Parallel Discrete Harmony Search Algorithm for the Graph Coloring Problem

Sofiane Chemaa
Mentouri Brothers Constantine 1 University, Algeria | MISC Lab, Constantine 2 University, Algeria
sofiane.chemaa@umc.edu.dz (corresponding author)

Akram Kout
Ferhat Abbas Setif 1 University, Algeria | MISC Lab, Constantine 2 University, Algeria
akram-kout@univ-setif.dz (corresponding author)

Halima Djelloul
Mentouri Brothers Constantine 1 University, Algeria | MISC Lab, Constantine 2 University, Algeria
Halima.djelloul@umc.edu.dz

Nassir Harrag
Mechatronics Laboratory, Ferhat Abbas Setif 1 University, Algeria
nassir.harrag@gmail.com

Received: 30 July 2024 | Revised: 23 August 2024 | Accepted: 3 September 2024

 DOI: https://doi.org/10.48084/etasr.8565

ABSTRACT

Graph coloring is an NP-hard combinatorial optimization problem with significant implications in both theoretical and practical contexts due to its complexity and extensive applicability. In this work, a novel approach is proposed to address the graph coloring problem through a parallel discrete Harmony Search algorithm, termed *PDHSCol*. By harnessing the robustness of the Harmony Search algorithm and integrating parallel processing, our algorithm enhances performance by concurrently generating and evaluating multiple solutions. Implemented in MATLAB, *PDHSCol* was evaluated using a variety of *DIMACS* benchmark instances. The experimental results demonstrate that the algorithm performs effectively and yields promising improvements over various other methods, highlighting its potential to deliver high-quality solutions.

Keywords-graph coloring problem; NP-hard problem; harmony search algorithm; DPHSCol; DIMACS

I. INTRODUCTION

Graph theory is a fundamental discipline focused on the study of graphs, which are mathematical structures used to represent relationships between objects in a pairwise manner. Among the myriad problems in graph theory, the Graph Coloring Problem (GCP) stands out as particularly renowned and challenging. Graph coloring is a combinatorial optimization problem with significant real-world applications, making it an area of substantial interest. Graph coloring can be defined in various forms depending on the constraints applied. These include proper coloring, edge coloring, and total coloring (simultaneous coloring of vertices and edges), among others. These coloring types are utilized to model various practical problems such as frequency allocation [1], scheduling [2], and resource allocation [3]. A proper coloring of a graph is a function that assigns a color to each vertex such that no two adjacent vertices share the same color. The primary objective is to determine the chromatic number of a given graph, which is the minimum number of distinct colors required for such a coloring. Unfortunately, finding the chromatic number is an NP-hard problem, meaning that no polynomial-time algorithm currently exists to optimally color all graphs.

The study of graph coloring has led to the development of numerous resolution methods, which can be broadly categorized into three main types:

- Exact Methods: The fundamental approach for coloring a graph with n vertices is exhaustive search, which involves evaluating all possible color assignments for each vertex. While methods such as those described in [4, 5] can ensure optimal results, their exponential time complexity renders them impractical for large graphs.
- Constructive Methods: Constructive techniques, including DSATUR [6] and W&P [7], color the vertices of a graph sequentially, selecting the vertex that appears most promising according to a predefined criterion at each step.

Although these heuristics are fast and produce reasonably good results, they do not guarantee optimal solutions as the quality of the outcome is highly dependent on the selection criteria for vertices.

- Metaheuristic Methods: Metaheuristics begin with an arbitrarily chosen coloring and aim to improve the current coloring to achieve a better solution. Unlike constructive methods, which start from an uncolored graph and sequentially assign colors, metaheuristics such as genetic algorithms [8], ant colony optimization [9], and Variable Neighborhood Search (VNS) [10] have demonstrated superior performance and high-quality solutions. Their flexibility, adaptability, and efficiency make them promising methodologies for graph coloring problems.

Among metaheuristics, Harmony Search (*HS*), introduced in [11], stands out as one of the most robust methods. Inspired by the natural process of musical performance, where musicians seek a state of harmony, *HS* algorithm has garnered significant attention in optimization research. It is distinguished by its algorithmic simplicity and effective search capabilities. Since its inception, *HS* has been successfully applied to various optimization problems, including Spam Email Detection [12], the Cell Placement Problem [13], data classification [14], multi-UAV task assignment [15], image segmentation [16], and the Orienteering Problem [17]. Its versatility has proven valuable in addressing a broad range of optimization challenges.

