mathematics

IMPACT FACTOR 2.3

CITESCORE 4.6

Article

Harmony Search Algorithm with Two Problem-Specific Operators for Solving Nonogram Puzzle

Geonhee Lee and Zong Woo Geem

MDPI

https://doi.org/10.3390/math13091470

mathematics

MDPI

Article

Harmony Search Algorithm with Two Problem-Specific Operators for Solving Nonogram Puzzle

Geonhee Lee and Zong Woo Geem *

College of IT Convergence, Gachon University, Seongnam 13120, Republic of Korea; ghlee490@gmail.com
* Correspondence: geem@gachon.ac.kr

Abstract: The nonogram is a logic puzzle where each cell should be colored or left blank according to row and column clues to reveal a hidden picture. This puzzle is known as an NP-complete combinatorial problem characterized by an exponential increase in the number of candidate solutions with increasing puzzle size. So far, some methods have been investigated to address these challenges, including conventional line-solving techniques, integer programming, and neural networks. This study introduces a novel Harmony Search (HS)-based approach for solving nonogram puzzles, incorporating problem-specific operators designed to effectively reduce the solution search space and accelerate convergence. Experimental results obtained from benchmark puzzles demonstrate that the proposed HS model utilizing a clue-constrained random-generation operator significantly reduces the average number of iterations and enhances the solution-finding success rate. Additionally, the HS model integrating an initially confirmed cell-scanning operator exhibited promising performance on specific benchmark problems. The authors think that the nonogram puzzle can be a good benchmark problem for quantum computing-based optimization in the future, and the proposed HS algorithm can also be combined with quantum computing mechanisms.

Keywords: nonogram puzzle; optimization; harmony search

MSC: 05-08

check for updates

Academic Editor: João Nuno Prata

Received: 29 March 2025
Revised: 24 April 2025
Accepted: 26 April 2025
Published: 29 April 2025

Citation: Lee, G.; Geem, Z.W. Harmony Search Algorithm with Two Problem-Specific Operators for Solving Nonogram Puzzle. *Mathematics* **2025**, *13*, 1470. https://doi.org/10.3390/math13091470

1. Introduction

The nonogram is a type of logic puzzle in which the goal is to complete a specific pattern by filling cells based on given row and column clues. This puzzle is known to be an NP-complete problem [1]; thus, as the puzzle size increases, the number of possible patterns grows exponentially, leading to a rapid increase in the computational burden required to find solutions. For instance, a 10×10 puzzle has $2^{100} = 1.27 \times 10^{30}$ candidate solutions, whereas a 15×15 puzzle has $2^{225} = 5.39 \times 10^{67}$ possible solutions, demonstrating an incomparably larger complexity.

Previous research has addressed solving nonogram puzzles through various techniques, including line-solving methods [2], depth-first search algorithms [3], dynamic programming combined with exhaustive search [4], local inference step sequences [5], modified genetic algorithms (CGA and IGA) [6], Taguchi-based genetic algorithms [7], chronological backtracking algorithms [8], integer linear programming (ILP) [9], neural networks [10], and physical card-based zero-knowledge proof protocols [11].

The present study proposes three optimization models based on the Harmony Search (HS) algorithm to solve the nonogram puzzle. The HS algorithm is a phenomenon-mimicking optimization technique inspired by the improvisational process of musicians

creating harmonious notes [12]. It probabilistically explores solution spaces without relying on gradient information, thus providing the advantage of applicability to non-differentiable functions [13]. Moreover, HS is characterized by relatively few algorithm parameters, simple implementation, easy coding, and fast convergence [14], making it suitable for solving diverse engineering, computing, and medical optimization problems [15].

Examples of such applications include steel bridge design [16], urban drainage planning [17], vehicle routing with time windows [18], generative AI-based vehicular networking [19], unmanned surface vessel scheduling [20], forest monitoring [21], energy autonomy of greenhouses [22], large language model (LLM) prompting [23], cryptographing [24], robotics [25], cloud computing-based multi-objective optimization [26], bone age estimation [27], cognitive impairment detection [28], alcoholic EEG signal detection [29], lung nodule detection [30], mammography-based cancer detection [31], and RNA multiple sequence alignment [32]. Also, HS has been applied to recreational problems, including tour routing [33], music composition education [34], fine art volume cognition [35], Tetris puzzles [36], and Sudoku puzzles [37].

The objective of this study is to apply the HS algorithm with a couple of problem-specific operators to the nonogram, providing another way to solve the puzzle. Although it appears to be a recreational mathematical problem, this combinatorial problem might be utilized as a benchmark problem for quantum computing, where each cell (binary variables) can denote a qubit and the relationship between cells can denote quantum entanglement.

The contributions of this study are summarized as follows:

1. **Proposal of Novel HS-based Approaches:**
   This study proposes two novel problem-specific operators tailored to the characteristics of nonogram puzzles, demonstrating the effective applicability of the Harmony Search (HS) algorithm to nonogram-solving.
2. **Initially Confirmed Cell-Scanning Operator:**
   The authors introduce an operator that efficiently reduces the search space and accelerates early-stage convergence by identifying and fixing cells whose states can be definitively confirmed at the initial phase of puzzle-solving.
3. **Clue-Constrained Random Generation Operator:**
   A random generation operator strictly adhering to row clues is proposed, significantly enhancing the validity of candidate solutions. This approach substantially reduces the search space, thereby dramatically decreasing computational burden and iteration counts.
4. **Extensive Experimental Validation:**
   Through extensive experimental evaluations using diverse benchmark nonogram puzzles, the authors validate that the proposed operators effectively improve the performance of the HS algorithm. Particularly, the Clue-Constrained Random Generation operator demonstrates high success rates and rapid convergence, even in complex puzzles.
5. **Benchmark for Quantum Computing Applications:**
   The authors suggest the Nonogram puzzle as a valuable benchmark problem for future quantum computing-based optimization research and propose the potential integration of the developed HS algorithm with quantum computing mechanisms.

2. Problem Definition and Optimization Modeling

2.1. Problem Definition of Nonogram

In a Nonogram puzzle, numerical clues for each row are presented in the leftmost column, referred to as "row clues". Similarly, numerical clues for each column are provided in the topmost row, defined as "column clues".

Row clues are typically separated by spaces. For instance, in Figure 1, the clue for row 1 is represented as "1 1", indicating two separate occurrences of a single black cell. Conversely, row 5 displays the clue "3", indicating one group of three consecutive black cells.

	1	2 1	1	2 1	1
1 1					
1 1					
0					
1 1					
3					

Figure 1. Example of a 5 × 5 size nonogram puzzle.

The clue "1 1" for row 1 implies that within the five-cell row, there must be exactly two black cells, each separated by at least one white cell. Consequently, there are six possible configurations for this scenario, as illustrated in Figure 2.

