An Algorithm Inspired by the Ant Colony System for Solving N × N Sliding Puzzles

Bohdan Kostyk[1]

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice, Poland

Abstract

This paper presents an Ant Colony System (ACS)-inspired algorithm designed to navigate the massive state spaces of $n \times n$ sliding puzzles. The proposed method drives the colony's intelligence using a heuristic function combining Manhattan distance and linear conflict, supported by a novel relation-based pheromone dictionary that replaces traditional edge-based approaches. Because the algorithm's success is strongly parameter-dependent, the article details all configuration parameters and their governing mathematical formulas, with a specific focus on disturbance management to escape heuristic traps. A complete breakdown of the move detection and overarching search components is provided. Finally, performance is evaluated across various puzzle sizes and search strategies, proving the algorithm's robustness and viability as a solver for extreme-scale problems up to 11×11 states.

Keywords

sliding puzzle, ACS, heuristic-driven search, pheromone-driven search, Manhattan distance, linear conflict, pheromones, tiles relations, evaporation, colony disturbance,

1. Introduction

Heuristic algorithms are widely used to solve optimization problems and complex tasks where deterministic solving algorithms are either unknown or computationally prohibitive. Frequently, the mathematical models for these algorithms draw inspiration from the animal or plant kingdoms. Within this domain, swarm intelligence algorithms form a highly effective subset. These methods implement computational simulations of collective creature behavior, mirroring swarm dynamics that enable a group to synchronize and achieve a common goal, even as individual agents independently alter their directions.

The defining challenge of the sliding puzzle is its overwhelming variety of states, which grows factorially alongside the board size. Traditional deterministic algorithms struggle to scale and inevitably become ineffective on large $n \times n$ sliding puzzles. As Wilt and Ruml (2012) [1] demonstrated, the efficiency of Weighted A* (WA^*) depends heavily on the heuristic error remaining proportional to the actual distance to the goal. In scaled sliding puzzles, however, the true cost (h^*) outpaces the standard Manhattan Distance (h) due to intense tile congestion. This disproportionate growth generates vast heuristic plateaus where WA^* is forced to expand an exponential number of nodes without advancing toward the solution, ultimately blinded by its own greediness. While swarm intelligence methods like the Artificial Bee Colony algorithm have successfully solved smaller sliding puzzles [2], their viability on extreme-scale boards remains largely untested. Adapting an algorithm inspired by Dorigo's Ant Colony System (ACS) [3] not only preserves the intelligent searching, but also provides a highly informative perspective on the structural versatility and scalability of swarm algorithms.

2. Heuristic Function

The proposed heuristic function for this algorithm is the sum of two different metrics:

- **Manhattan Distance**: The sum of the vertical and horizontal distances of each tile from its target position in the solved state. This distance is calculated by summing the absolute differences

IVUS 2026: 31st International Conference on Information Society and University Studies, 12 May 2026, Kaunas, Lithuania
bk318913@student.polsl.pl (B. Kostyk)
Copyright © 2026 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-06

CEUR-WS.org/Vol-4317/paper20.pdf

of the row and column coordinates.
- **Linear Conflict**: An enhancement that adds a penalty (+2) to the overall heuristic value when two tiles are in their correct target row or column but are reversed relative to each other. This creates a ”conflict” because one tile must temporarily move out of the row/column to allow the other to pass, requiring at least two extra moves [4].

3. Structures

3.1. Node

A structure whose main purpose is to preserve necessary information about the search path alongside the state of the sliding puzzle itself.

- **Heuristic function value** - The sum of the Manhattan distance and linear conflict values. This information is necessary for comparing different nodes and finding the ”best ant”—the ant that has found the node with the best heuristic function value in the colony.
- **State** - The state of the sliding puzzle, representing the specific sequence of numbers and how the puzzle would physically look.
- **Position of blank element** - The actual position of the empty space in the sliding puzzle for a specific state, representing a physical void where tiles can slide.
- **Parent** - A reference to the previous node from which the move was made to reach the current state.

3.2. Ant

The agent, which preserves information regarding:

- **Visited states** - States that the ant has already visited during one global iteration (before starting from a new position determined by the elitism of the best ants). This memory resets after the global iteration ends.
- **Current node** - The node where the specific ant is currently located.
- **Best node** - The node with the best heuristic function value the ant has explore during the current global iteration. Resets at the end of the iteration.

3.3. Pheromones Dictionary

A structure that preserves key-value pairs, where the key represents a correct spatial relation between tiles in an arbitrary-sized sliding puzzle. A ”correct relation” is defined as an ordered triple consisting of a tile, its neighbor, and the directional relation between them. These relations are mapped to integers: 0 (up), 1 (right), 2 (down), and 3 (left). This ordered triple acts as the key, while its corresponding value represents the amount of deposited pheromone, ranging from 0 to 1. This value is initialized by the τ_0 parameter, which establishes the baseline pheromone level for every correct relation.

The dictionary exclusively stores correct relations—meaning relations derived directly from a solved sliding puzzle state. For example, Fig. 1 illustrates two states of a 4×4 sliding puzzle. The left state is solved, so every relation within it is correct (e.g., the arrow originating from tile ”1” and pointing to tile ”2”). Conversely, the unsolved state on the right contains incorrect relations, which are also highlighted with arrows. For instance, the triple $(1, 5, 3)$ represents an incorrect relation, depicted by an arrow originating from tile ”1” and pointing to tile ”5”. In this specific state, there are 9 incorrect relations in total. The theoretical assumption underpinning this structure is that ants will prioritize moving entire micro-structures of correctly related tiles, driven by the algorithmic desire to traverse paths with higher pheromone concentrations.