In this work, we propose a parallel discrete *HS* algorithm to address the graph coloring problem. While the canonical *HS* was originally designed for continuous optimization, we adapt it to handle the discrete nature of graph coloring. By incorporating parallelism, the proposed approach accelerates computation and enhances the algorithm's ability to find effective solutions across various types of graphs.

II. THE ADOPTED PROPER GCP FORMULATION

The proper GCP involves assigning colors to each vertex such that no two adjacent vertices share the same color. The primary goal is to minimize the total number of colors used to color the graph.

Consider an undirected graph $G = (V, E)$, where V is a finite set of vertices and E is a set of edges, and let T be a given integer. A T-coloring of G is defined by a set $P=\{Col(v_1),\ldots,Col(v_n)\}$, where Col: $V \rightarrow \{1,2,\ldots,T\}$ is the coloring function, $Col(v_i)$ represents the color assigned to vertex v_i, and n denotes the number of vertices in V. If for every edge $\{u,w\} \in E$, $Col(u) \neq Col(w)$, then P is considered a valid T-coloring. Conversely, if there is an edge $\{u,w\} \in E$ where $Col(u)=Col(w)$, P is deemed an invalid T-coloring.

Formally, the Proper *GCP* can be articulated as follows:

For a given T-coloring P, the evaluation function *Fitness* measures the number of conflicting vertices resulting from P. Therefore, *Fitness* is expressed by:

$$Fitness(P) = \sum_{\{u,w\} \in E} \Delta_{uw} \quad (1)$$

where:

$$\Delta_{uw} = \begin{cases} 1, & \text{if } Col(u) = Col(w) \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Consequently, only a coloring P that satisfies $Fitness(P)=0$ is considered valid.

III. HARMONY SEARCH ALGORITHM

The *HS* algorithm is a population-based metaheuristic inspired by the process of musical improvisation. This method is characterized by the interplay of multiple sound waves at different frequencies. Evaluating the improvised harmony relies on aesthetic judgment, which encourages musicians to practice and perfect their coordination. There are notable parallels between musical improvisation and optimization processes. In optimization, the goal is to find the global optimum of an objective function by adjusting a set of decision variables. These variables form a solution vector that is evaluated for quality. The optimization process iteratively updates this vector until the global optimum is achieved [18]. In general, the *HS* algorithm comprises six distinct steps [11, 18]:

1. *Initialization of Problem and Algorithm Parameters:* The initial stage involves setting essential parameters for both the algorithm and the specific problem. This includes defining the objective function and the range of permissible values for the decision variables (Y_i), constrained by lower (Y_{iL}) and upper bounds (Y_{iU}) for each variable (i = 1...D, where D denotes the problem dimension). Additionally, key parameters such as Harmony Memory Size (*HMS*), Harmony Memory Consideration Rate (*HMCR*), Bandwidth (*BW*), Pitch Adjustment Rate (*PAR*), and the number of improvisations (*NbrImp*) are established at this stage.

2. *Harmony Memory Initialization*: This step involves generating initial harmonies within the defined range $[Y_{iL}, Y_{iU}]$ for each decision variable Y_i (i = 1 .. D). The initial harmonies are created with the use of (3):

$$Y_i^j = Y_{iL} + rand() \times (Y_{iU} - Y_{iL}) \quad (3)$$

where j = 1..*HMS*, $rand()$ represents a randomly generated value from a uniform distribution within the range [0, 1].

3. *New Harmony Generation*: In this step, a new harmony, Y^{New}, is generated by applying the core principles of the *HS* algorithm. The new harmony vector, $Y^{New} = (Y_1^{New}, Y_2^{New}, \cdots, Y_D^{New})$, is created following three key rules: memory consideration, random selection, and pitch adjustment. The *HMCR*, ranging between 0 and 1, governs memory consideration by determining the likelihood of selecting a value from the Harmony Memory (*HM*), while 1- *HMCR* represents the probability of randomly selecting a value from the permissible range. To avoid local optima, the pitch adjustment mechanism allows the improvised note to shift within the feasible range, controlled by the *PAR*, which also ranges from 0 to 1. Smaller *PAR* values result in weaker adjustments, whereas larger *PAR* values lead to more significant changes. The Algorithm of

