, indicating the minimal bottleneck cost is achieved, the loop terminates after current iteration (lines 12-14). Otherwise, Len increments to expand E_p (line 15), continuing until success. The resultant perfect matching $\mathcal{M}$, which minimizes maximum edge cost in the LBAP instance, then repairs $\bar{\varphi}$ into a complete solution φ (line 17). This φ is returned as the optimal assignment.

Algorithm 5 Threshold algorithm

1: **Input**: Input partial solution $\bar{\varphi}$, vertex set V_{ind} and the set of free labels L_{free}
2: **Output**: The repaired solution φ
3: $(U_{lbap}, V_{lbap}, E_{lbap}) \leftarrow ConstructLBAP(\bar{\varphi}, V_{ind}, L_{free})$
4: $GrpEdge \leftarrow PartitionEdge(E_{lbap})$
5: $Len \leftarrow 1$
6: $E_p \leftarrow \emptyset$
7: $Flag \leftarrow True$
8: **while** $Flag$ **do**
9: $\quad E_p \leftarrow AddEdge(GrpEdge, Len)$
10: $\quad G_p \leftarrow ConstructBipartieGraph(U_{lbap}, V_{lbap}, E_p)$
11: $\quad M \leftarrow HopcroftKarpAlgo(G_p)$
12: $\quad$ **if** M is a perfect matching **then**
13: $\quad\quad Flag \leftarrow False$
14: $\quad$ **end if**
15: $\quad Len \leftarrow Len + 1$
16: **end while**
17: $\varphi \leftarrow RepairSolution(M, \bar{\varphi})$
18: **return** φ

Note critically that the algorithm's correctness hinges on two invariants: First, the graph construction ensures $|U_{lbap}| = |V_{lbap}|$ with complete bipartite connectivity, guaranteeing perfect matching existence by combinatorial necessity. Second, processing edges in ascending cost order preserves the equivalence between LBAP's min-max objective and the original CPAMP's

max-min formulation. □