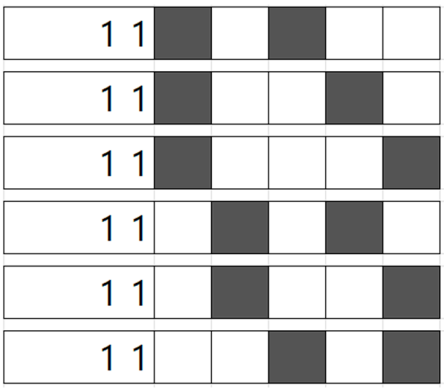

Figure 2. Possible combinations when the row clue is "1 1".

Meanwhile, the clue "3" for row 5 indicates that there must be three consecutive black cells. Therefore, there are three possible configurations, as shown in Figure 3.

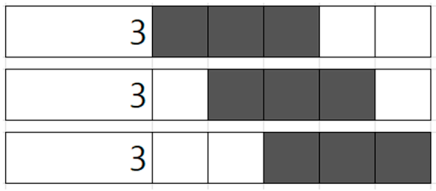

Figure 3. Possible combinations when the row clue is "3".

Examining the six configurations for the clue "1 1", it can be observed that, when superimposing all configurations, the first column has a black cell in only three out of six cases; the second column has a black cell in just two out of six cases; the third column has a black cell in just two out of six cases; the fourth column has a black cell in just two out of

six cases; and the fifth column has a black cell in just three out of six cases. Thus, neither a definite black nor white cell can be confirmed from these cases.

In contrast, when reviewing the three possible configurations for the clue "3", it becomes evident that the third column consistently has a black cell in all configurations. Hence, given the clue "3", the third column can be conclusively determined as black, while the colors of other columns remain uncertain.

In this manner, by iteratively evaluating clues for each row and column and filling in confirmed black or white cells, new information progressively emerges. Consequently, cells that were initially indeterminate with the given clues become gradually resolvable. Repeated application of this evaluation process across all rows and columns ultimately leads to reducing the search space for the nonogram puzzle.

2.2. Optimization Modeling

In order to solve a nonogram using optimization algorithms such as HS, it is necessary to formulate an optimization model consisting of two essential elements: a group of decision variables and an objective function.

Decision variables in this study are defined using a series of binary encoding, representing black cells as 1 and white cells as 0. Each row is represented as one decision variable. For example, if row 1 has the configuration shown in Figure 4, the decision variable has a value of 10100.

Figure 4. Board situation with a decision variable value of 10100.

This study defines the objective function as the number of clue violations. Thus, the smaller the better. This error-scoring function quantifies the difference between the current solution and the given clues. In the situation illustrated in Figure 4, the decision variable is 10100. For scoring, this binary representation must be translated into a clue-like format, indicating the number and sequence of consecutive black cells separated by white cells. The above example translates to "1 1", exactly matching the given clue, thus yielding an error score of 0.

Meanwhile, in the scenario depicted in Figure 5, the given clue is "3", whereas the decision variable is 10110.

Figure 5. Board situation with a decision variable value of 10110.

Converting this variable into clue format results in "1 2". The error calculation compares the clue-based decision variable with the given clue as follows:

1. Calculate the error between the first value "1" of the converted decision variable and the first given clue "3".
2. Calculate the error between the second value "2" of the converted decision variable and the second given clue (0 because there is no second clue).
   Sum these two differences to obtain the total error. Thus, in this example, the total error score is calculated as $|1-3|+|2-0|=4$.

This error calculation is performed separately for each row, from row 1 through the final row n. Similarly, errors are computed and accumulated for each column from column 1 through column m. The final value of the objective function is the sum of all row and column errors. In other words, the objective function measures the deviation between the

provided row and column clues and the current solution after converting the solution into clue-like formats.

Formally, the objective function for an $N \times M$ board is expressed as:

$$\text{Error score} = \sum_{i=1}^{N} \sum_{j=1}^{R_i} \left| ry_{ij} - rx_{ij} \right| + \sum_{i=1}^{M} \sum_{j=1}^{C_i} \left| cy_{ij} - cx_{ij} \right| \quad (1)$$

Here, ry_{ij} denotes the j-th clue provided for the i-th row. For example, if row 3 has clues "3 1", then $ry_{31} = 3$ and $ry_{32} = 1$.

Similarly, rx_{ij} represents the j-th clue derived from the i-th row of the current solution. If the third row's decision variable is 101101, it translates into clues "1 2 1", yielding $rx_{31} = 1$, $rx_{32} = 2$, and $rx_{33} = 1$.

The term R_i is the maximum of the number of clues provided for the i-th row and the number of clues derived from the decision variable. In the above example, the given clue for row 3 ("3 1") consists of a total of two clues, while the decision variable ("1 2 1") has a total of three clues; thus, R_3 is 3, the larger value.

Likewise, cy_{ij} represents the clues given for the i-th column, with i being the column index and j the clue number, analogous to ry_{ij}. Similarly, cx_{ij} represents the clues derived from the current solution, obtained by reading columns in binary code format. The term C_i, analogous to R_i, denotes the larger number of clues between the given column clues and those derived from the current solution.

3. Harmony Search Algorithm and Its Variants for Nonogram

3.1. Overview of the Harmony Search Algorithm

The HS algorithm is a meta-heuristic inspired by the improvisational process of musicians creating harmonious melodies. HS mainly comprises the following three operators:

- Random Selection: Generates entirely new random solutions to increase search diversity.
- Memory Consideration: Selects elements from existing solutions stored in the harmony memory (HM) to create new solutions.
- Pitch Adjustment: Slightly modifies solutions from harmony memory to enhance local search around existing solutions.

The optimization procedure of the HS algorithm is briefly summarized as follows:

1. Initialization (Step 1): Hyperparameters are established, and the harmony memory is initialized, usually with random solutions.
2. Improvisation (Step 2): New solutions are generated to improve harmony memory, probabilistically combining the three main operators (Random Selection, Harmony Memory Consideration, and Pitch Adjustment) based on hyperparameter settings from Step 1.
3. Evaluation (Step 3): The newly generated solutions are evaluated using the objective function, assigning scores to each new harmony.
4. HM update (Step 4): If the new solution's score is better than the lowest-scoring solution currently in harmony memory, the new solution replaces the worst one in the harmony memory.
5. Termination or Repeat (Step 5): If the termination condition (usually a maximum number of iterations) is not satisfied, return to Step 2 and repeat the process of solution generation, evaluation, and harmony memory update. Once the termination condition is satisfied, the best-scoring solution in the harmony memory is returned as the final solution.

3.2. Harmony Search Variants for Nonogram

This study proposes two variants of the HS algorithm, on top of original version [38], to be adapted for the nonogram problem.

3.2.1. Basic Version (Version A): This Version Utilizes Only Three Basic Operators (Random Selection, Memory Consideration, and Pitch Adjustment) of the Original HS Algorithm

- The operator of Random Selection generates random binary sequences of 1 s and 0 s equal to the number of cells per row. For example, for a problem with a column size of 5, a randomly generated solution might look like 11001.
- The operator of Memory Consideration randomly selects one solution from HM and adopts one of its decision variable values for generating a new harmony.
- The operator of Pitch Adjustment slightly modifies a solution selected via HM Consideration by flipping one randomly chosen bit (changing 1 to 0 or 0 to 1).