generating a new harmony, often called "improvisation", is shown below. In this context, Y_i^{New} $(i = 1..D)$ represents the *i*th variable of Y^{New}, and Y_i^k $(k = 1..HMS)$ represents the *i*th variable of the *k*th harmony vector in the *HM*. The function $randi(HMS)$ generates a pseudorandom scalar integer between 1 and *HMS* for selecting a harmony vector from the *HM*. If the pitch adjustment mechanism is chosen, *BW* determines the step size, with larger *BW* values increasing the distance between the new value and the *HM* value, balancing global and local search strategies.

```
Algorithm 1: Improvisation
1:For i =1:D
2:  If (rand() ≤ HMCR)
3:    k = randi(HMS)
4:    Y_i^New = Y_i^k
5:    If (rand() ≤ PAR)
6:      Y_i^New = Y_i^New ±rand()×BW
7:    EndIf
8:  Else
9:    Y_i^New = Y_iL +rand() ×(Y_iU −Y_iL)
10: EndIf
11:EndFor
```

4. *HM Update*: Similar to the replacement operator in genetic algorithms, if the fitness of the newly improvised harmony (Y^{New}) is better than the fitness of the worst harmony in the *HM*, the worst harmony is replaced with Y^{New}.

5. *Stopping-Criterion Checking:* If *NbrImp* is reached, the process moves to the final step; otherwise, Steps 3 and 4 are repeated until the stopping criterion is met.

6. *Result Return:* The final step returns the best harmony stored in the *HM* as the optimal solution for the problem.

IV. THE PARALLEL DISCRETE HARMONY SEARCH ALGORITHM FOR GRAPH COLORING

In this section, the proposed approach to graph coloring using a parallel discrete harmony search algorithm, which is named *PDHSCol*, is presented. We will first describe the methods we used for encoding both graphs and coloring solutions. Following this, we will outline the algorithm's architecture and the operations it performs to address the graph coloring problem. This presentation will highlight the specific aspects of our method.

A. Representation and Encoding Method

Data representation and solution encoding are crucial steps in population-based evolutionary algorithms. In our approach, graphs are represented using an adjacency matrix which is a binary $n \times n$ matrix, where n represents the number of vertices in the graph. Each element of the matrix indicates the presence or absence of an edge between two vertices. If an edge exists between vertex u and vertex w, the element (u, w) of the matrix will be 1; otherwise, it will be 0. This representation effectively captures the structure of the graph and facilitates the necessary operations for our algorithm. For example, Figure 1 shows a graph along with its corresponding adjacency matrix.

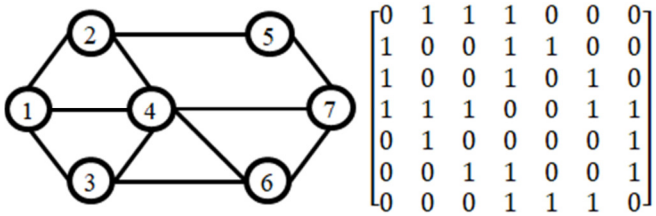

Fig. 1. Graph representation by the adjacency matrix.

In addition to graph representation using adjacency matrices, we also adopted a vector-based represention of coloring solutions. Each solution is represented by an integer vector P of size n, where n is the number of vertices in the graph. In this vector, the value of element $P[i]$ denotes the color assigned to vertex i. An example of this representation is shown in Figure 2, which includes a graph with its nodes colored and the corresponding solution vector. This method provides a clear and direct management of the assigned colors, facilitating the evaluation and optimization of coloring solutions.

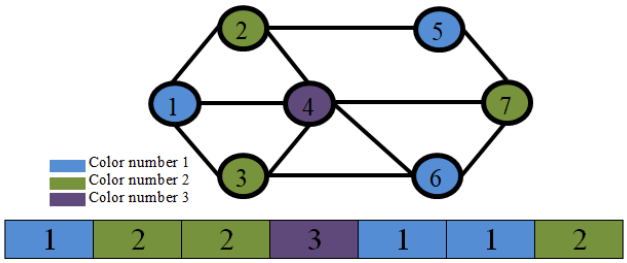

Fig. 2. Graph coloring vector representation.

B. Architectural Framework and Core Processes of the PDHSCol Algorithm

The behavior of the proposed algorithm is illustrated and visualized by the activity diagram presented in Figure 3. *PDHSCol* begins with the initialization of parameters associated with the parallel discrete *HS* algorithm, as well as the initialization of data related to the *GCP*. Following this, *PDHSCol* generates an initial population using a method that will be detailed later. Once the initial population is generated, the best solution is retrieved to evaluate its fitness, which represents the number of conflicts between vertices. The objective is to reduce this fitness to zero. If the fitness of the best solution is greater than zero, our algorithm enters a set of iterations. Each iteration generates *PH* solutions in parallel, where *PH* represents the number of solutions generated simultaneously. This generation follows the improvisation process of our algorithm. At the end of this parallel generation, an update process is executed concurrently. For each generated harmony, the algorithm compares its fitness with that of the existing harmonies in the *HM*. If the fitness of the new harmony is better than that of the least performing harmony, the latter is replaced by the new harmony. Once this step is completed, the algorithm retrieves the best solution in the *HM*. If the fitness of this solution is still greater than zero, the parallel generation process is re-executed. If, at any point, our algorithm finds a solution with a fitness equal to zero, it

displays this solution, representing a valid *T*-coloring. The algorithm then automatically proceeds to search for a (*T*-1)-coloring. If the maximum number of allowed iterations (*NbrImp*) is exceeded, the algorithm terminates its search.

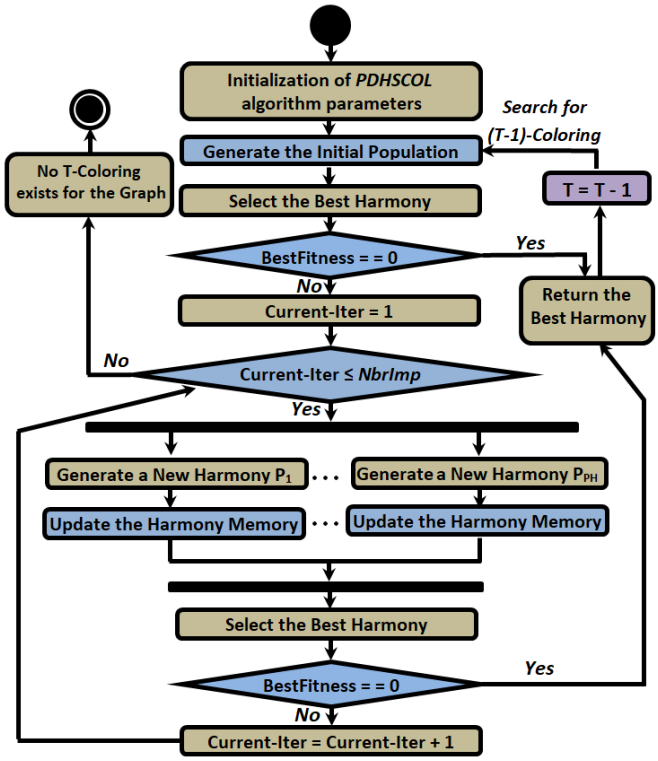

Fig. 3. *PDHSCol* algorithm architecture.

Following the description of the activity diagram, we will provide a thorough explanation of the principal operations executed by the proposed algorithm. This detailed examination will offer a comprehensive understanding of the algorithm's functionality and illustrate how each step contributes to identifying an effective graph coloring solution.

1) Initialization of Algorithm Parameters

This initial operation constitutes a foundational step of the algorithm. We begin by importing the graph to be colored and representing it as an adjacency matrix. Following this, several essential parameters are defined to ensure the proper functioning of the algorithm. These parameters include the number of colors to be used for graph coloring (*T*), *HMS*, *HMCR*, *NbrImp*, and *PAR* with its minimum and maximum values. Additionally, the number of harmonies to be generated in parallel *PH* is specified. For our specific implementation, we have set the following values: *HMS* = 30, *HMCR* = 0.92, *MinPAR* = 0.2, *MaxPAR* = 0.9, and *PH* = 4. This initialization step configures the algorithm and facilitates the transition to the subsequent operations.

2) Generation of the Initial Population

In order to generate the initial population, we adopted a random sequential coloring approach. The technique used allows for creating a set of parallel solutions, significantly accelerating the generation process. Our proposed algorithm is described in detail in the pseudo-code presented in Algorithm 2.

```
Algorithm 2: Parallel Random Sequential
Colorings
Input: A, N, T
Output: Best Pi Solution
1:Initialize PH
2:ParFor i = 1:PH
3:   UVi = randperm(N)
4:   Pi = zeros(1, N)
5:   CCi = 1
6:   While ((UVi ≠ Ø)&&(CCi ≤ T))
7:     v = UVi(1)
8:     Pi(v) = CCi
9:     UVi = UVi \ v
10:    For q = 1:length(UVi)
11:      v = UVi(q)
12:      Coloring = True
13:      For j = 1:N
14:        If ((A(v,j)==1)&&(Pi(j)==CCi))
15:          Coloring = False
16:          Break
17:        EndIf
18:      EndFor
19:      If (Coloring == True)
20:        Pi(v) = CCi
21:        UVi = UVi \ v
22:      EndIf
23:    EndFor
24:    CCi = CCi + 1
25: EndWhile
26: If (UVi ≠ Ø)
27:    For (j = 1:length(UVi))
28:      Pi (UVi(j)) = randi(T),
29:    EndFor
30: EndIf
31:EndParFor
32:Return the Best Pi Coloring
```