3.2.2. Initially Confirmed Cell-Scanning Version (Version B)

In Version B, the algorithm initially scans all rows and columns to identify cells that can be definitively confirmed as either black (1) or white (0). These confirmed cells are fixed in advance, after which the HS algorithm applies the three operators described in the basic version.

More specifically, for a puzzle with board size N, having clues $X_1, X_2, \ldots, X_n$ for a given row or column, the number of initially confirmed cells is calculated using the following formula:

$$\sum_{m=1}^{n} \text{Max}\left(X_m - \left(N - (n-1) - \sum_{k=1}^{n} X_k\right), 0\right) \quad (2)$$

Figure 2 is based on the 'single-line solving' method, the most common technique used for solving nonogram puzzles. The 'single-line solving' method, described in Section 3 of reference [39], determines whether a specific cell can be definitively confirmed as black based solely on the clues of a single row (or column). This approach involves comparing the total value of all clues, including mandatory gaps, to the board size.

This is reflected in Equation (2) as $N - (n-1) - \sum_{k=1}^{n} X_k$, where $n-1$ represents the number of mandatory gaps between n clues, and $\sum_{k=1}^{n} X_k$ denotes the sum of all clue values. Thus, subtracting these from the board size N, the value $N - (n-1) - \sum_{k=1}^{n} X_k$ becomes the criterion for judging the existence of cells that can be definitively determined within the line. For instance, if this value is 3, it implies that the number of confirmed cells corresponds to the amount by which each clue exceeds 3.

In other words, Equation (2), expressed as $\text{Max}(X_m - (N - (n-1) - \sum_{k=1}^{n} X_k), 0)$, means comparing each clue value X_m to the value $N - (n-1) - \sum_{k=1}^{n} X_k$; if positive, it indicates that confirmed cells exist within the respective clue. Equation (2), formulated in this way, sums the number of confirmed cells for all clues in the corresponding line.

For example, as illustrated in Figure 6, for a board size $N = 10$ and a single clue $X_1 = 3$, it follows that $\sum_{k=1}^{n} X_k = 3$. Thus, $N - (n-1) - \sum_{k=1}^{n} X_k = 10 - (1-1) - 3 = 7$

Figure 6. Example where the column size is 10 and the clue is given as "3".

After calculating (board size − required spaces − sum of clues), any clue value greater than this result indicates cells that can be definitively confirmed. In this context, if any clue exceeds 7, corresponding cells can be confirmed. However, since the clue value 3 is smaller than 7, no cells can be confirmed in this specific scenario.

This formula intuitively represents the intersection of cells obtained by placing clues without unnecessary spaces both from the far left and from the far right, thereby confirming any overlapping black cells.

Applying this logic to the "3" clue example in Figure 7, filling three cells from the leftmost and rightmost positions results in no overlapping cells; thus, no cells are confirmed.

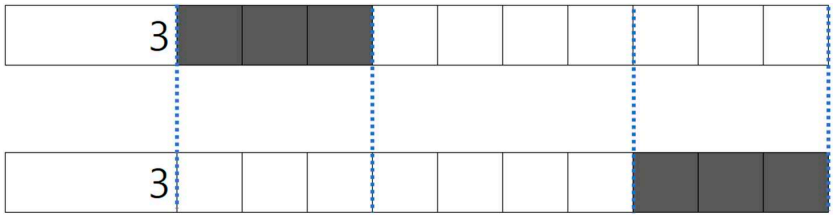

Figure 7. Process of identifying confirmed cells when the board size is 10 and clue is "3".

Another scenario, illustrated in Figure 8, has two clues: "4" and "3".

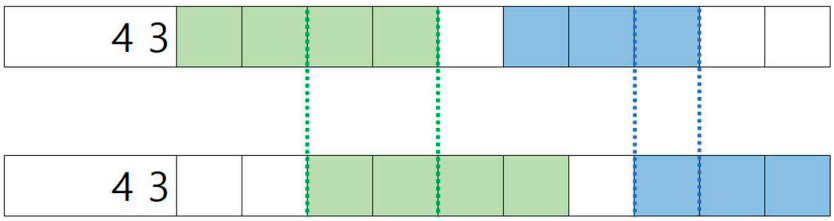

Figure 8. Process of identifying confirmed cells when the board size is 10 and clue is "4 3".

Placing these clues without extra gaps from the leftmost and rightmost positions, respectively, results in overlapping colored cells in columns 3, 4, and 8, confirming these three cells. Substituting this scenario into a part of Equation (2) yields $n = 2$, $X_1 = 4$, $X_2 = 3$: $N - (n-1) - \sum_{k=1}^{n} X_k = 10 - (2-1) - (4+3) = 2$.

Thus, any clue larger than 2 indicates confirmed cells. Here, the confirmed cell count is $(4-2) + (3-2) = 3$, which matches the previously mentioned visual confirmation in Figure 8. Consequently, this reduces the number of combinations generated by random selection from $2^{10} = 1024$ to $2^7 = 128$.

For a larger puzzle example shown in Figure 9 ($N = 15$, $n = 4$, $X_1 = 3$, $X_2 = 1$, $X_3 = 4$, and $X_4 = 2$), it can be visually confirmed that filling from the leftmost and rightmost positions creates overlapping cells in three positions.

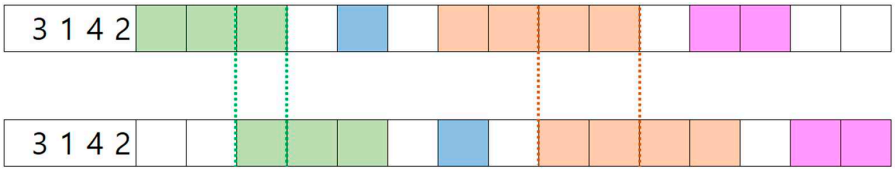

Figure 9. Process of identifying confirmed cells when the board size is 15 and the clue is "3 1 4 2".

Substituting this scenario into part of Equation (2) yields: $N - (n-1) - \sum_{k=1}^{n} X_k = 15 - (4-1) - (3+1+4+2) = 2$. Completing the calculation of the remaining part of Equation (2) yields the following: $\mathrm{Max}(3-2,\ 0) + \mathrm{Max}(1-2,\ 0) + \mathrm{Max}(4-2,\ 0) + \mathrm{Max}(2-2,\ 0) = 3$.

This result matches the visually confirmed cells in Figure 9. Hence, this technique is particularly effective for puzzles with large total clue sums or large individual clue values, as it can significantly reduce the complexity of the solution space.

3.2.3. Clue-Constrained Random Generation Version (Version C)

In Version C, rather than generating entirely random solutions of 1 or 0 as in Versions A and B, the random generation (RG) operator selects solutions that strictly satisfy the

given row clues. The operator randomly selects solutions from combinations that meet the row constraints, effectively treating the row clues as constraints for decision variables.