To facilitate the understanding of Algorithm 2, the meaning of the variables used is explained below:

- *G*: the graph to be colored, represented by an adjacency matrix *A*.
- *N*: the number of vertices in the graph *G*.
- *T*: the maximum number of colors to be used.
- *PH*: the number of solutions to be generated in parallel.
- P_1, $P_2 \ldots P_{PH}$: the solution vectors to be generated in parallel. Each vector represents a solution with colors assigned to the vertices.
- UV_1, $UV_2 \ldots UV_{PS}$: the sets of uncolored vertices for each solution. These sets are generated with a random order of vertices.

- CC_i: the current color used to generate the solution P_i (i=1: PH).

Conforming to the algorithm, the following steps are undertaken to generate the initial population:

```
Initialization
1. Determine the value of PH.
2. For each parallel solution Pi (i=1:PH):
2.1 Generate the UVi vector containing the uncolored vertices of the graph G in a random order.
2.2 Create the Pi solution vectors of size N, initialized with zero values.
2.3 Initialize CCi with the value 1

Construction of the initial population
1. For each parallel solution Pi (i=1:PH):
1.1 While the UVi vector is not empty and CCi is less than or equal to T:
(a) Take the first vertex v appearing in the UVi vector.
(b) Update the UVi vector and the Pi solution vector by removing vertex v from the UVi vector and assigning the color CCi to vertex v in the Pi vector.
(c) For all remaining vertices in the UVi vector, test the possibility of coloring each vertex using the color CCi, i.e., check if there are no adjacent vertices with the same color. If a vertex can be colored with the color CCi, update the UVi vector and the Pi solution vector.
(d) Move to the next color (CCi = CCi + 1).
1.2 If the UVi vector is not empty, assign a random color between 1 and T to all remaining vertices in UVi.
2. Return the best solution among the solutions generated in parallel.
```

By following this algorithmic process, we obtain a diverse initial population, which reduces the convergence time of the *PDHSCol* algorithm. This diversity allows for a more comprehensive exploration of the solution space, enhancing the algorithm's ability to find high-quality colorings for the given graph.

3) The Improvisation Procedure

The *HS* algorithm has undergone significant enhancements over time, with one of the most robust versions being proposed in [19]. Unlike the traditional approach, which employs fixed values for *PAR* and *BW* and thus requires numerous iterations to reach an optimal solution, the enhanced version dynamically adjusts these parameters. By utilizing dynamic values that evolve with each iteration, this improvement overcomes the limitations of fixed parameters.

In our work, we adopted these dynamic values for *PAR*, thereby enhancing the exploration and exploitation capabilities of the algorithm and leading to more efficient convergence to high-quality solutions.

Regarding discrete versions of the *HS* algorithm proposed in the literature, one notable example is the binary version presented in [20]. In this version, the authors introduced a new Pitch adjustment rule that selects the adjacent value from the structural neighborhood rather than from the *HM*. For ease of implementation, the neighbor of each *HS* vector is defined as the globally optimal harmony vector in the *HM*. Additionally, a new discrete version of the algorithm was proposed in [21]. This version utilizes a Pitch adjustment method to select an adjacent value from the *HM*, based on (1)-order and (-1)-order vectors relative to the selected vectors when harmony consideration is applied. Given the discrete nature of the GCP, we propose a parallel discrete version to effectively address this problem. Algorithm 3 illustrates the pseudocode demonstrating the improvisation method of the proposed *PDHSCol* algorithm. In our algorithm, we introduce a new Pitch adjustment method based on two main techniques. The first technique selects an adjacent value from the *HM* with the condition of minimizing the fitness of the harmony being generated. The second technique, inspired by [20], directly selects the value from the globally optimal harmony vector in the *HM*.

```
Algorithm 3: PDHSCol Improvisation
ParFor i = 1:PH
  Pi = zeros(1,N)
  For j = 1:N
    R1 = rand()
    If (R1 ≤ HMCR)
      Index = randi(HMS)
      Pi(j) = HM(Index,j)
      R2 = rand()
      If (R2 ≤ PAR)
        R3 = rand()
        If (R3 ≤ 0.5)
          For w=1:HMS
            Pi(j) = HM(w,j)
            F = Fitness (Pi)
            If (w == 1)
              BestF = F
              BestIndex =1
            Else
              If (F < BestF)
                BestF = F
                BestIndex = w
              EndIf
            EndIf
          EndFor
          Pi(j) = HM(BestIndex,j)
        Else
          Pi(j) = BestHarmony(j)
        EndIf
      EndIf
    Else
     Pi(j) = randi(T)
  EndFor
EndParFor
```

V. EXPERIMENTAL RESULTS

To implement the proposed approach for graph coloring, we developed the *PDHSCol* algorithm using MATLAB R2023a. MATLAB was selected for its robust parallel computing capabilities, which are facilitated through its parallel pool feature. A parallel pool in MATLAB is a set of workers, either on a compute cluster or a desktop, that can run multiple tasks simultaneously. This feature was crucial for efficiently executing our parallelized algorithm. The program was run on a microcomputer equipped with an Intel® Core™ i5-1145G7 processor, operating at 2.60 GHz with a turbo boost up to 4.40 GHz and 16 GB of RAM.

To demonstrate the effectiveness of the proposed algorithm, we conducted experiments on a diverse set of DIMACS benchmark instances [22]. Our objective was to evaluate the performance of *PDHSCol* in comparison to other graph coloring approaches, namely: the Binary Cuckoo Search Algorithm for the Graph Coloring Problem (*BCSCol*) [23], the Binary Bat Algorithm for the Graph Coloring Problem (*BBCol*) [24], the Fuzzy Logic and Whale Optimization Algorithm (*FWOA*) [25], and the Greedy Graph Coloring Algorithm based on Depth-First Search (*GGCADFS*) [26]. The experimental results, along with those of the comparative methods, are presented in Table I. Notably, Column 4 represents the exact solution for each instance.

TABLE I. RESULTS ON DIMACS BENCHMARK INSTANCES

Instance	|V|	|E|	χ	PDHSCol	[23]	[24]	[25]	[26]
myciel3	11	20	4	4	4	4	4	4
myciel4	23	71	5	5	5	5	5	5
queen5_5	25	160	5	5	5	5	6	5
queen6_6	36	290	7	7	8	8	8	8
myciel5	47	236	6	6	6	6	6	6
queen7_7	49	476	7	7	8	8	7	10
queen8_8	64	728	9	10	10	11	9	11
Huck	74	301	11	11	11	11	11	11
Jean	80	254	10	10	10	10	10	10
David	87	406	11	11	11	11	11	11
myciel6	95	755	7	7	7	7	8	7
games120	120	638	9	9	9	9	9	9
miles250	128	387	8	8	8	8	8	8
miles500	128	1170	20	20	21	21	*	20
miles750	128	2113	31	31	32	32	32	31
Anna	138	493	11	11	11	11	11	11
mulsol.i.1	197	3925	49	49	49	49	49	49
zeroin.i.1	211	4100	49	49	49	49	49	49
fpsol2.i.3	425	8688	30	30	30	30	30	30
fpsol2.i.2	451	8691	30	30	30	30	*	30
fpsol2.i.1	496	11654	65	65	65	65	65	65
Homer	561	1629	13	13	13	13	*	13
2-Insertions-5	597	3936	6	6	6	6	*	6
inithx.i.1	864	18707	54	54	54	54	*	54
3-Insertions-5	1406	9695	6	6	6	6	*	6

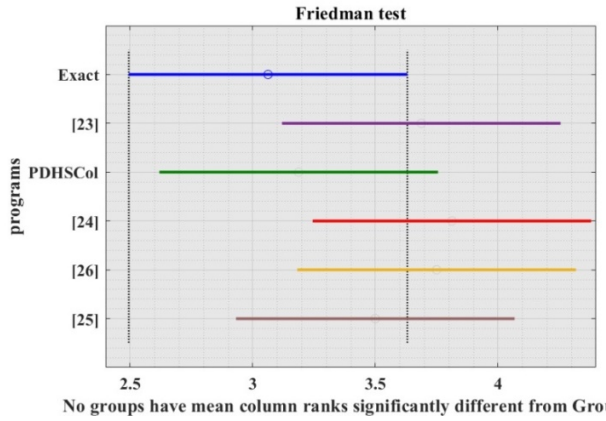

Fig. 4. Friedman test tesult comparison of the proposed *PDHSCol*, BCSCol [23], BBCol [24], FWOA [25], GGCADFS [26] and the exact solutions.