This approach reduces the computational burden, as generated decision variables inherently satisfy row clues, necessitating the calculation of only column-based error scores. Moreover, by limiting the solution space, it increases the probability of finding the global optimum.

The modified random generation operator performs as follows:

1. Step 1: Initially, cells are filled according to the minimal arrangement dictated by the row clues. For example, given a 10 × 10 puzzle row with clues "3 1 2", the minimal arrangement required to satisfy these clues is shown in Figure 10.

Figure 10. Initial state of the changed RG operator when the clue is "3 1 2".

2. Step 2: Next, the number and potential locations of extra white cells are determined. In the example from Step 1, eight cells are already occupied. Given the row size of 10, there are two remaining extra white cells. Extra white cells must be placed either at the ends of the row or between groups of black cells to maintain clue satisfaction. As illustrated in Figure 11, four potential positions for extra white cells are indicated by green arrows.

Figure 11. Candidate positions where extra white cells can be added when the clue is "3 1 2".

3. Step 3: One of the identified positions from Step 2 is randomly selected to place an extra white cell. For example, if position #3 is chosen, the resulting configuration is updated accordingly, as depicted in Figure 12.

Figure 12. Added an extra white cell in the third position when the clue is "3 1 2".

Since two extra white cells were identified in Step 2, this random selection is performed one more time among positions 1 to 4. If position #1 is selected, the resulting decision variable arrangement is as shown in Figure 13.

Figure 13. Last extra white cell added to the first position when the clue is "3 1 2".

Through this process, decision variables are randomly generated within the constraints set by the given row clues.

Additionally, information about the number and locations of extra white cells is recorded for subsequent use by the pitch adjustment operator. For instance, if Figure 13 corresponds to decision variable x3, then a tracking variable such as x3_extra_blank = [1,0,1,0] would indicate four potential positions, with one extra white cell each in positions one and three.

Similarly, the pitch adjustment operator is modified to ensure continued satisfaction of row clues by applying the following method:

1. Step 1: Extra white cell information is retrieved from the random generation operation. For the example of Figure 13, x3_extra_blank = [1,0,1,0].
2. Step 2: One position among those with a non-zero number of extra white cells is randomly chosen. In this example, positions one and three are eligible. If position three is randomly chosen, one cell is removed from this position, updating the tracking variable to x3_extra_blank = [1,0,0,0].
3. Step 3: Then, another position from the remaining positions (excluding the one just selected) is randomly chosen. If position one is chosen, one extra white cell is added to that position, updating the variable to x3_extra_blank = [2,0,0,0], as illustrated in Figure 14.

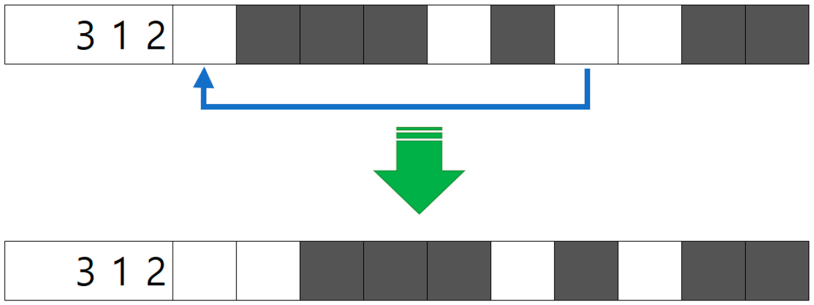

Figure 14. Example where the extra white cell is shifted from position 3 to position 1 through the pitch adjustment operator.

Thus, the pitch adjustment operator can continue making adjustments within the constraint bounds defined by row clues, ensuring that the generated solutions consistently adhere to these clues.

Figure 15 illustrates the number of possible combinations satisfying the clue "3 1 2" on a board of size 10. Specifically, the random generation operator in Version C produces only 10 feasible combinations, representing merely 1% of the total 2^{10} = 1024 combinations that Version A's random generation operator must explore in the same scenario. By significantly reducing the solution space, this approach effectively decreases the iterations required for convergence to the optimal solution.

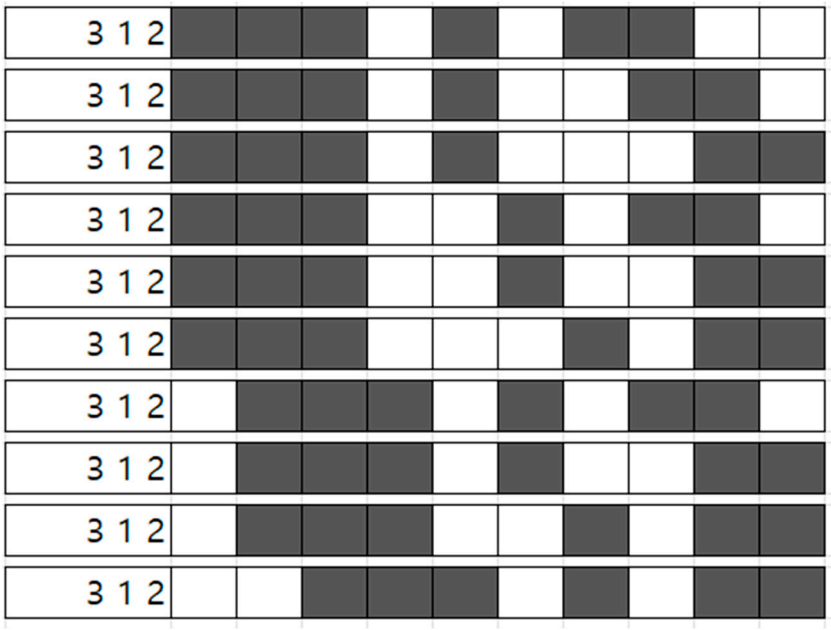

Figure 15. The number of possible combinations satisfying the clue "3 1 2".

4. Experiments and Performance Analysis

The abovementioned HS models were applied to various benchmark nonogram puzzles. As an initial step, relatively simple problems (Group I) were tested, including mine (5×5 size), crane (6×6 size), heart (6×7 size), and horse (8×8 size), as shown in Figure 16.

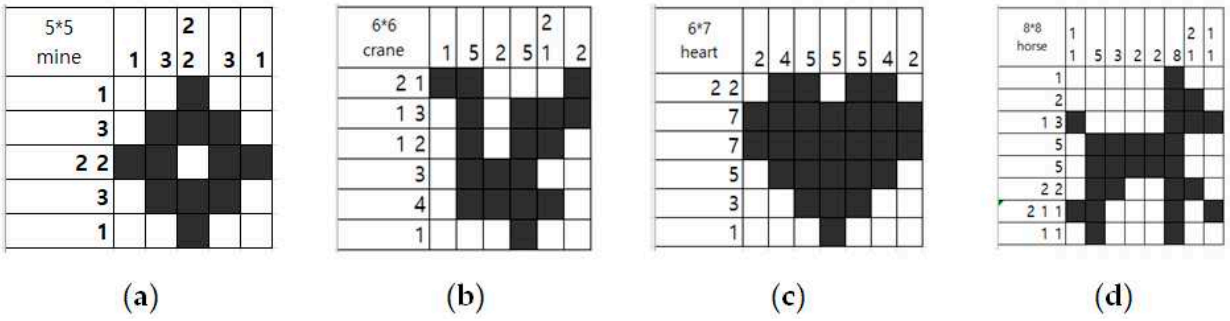

Figure 16. Benchmark Group I: (**a**) mine, (**b**) crane, (**c**) heart, and (**d**) horse.

Experiments were conducted with ten trials for each puzzle in Group I using three HS versions and GA. Detailed computation results are shown in Table 1, which gives the success rate of puzzle solving (success number out of ten trials) and the average number of iterations where HS found the optimal solution (when the objective function value becomes zero). An iteration refers to the entire process of generating a candidate solution and performing a function evaluation on it.

Table 1. Simulation results for Group I.

Puzzle Name	Success Rate (out of 10 Trials)				Average Iteration (Function Evaluation) [Min–Max Range]			
	GA	HS Ver. A	HS Ver. B	HS Ver. C	GA	HS Ver. A	HS Ver. B	HS Ver. C
(a) Mine (5×5)	10/10	10/10	10/10	10/10	2190 [270 ~6810]	3222.2 [574 ~6192]	1.0 [1~1]	35.1 [1~237]
(b) Crane (6×6)	10/10	10/10	10/10	10/10	7935 [2370 ~15,150]	87,376.4 [7792 ~516,640]	6740.5 [564 ~16,877]	3933 [222 ~20,993]
(c) Heart (6×7)	10/10	9/10	10/10	10/10	17,901 [3240 ~39,360]	19,856.2 [6558 ~38,207]	170.7 [4~463]	101 [1~349]
(d) Horse (8×8)	10/10	3/10	4/10	9/10	43,740 [15,240 ~122,520]	500,291.3 [337,257 ~670,754]	172,234 [124,671 ~243,174]	28,121 [2891 ~57,043]

The parameter values used in the simulation are as follows:

For the HS algorithm, the harmony memory size (HMS) was set to 30, the harmony memory consideration rate (HMCR) to 0.95, the pitch adjusting rate (PAR) to 0.7, and the maximum number of iterations to 10^6.

For the Genetic Algorithm (GA), the population size (PS) was set to 30, the crossover probability (Pc) to 1, the mutation probability (Pm) to 0.01, and the maximum number of iterations to 10^6.

For the mine puzzle ($5 * 5$ size, number of cases = $2^{25} = 3.4 * 10^7$), all three versions successfully found the optimal solution without any failure. With respect to average iterations, while Version A took an average of 3222.2 iterations, with a range of a minimum

of 574 iterations and a maximum of 6192 iterations, Version B found the optimal solution from the beginning.

In cases like the *mine* puzzle, where a large number of cells can be definitively determined in the initial stage, the operator used in Version B proves highly effective, even outperforming Version C.

In the *mine* puzzle, Version B is able to confirm 21 out of 25 cells in advance, leaving only 4 cells undetermined. As a result, the number of possible combinations to be searched is reduced to just $2^4 = 16$. This extreme reduction in the solution space enabled the algorithm to identify the optimal solution with high probability among the 30 randomly initialized harmonies, leading to an average of only 1 iteration in Version B, as shown in Table 1.

In contrast, the number of possible combinations generated by Version C is 225, leading to a slightly higher average of 35.1 iterations.

However, as the problem size increases, the number of cells that can be initially confirmed by Version B generally decreases.

For example, in the *crane* puzzle, Version B can initially determine 18 out of 36 cells. Consequently, the number of combinations to be explored is $2^{18} = 262,144$. On the other hand, Version C produces 7776 combinations for the *crane* puzzle, allowing it to achieve a lower average iteration count than Version B.

For the horse puzzle ($8 * 8$ size, number of cases = $2^{64} = 1.8 * 10^{19}$), all three versions found the optimal solution with some failure under the maximum iterations of 10^6. Version A found the optimum three times out of ten, Version B found it four times out of ten, and Version C found it nine times out of ten. Figure 17 shows several local optima in the horse puzzle from Figure 16d.

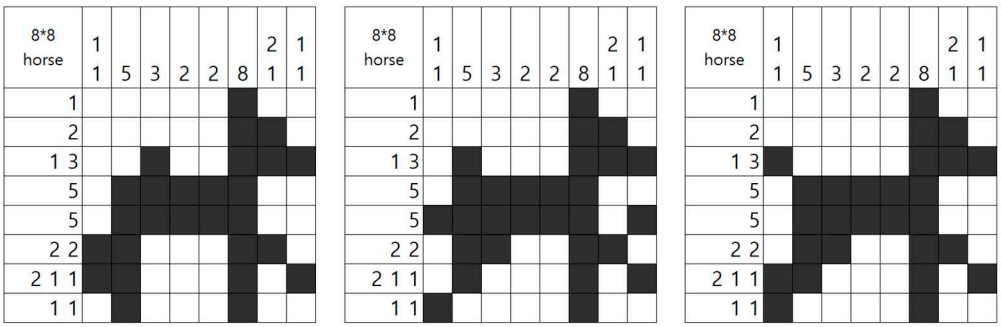

Figure 17. Local optimum cases in the horse benchmark puzzle.

With respect to average iterations, Version A took an average of 500,291.3 iterations, with a range from a minimum of 337,257 iterations to a maximum of 670,754 iterations; Version B took an average of 172,234 iterations, with a range from a minimum of 124,671 iterations to a maximum of 243,174 iterations; and Version C took an average of 28,121 iterations, with a range of a minimum of 2891 iterations to a maximum of 57,043 iterations.

Figure 18 shows convergence curves illustrating the average error score over five simulations for each algorithm version, while Figure 19 magnifies the early stages of Figure 18.

The original HS algorithm (Version A) required significantly more iterations to reach a local optimum, with an error score between 2 and 4, compared to Versions B and C, as shown in Figure 19. This indicates that Version B, utilizing the initially confirmed cell-scanning method, and Version C, employing clue-constrained random generation, effectively narrowed the search space in the early stage. However, Version B often failed to

escape local optima, with error scores of 2–4, whereas Version C quickly converged to the global optimum, typically within 100,000 iterations, as demonstrated in Figure 18.