As shown in Table I, *PDHSCol* yields highly favorable results, surpassing the other methods across various instances. Specifically, *PDHSCol* attained optimal colorings in 24 out of 25 instances. The Friedman statistical test, detailed in Figure 4, validates the performance of *PDHSCol*, demonstrating its closeness to exact solutions. Furthermore, statistical analyses reveal significant advantages of *PDHSCol* over competing algorithms. These results highlight the effectiveness and competitiveness of our approach in addressing graph coloring challenges, establishing it as a promising method in this domain.

VI. CONCLUSION

In this work, we introduced a novel approach for graph coloring using a parallel discrete Harmony Search algorithm, named *PDHSCol*. The integration of parallelism within the proposed algorithm enhances performance by accelerating the search process and improving solution quality. The effectiveness of the proposed method is attributed to well-chosen parameters, accurate data representation, and the incorporation of an improved improvisation technique within the Harmony Search algorithm. These factors collectively contribute to achieving effective solutions.

The proposed algorithm, implemented in MATLAB and leveraging its parallel computing capabilities, has been thoroughly tested on a set of DIMACS benchmark instances. The experimental results demonstrate that *PDHSCol* consistently outperforms the four algorithms it was compared against. Specifically, *PDHSCol* frequently provides higher-quality colorings and exhibits enhanced robustness across various instances. These findings underscore the effectiveness of our approach and highlight its competitive advantage in addressing graph coloring problems.

Overall, the proposed method provides a robust and well-adapted solution to the challenges of graph coloring, positioning *PDHSCol* as a promising approach in the field.

REFERENCES

[1] A. Gamst, "Some lower bounds for a class of frequency assignment problems," *IEEE Transactions on Vehicular Technology*, vol. 35, no. 1, pp. 8–14, Oct. 1986, https://doi.org/10.1109/T-VT.1986.24063.

[2] D. de Werra, Ch. Eisenbeis, S. Lelait, and B. Marmol, "On a graph-theoretical model for cyclic register allocation," *Discrete Applied Mathematics*, vol. 93, no. 2, pp. 191–203, Jul. 1999, https://doi.org/10.1016/S0166-218X(99)00105-5.

[3] Y.-M. Chen and W.-C. Wang, "An adaptive rescheduling scheme based heuristic algorithm for cloud services applications," in *International Conference on Machine Learning and Cybernetics*, Guilin, China, Jul. 2011, vol. 3, pp. 961–966, https://doi.org/10.1109/ICMLC.2011.6016893.

[4] I. Mendez-Diaz and P. Zabala, "A Branch-and-Cut algorithm for graph coloring," *Discrete Applied Mathematics*, vol. 154, no. 5, pp. 826–847, Apr. 2006, https://doi.org/10.1016/j.dam.2005.05.022.

[5] A. Mehrotra and M. A. Trick, "A Column Generation Approach for Graph Coloring," *INFORMS Journal on Computing*, vol. 8, no. 4, pp. 344–354, Nov. 1996, https://doi.org/10.1287/ijoc.8.4.344.

[6] D. Brelaz, "New methods to color the vertices of a graph," *Communications of the ACM*, vol. 22, no. 4, pp. 251–256, Dec. 1979, https://doi.org/10.1145/359094.359101.

[7] D. J. A. Welsh and M. B. Powell, "An upper bound for the chromatic number of a graph and its application to timetabling problems," *The Computer Journal*, vol. 10, no. 1, pp. 85–86, Jan. 1967, https://doi.org/10.1093/comjnl/10.1.85.

[8] C. Fleurent and J. A. Ferland, "Genetic and hybrid algorithms for graph coloring," *Annals of Operations Research*, vol. 63, no. 3, pp. 437–461, Jun. 1996, https://doi.org/10.1007/BF02125407.

[9] K. A. Dowsland and J. M. Thompson, "An improved ant colony optimisation heuristic for graph colouring," *Discrete Applied Mathematics*, vol. 156, no. 3, pp. 313–324, Feb. 2008, https://doi.org/10.1016/j.dam.2007.03.025.

[10] C. Avanthay, A. Hertz, and N. Zufferey, "A variable neighborhood search for graph coloring," *European Journal of Operational Research*, vol. 151, no. 2, pp. 379–388, Dec. 2003, https://doi.org/10.1016/S0377-2217(02)00832-9.