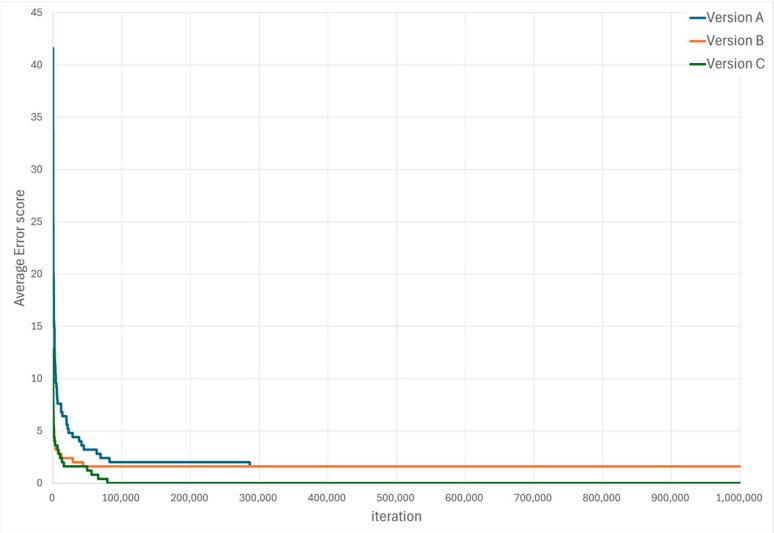

Figure 18. Convergence curves of three versions of the HS algorithm.

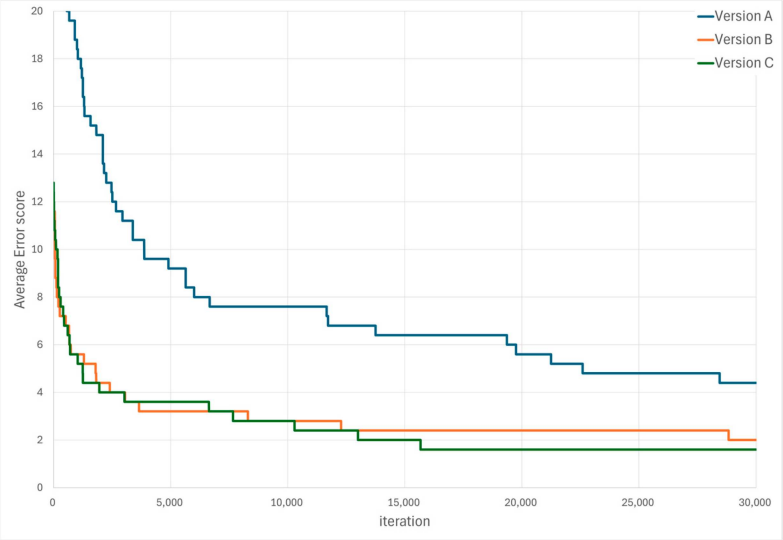

Figure 19. Convergence curves of three versions of the HS algorithm (early-stage focus).

This suggests that the initially confirmed cell-scanning operator (Version B) is effective at reducing the search space at the early stage but becomes less impactful once harmony memory is filled with relatively good solutions.

In contrast, the clue-based generation and pitch adjustment of Version C continued to effectively refine solutions even at later stages, demonstrating superior overall performance. We further applied the HS models to more complex problems (Group II) such as (a) musical note I, (b) musical note II, (c) flower word, (d) leaf, (e) dog, (f) mushroom cloud, (g) tea, (h) TV, (i) snail, and (j) scorpion, as shown in Figure 20. All puzzles have the same size ($10 * 10$ size $= 2^{100} = 1.3 * 10^{30}$). Detailed computation results are shown in Table 2.

The experimental results demonstrated the effectiveness of the proposed HS-based solvers across various benchmark puzzles. In particular, the Clue-Constrained Random Generation method (Version C) showed the best overall performance, while the Initially Confirmed Cell-Scanning method (Version B) also demonstrated good results.

For certain puzzles in Group II—(b) musical note II, (e) dog, (h) TV, and (i) snail—the results are analyzed in detail as follows:

1. Improvement in Success Rate
   - For small puzzles (less than 10×10), all algorithm versions (A, B, C) showed high success rates, with Version C consistently maintaining the highest performance.
   - For 10×10 puzzles, the original HS algorithm (Version A) rarely found solutions. However, the solving capability significantly improved for Versions B and C. Specifically, Version C successfully solved some challenging puzzles completely.
2. Reduction in Average Iterations
   - Version B showed reduced computational burden and improved convergence speed by pre-determining confirmed cells, effectively shrinking the search space.
   - Version C drastically reduced the search space by constraining random generation within the row clues, significantly reducing the average number of iterations.
   - While the performance of Versions B and C was similar for easier puzzles, Version C showed superior performance for puzzles of higher difficulty.

Through the above analysis, it is confirmed that applying improved operators tailored to the specific characteristics of the nonogram problem substantially enhances the algorithm's performance compared to using the basic HS algorithm alone. Particularly, pre-determining confirmed cells and generating solutions constrained by row clues significantly reduces search space and computational burden.

In Table 1, GA achieved a 100% success rate for all four problems; however, its average iteration count was consistently worse than Version C, better than Version A, and generally worse than Version B, except for the horse problem, where GA performed better than Version B.

As shown in Table 2, GA exhibited a 0% success rate across all ten problems, performing worse than all versions of HS. This suggests that GA is more sensitive to problem size.

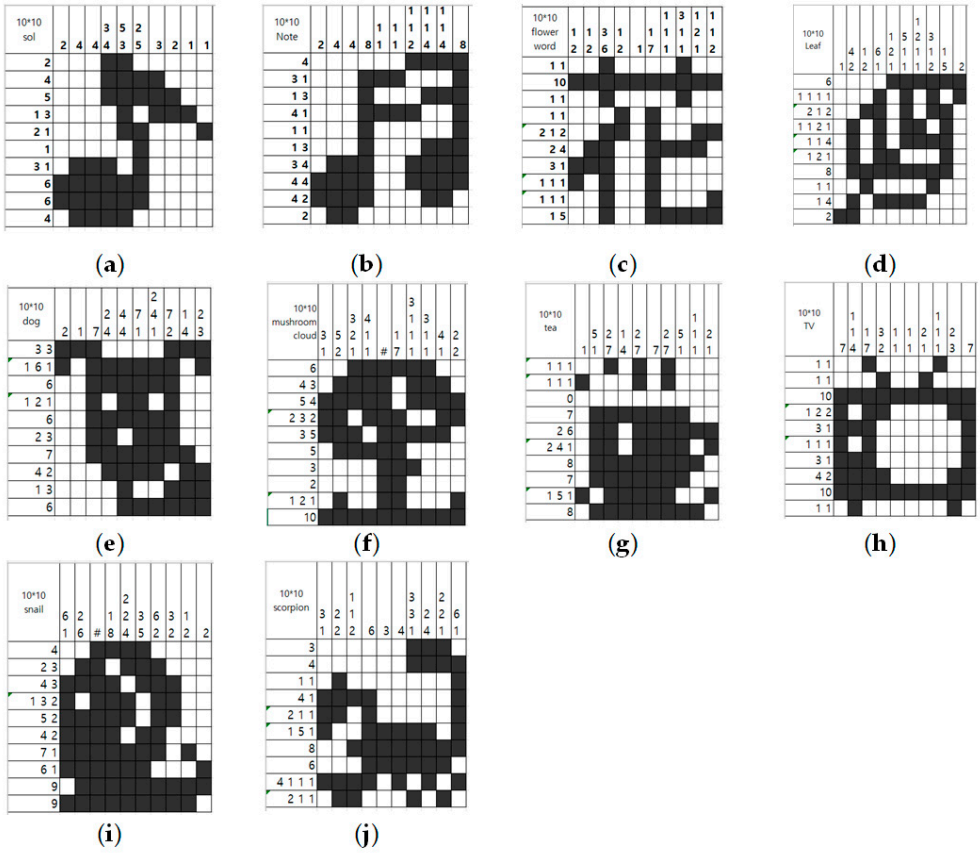

Figure 20. Benchmark Group II: (**a**) musical note I, (**b**) musical note II, (**c**) flower word, (**d**) leaf, (**e**) dog, (**f**) mushroom cloud, (**g**) tea, (**h**) TV, (**i**) snail, (**j**) scorpion.

Table 2. Simulation results for Group II.

Puzzle Name	Success Rate (out of 10 Trials)				Average Iteration (Function Evaluation) [Min–Max Range]			
	GA	HS Ver. A	HS Ver. B	HS Ver. C	GA	HS Ver. A	HS Ver. B	HS Ver. C
(a) Sol	0/10	0/10	0/10	1/10	-	-	-	385,592 [385,592 ~385,592]
(b) Note	0/10	0/10	10/10	10/10	-	-	11,467.9 [5874 ~17,724]	170,829.7 [28,453 ~426,283]
(c) Flower word	0/10	0/10	0/10	1/10	-	-	-	122,062 [122,062 ~122,062]
(d) Leaf	0/10	0/10	0/10	3/10	-	-	-	432,767.3 [314,904 ~585,708]
(e) Dog	0/10	1/10	10/10	10/10	-	798,079 [798,079 ~798,079]	13,062.1 [5874 ~17,724]	100,865.4 [29,212 ~205,127]
(f) Mushroom cloud	0/10	1/10	0/10	10/10	-	775,262 [775,262 ~775,262]	-	20,087.7 [248 ~59,309]
(g) Tea	0/10	2/10	0/10	6/10	-	603,041 [585,912 ~620,170]	-	81,204.2 [30,448 ~173,770]
(h) TV	0/10	1/10	9/10	5/10	-	771,885 [771,885 ~771,885]	110,705.6 [48,405 ~240,562]	143,291.2 [83,907 ~209,201]
(i) Snail	0/10	0/10	10/10	10/10	-	-	36,689.5 [5691 ~179,136]	124,208.2 [13,839 ~285,882]
(j) Scorpion	0/10	0/10	0/10	8/10	-	-	-	122,928.1 [36,311 ~282,549]

5. Conclusions and Future Work

This study proposed an HS algorithm-based approach for solving nonogram puzzles modeled as optimization problems. Experimental findings indicated notable limitations with the basic HS algorithm (Version A), specifically high failure rates and significantly large average iteration counts when tackling difficult puzzles. To address these issues, improved problem-specific operators were introduced. The Initially Confirmed Cell-Scanning method (Version B) effectively reduced the computational burden by pre-filling confirmed cells, substantially decreasing the search space. Moreover, the Clue-Constrained Random Generation method (Version C) optimized the search space by generating solutions strictly within row clue constraints, thereby significantly accelerating convergence to optimal solutions. Performance verification of these proposed methods highlighted the superior capabilities of Version C, which demonstrated improved solving rates and significantly reduced average iteration counts. Particularly in challenging puzzle scenarios, Version C substantially enhanced the success rate, underscoring its effectiveness relative to previous methods. These results confirm the effectiveness of modeling nonogram puzzles as optimization problems and adapting the HS algorithm with problem-specific characteristics.

This study provides fundamental insights into applying the HS algorithm to nonogram puzzles. Future research could explore integrating HS with other optimization methods to further enhance solving performance and effectively address local optima issues. Additionally, developing novel operators with more sophisticated heuristics or leveraging

puzzle-specific features such as symmetry could improve algorithmic efficiency. Finally, investigating quantum computing-based approaches could offer substantial advantages due to quantum computing's inherent parallel processing capabilities, which are particularly well-suited for complex combinatorial optimization problems like nonograms.

Author Contributions: Conceptualization, G.L. and Z.W.G.; methodology, G.L.; software, G.L.; validation, G.L. and Z.W.G.; formal analysis, G.L.; data curation, G.L. and Z.W.G.; writing—original draft preparation, G.L. and Z.W.G.; writing—review and editing, Z.W.G.; visualization, G.L.; supervision, Z.W.G.; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Korea Institute of Energy Technology Evaluation and Planning (KETEP) and the Ministry of Trade, Industry, and Energy, Republic of Korea (RS-2024-00441420; RS-2024-00442817).