[11] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search," *SIMULATION*, vol. 76, no. 2, pp. 60–68, Feb. 2001, https://doi.org/10.1177/003754970107600201.

[12] M. Z. Gashti, "Detection of Spam Email by Combining Harmony Search Algorithm and Decision Tree," *Engineering, Technology & Applied Science Research*, vol. 7, no. 3, pp. 1713–1718, Jun. 2017, https://doi.org/10.48084/etasr.1171.

[13] R. M. A. Qasem and S. M. Massadeh, "Solving Cell Placement Problem Using Harmony Search Algorithms," *Engineering, Technology & Applied Science Research*, vol. 8, no. 4, pp. 3172–3176, Aug. 2018, https://doi.org/10.48084/etasr.2113.

[14] X. Wang, X.-Z. Gao, and S. J. Ovaska, "Fusion of clonal selection algorithm and harmony search method in optimisation of fuzzy classification systems," *International Journal of Bio-Inspired Computation*, vol. 1, no. 1–2, pp. 80–88, Jan. 2009, https://doi.org/10.1504/IJBIC.2009.022776.

[15] Y. Cui, W. Dong, D. Hu, and H. Liu, "The Application of Improved Harmony Search Algorithm to Multi-UAV Task Assignment," *Electronics*, vol. 11, no. 8, Jan. 2022, Art. no. 1171, https://doi.org/10.3390/electronics11081171.

[16] X. Li, X. Li, and G. Yang, "A novelty harmony search algorithm of image segmentation for multilevel thresholding using learning experience and search space constraints," *Multimedia Tools and Applications*, vol. 82, no. 1, pp. 703–723, Jan. 2023, https://doi.org/10.1007/s11042-022-13288-y.

[17] K. Szwarc and U. Boryczka, "A novel approach to the Orienteering Problem based on the Harmony Search algorithm," *PLOS ONE*, vol. 17, no. 2, Feb. 2022, Art. no. e0264584, https://doi.org/10.1371/journal.pone.0264584.

[18] A. Askarzadeh and E. Rashedi, "Harmony search algorithm: Basic concepts and engineering applications," in *Recent Developments in Intelligent Nature-Inspired Computing*, P. Srikanta, Ed. Hershey, PA, USA: IGI Global, 2017, pp. 1–36.

[19] M. Mahdavi, M. Fesanghary, and E. Damangir, "An improved harmony search algorithm for solving optimization problems," *Applied Mathematics and Computation*, vol. 188, no. 2, pp. 1567–1579, May 2007, https://doi.org/10.1016/j.amc.2006.11.033.

[20] L. Wang, R. Yang, Y. Xu, Q. Niu, P. M. Pardalos, and M. Fei, "An improved adaptive binary Harmony Search algorithm," *Information Sciences*, vol. 232, pp. 58–87, May 2013, https://doi.org/10.1016/j.ins.2012.12.043.

[21] K. S. Lee, Z. W. Geem, S. Lee, and K. Bae, "The harmony search heuristic algorithm for discrete structural optimization," *Engineering Optimization*, vol. 37, no. 7, pp. 663–684, Oct. 2005, https://doi.org/10.1080/03052150500211895.

[22] D. S. Johnson and M. A. Trick, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. Providence, Rhode Island: American Mathematical Society, 1996.

[23] H. Djelloul, A. Layeb, and S. Chikhi, "A Binary Cuckoo Search Algorithm for Graph Coloring Problem," *International Journal of Applied Evolutionary Computation*, vol. 5, no. 3, pp. 42–56, Jul. 2014, https://doi.org/10.4018/ijaec.2014070103.

[24] H. Djelloul, S. Sabba, and S. Chikhi, "Binary bat algorithm for graph coloring problem," in *Second World Conference on Complex Systems*, Agadir, Morocco, Nov. 2014, pp. 481–486, https://doi.org/10.1109/ICoCS.2014.7060988.

[25] T. M. Mostafaie, F. Modarres Khiyabani, N. Jafari Navimipour, and B. Daneshian, "A new method for solving of the Graph Coloring Problem based on a fuzzy logic and whale optimization algorithm," *Iranian Journal of Optimization*, vol. 13, no. 2, pp. 115–121, Jun. 2021.

[26] S. Gupta and Singh, "Greedy Graph Coloring Algorithm Based on Depth First Search," *International Journal on Emerging Technologies*, vol. 11, no. 2, pp. 854–862, Mar. 2020.