Data Availability Statement: Dataset available upon request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ueda, N.; Nagao, T. *NP-Completeness Results for Nonogram via Parsimonious Reductions*; Tokyo Institute of Technology: Tokyo, Japan, 1996; pp. 1–8.
2. Batenburg, K.J.; Kosters, W.A. Solving Nonograms by combining relaxations. *Pattern Recognit.* **2009**, *42*, 1672–1683. [CrossRef]
3. Więckowski, J.; Shekhovtsov, A. Algorithms Effectiveness comparison in solving Nonogram boards. *Procedia Comput. Sci.* **2021**, *192*, 1885–1893. [CrossRef]
4. Wu, I.C.; Sun, D.J.; Chen, L.P.; Chen, K.Y.; Kuo, C.H.; Kang, H.H.; Lin, H.H. An efficient approach to solving nonograms. *IEEE Trans. Comput. Intell. AI Games* **2013**, *5*, 251–264. [CrossRef]
5. Batenburg, K.J.; Henstra, S.; Kosters, W.A.; Palenstijn, W.J. Constructing simple nonograms of varying difficulty. *Pure Math. Appl.* **2009**, *20*, 1–15.
6. Tsai, J.T.; Chou, P.Y.; Fang, J.C. Learning intelligent genetic algorithms using Japanese nonograms. *IEEE Trans. Educ.* **2011**, *55*, 164–168. [CrossRef]
7. Tsai, J.T. Solving Japanese nonograms by Taguchi-based genetic algorithm. *Appl. Intell.* **2012**, *37*, 405–419. [CrossRef]
8. Yu, C.H.; Lee, H.L.; Chen, L.H. An efficient algorithm for solving nonograms. *Appl. Intell.* **2011**, *35*, 18–31. [CrossRef]
9. Khan, K.A. Solving nonograms using integer programming without coloring. *IEEE Trans. Games* **2020**, *14*, 56–63. [CrossRef]
10. Buades Rubio, J.M.; Jaume-i-Capó, A.; López González, D.; Moyà Alcover, G. Solving nonograms using neural networks. *Entertain. Comput.* **2024**, *50*, 100652. [CrossRef]
11. Ruangwises, S. An improved physical ZKP for Nonogram and Nonogram color. *J. Comb. Optim.* **2023**, *45*, 122. [CrossRef]
12. Geem, Z.W.; Kim, J.H.; Loganathan, G.V. A New Heuristic Optimization Algorithm: Harmony Search. *Simulation* **2001**, *76*, 60–68. [CrossRef]
13. Geem, Z.W. Novel derivative of harmony search algorithm for discrete design variables. *Appl. Math. Comput.* **2008**, *199*, 223–230. [CrossRef]
14. Saka, M.P.; Hasançebi, O.; Geem, Z.W. Metaheuristics in structural optimization and discussions on harmony search algorithm. *Swarm Evol. Comput.* **2016**, *28*, 88–97. [CrossRef]
15. Manjarres, D.; Landa-Torres, I.; Gil-Lopez, S.; Del Ser, J.; Bilbao, M.N.; Salcedo-Sanz, S.; Geem, Z.W. A survey on applications of the harmony search algorithm. *Eng. Appl. Artif. Intell.* **2013**, *26*, 1818–1831. [CrossRef]
16. Kwon, T.Y.; Ma, S.S.; Huh, J.; Ahn, J.H. Optimizing steel arch bridge components using multi-objective harmony search. *Structures* **2025**, *73*, 108389. [CrossRef]
17. Aderyani, F.R.; Mousavi, S.J. Machine learning-based rainfall forecasting in real-time optimal operation of urban drainage systems. *J. Hydrol.* **2024**, *645*, 132118.
18. Zhang, Y.; Li, J. A Hybrid Heuristic Harmony Search Algorithm for the Vehicle Routing Problem with Time Windows. *IEEE Access* **2024**, *12*, 42083–42095. [CrossRef]
19. Xie, G.; Xiong, Z.; Zhang, X.; Xie, R.; Guo, S.; Guizani, M. GAI-IoV: Bridging Generative AI and Vehicular Networks for Ubiquitous Edge Intelligence. *IEEE Trans. Wirel. Commun.* **2024**, *23*, 12799–12814. [CrossRef]
20. Tang, W.; Gao, K.; Ma, Z.; Lin, Z.; Yu, H.; Huang, W.; Wu, N. Local search-based meta-heuristics combined with an improved K-Means++ clustering algorithm for unmanned surface vessel scheduling. *Int. J. Prod. Res.* **2025**. [CrossRef]
21. Etaati, A.; Bastam, M.; Ataie, E. Smart forest monitoring: A novel Internet of Things framework with shortest path routing for sustainable environmental management. *IET Netw.* **2024**, *13*, 528–545. [CrossRef]

22. Gholami, M.; Arefi, A.; Hasan, A.; Li, C.; Muyeen, S.M. Enhancing energy autonomy of greenhouses with semi-transparent photovoltaic systems through a comparative study of battery storage systems. *Sci. Rep.* **2025**, *15*, 2213. [CrossRef] [PubMed]
23. Pan, R. Plum: Prompt Learning Using Metaheuristics. Mater's Thesis, Hong Kong University of Science and Technology, Hong Kong, China, 2024.
24. Mitra, S.; Mahapatra, G.; Balas, V.E.; Chattaraj, R. Public Key Cryptography Using Harmony Search Algorithm. In *Innovations in Infrastructure*; Deb, D., Balas, V., Dey, R., Eds.; Springer: Singapore, 2019. [CrossRef]
25. Mokhtari, M.; Taghizadeh, M.; Mazare, M. Impedance control based on optimal adaptive high order super twisting sliding mode for a 7-DOF lower limb exoskeleton. *Meccanica* **2021**, *56*, 535–548. [CrossRef]
26. Li, W.; Du, W.; Tang, W.; Pan, Y.; Zhou, J.; Lin, Z. Parallel algorithm of multiobjective optimization harmony search based on cloud computing. *J. Algorithms Comput. Technol.* **2017**, *11*, 301–313. [CrossRef]
27. Sharma, P. Bone age estimation with HS-optimized Resnet and Yolo for child growth disorder. *Expert Syst. Appl.* **2025**, *259*, 125160. [CrossRef]
28. Li, A.; Li, J.; Hu, Y.; Geng, Y.; Qiang, Y.; Zhao, J. A Dynamic Adaptive Ensemble Learning Framework for Noninvasive Mild Cognitive Impairment Detection: Development and Validation Study. *JMIR Med. Inform.* **2025**, *13*, e60250. [CrossRef]
29. Manivannan, G.S.; Mani, K.; Rajaguru, H.; Talawar, S.V. Detection of Alcoholic EEG signal using LASSO regression with metaheuristics algorithms based LSTM and enhanced artificial neural network classification algorithms. *Sci. Rep.* **2024**, *14*, 21437. [CrossRef]
30. Zamanidoost, Y.; Ould-Bachir, T.; Martel, S. OMS-CNN: Optimized Multi-Scale CNN for Lung Nodule Detection Based on Faster R-CNN. *IEEE J. Biomed. Health Inform.* **2025**, *29*, 2148–2160. [CrossRef]
31. Kumar, M.G.; Kocharla, S.; Yaswanth, N.; Swamy, T.V.N.; Prasad, U.; Vamsee, T. EHA-LNN: Optimized light gradient-boosting machine enabled neural network for cancer detection using mammography. *Biomed. Signal Process. Control* **2025**, *105*, 107540.
32. Saif, M.; Abdullah, R.; Adib, M.; Ahmed, A.A.; Omar, N.A.; Mostafa, S.A. Analysis of Objective Functions for Ribonucleic Acid Multiple Sequence Alignment Fusion Based on Harmony Search Algorithm. *Fusion Pract. Appl.* **2025**, *17*, 1–10.
33. Geem, Z.W.; Tseng, C.L.; Park, Y. Harmony Search for Generalized Orienteering Problem: Best Touring in China. *Lect. Notes Comput. Sci.* **2005**, *3612*, 741–750.
34. Navarro, M.; Corchado, J.M.; Demazeau, Y. MUSIC-MAS: Modeling a harmonic composition system with virtual organizations to assist novice composers. *Expert Syst. Appl.* **2016**, *57*, 345–355. [CrossRef]
35. Koenderink, J.; van Doorn, A.; Wagemans, J. Picasso in the mind's eye of the beholder: Three-dimensional filling-in of ambiguous line drawings. *Cognition* **2012**, *125*, 394–412. [CrossRef] [PubMed]
36. Romero, V.M.; Tomes, L.L.; Yusiong, J.P.T. Tetris Agent Optimization Using Harmony Search Algorithm. *Int. J. Comput. Sci.* **2011**, *8*, 22–31.
37. Geem, Z.W. Harmony Search Algorithm for Solving Sudoku. *Lect. Notes Comput. Sci.* **2007**, *4692*, 371–378.
38. Lee, G.H.; Geem, Z.W. Harmony Search for Soving Nonogram Puzzle. *J. Korean Inst. Intell. Syst.* **2021**, *31*, 422–428.
39. Batenburg, K.J.; Kosters, W.A. A reasoning framework for solving Nonograms. In *International Workshop on Combinatorial Image Analysis*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 372–383.