This relation-based pheromone dictionary replaces the traditional ACS approach of depositing pheromones on the edges between distinct state nodes. This architectural adaptation was strictly

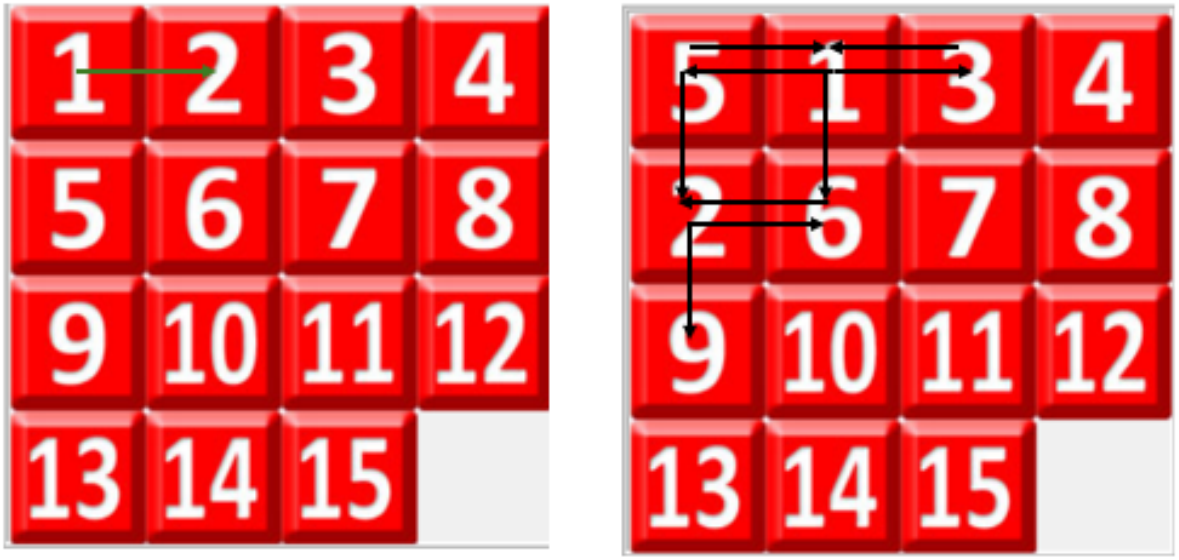


Figure 1: Relations between tiles. The green arrow on the left sliding puzzle's solved state represents an example of correct relation between tiles and black arrows on the right sliding puzzle's unsolved state represents incorrect relations

necessary to radically reduce the number of potential options where an ant can leave a pheromone trail. Because the sliding puzzle is a massive spatial problem (for instance, an 8×8 puzzle possesses $\approx 10^{89}$ states, far exceeding the estimated number of atoms in the observable universe), assigning pheromones to individual state transitions would be both useless in terms of steering the colony and preserving these separate values in memory.

4. Parameters

The algorithm can be finely tuned using a wide variety of parameters. Appropriate parameter selection for a specific sliding puzzle size is crucial, as differences between sets can lead to disparate performance outcomes regarding the number of steps in the found path and the time spent reaching a solution. To understand their use cases, the parameters are divided into the following functional groups.

Sliding puzzle initialization This group provides information about the initial state and initializes the pheromone dictionary.

- `initialState` - The one-dimensional representation of the starting state of the sliding puzzle.
- n - The side size of the sliding puzzle.

Colony management This group steers the colony in terms of population and movement.

- N - Number of ants in the colony.
- `top` - Number of elite ants from whom the best nodes are selected to initialize the next global iteration.
- $s_{\min}$ - Minimum number of steps an ant can make exploring states during one global iteration.
- $s_{\max}$ - Maximum number of steps an ant can make exploring states during one global iteration.

Pheromones management This group manages the pheromones ants leave on sliding puzzle relations.

- τ_0 - The base pheromone value for every correct relation.

- ρ - Global evaporation coefficient. Ensures that relations with additional pheromones will eventually relax back to the τ_0 value, meaning rarely visited relations lose their temporary pheromone impact.
- ξ - Local evaporation coefficient. Implies a communication method among ants during a global iteration. When an ant traverses a relation, it evaporates the pheromone slightly to make that path temporarily less attractive to others, encouraging broader exploration across the colony.
- α - Pheromone influence factor. Decides how strongly pheromones impact an ant's movement decision.

Disturbance management A crucial group of parameters ensuring the colony prefers escaping heuristic function "traps" (local minima where no standard move immediately improves the heuristic).

- R - A stagnation scaling factor that regulates how heavily non-improving iterations impact the increase in ant exploration steps and the overall disturbance of the colony.
- $a_{\min}$ - The influence of the heuristic function under low-disturbance (calm) conditions. The colony is less tolerant of moves that degrade the heuristic function.
- $a_{\max}$ - The influence of the heuristic function under high-disturbance conditions. The ants tolerate more heuristic degradation and become less selective, viewing different "bad" moves as more equally acceptable to escape local minima.
- D_c - Convergence rate of colony stability. Dictates how quickly ants stabilizes (reduces its disturbance coefficient) after finding a new global minimum.
- D_p - Stochastic escalation rate. Dictates how fast ants become disturbed during a period of stagnation.

Heuristic function impact A standalone parameter controlling the base importance of the heuristic.

- β - The influence of the heuristic function on ant move-making decisions.

5. Mathematical Formulation

This section provides the formulas and equations detailing the mathematical model of the algorithm, grouped similarly to the parameters above. Note that while these formulas effectively govern the current iteration of the algorithm, better mathematical optimizations may exist. Certain formulas have been adapted from Dorigo's works [3, 5].

5.1. Colony management

Global stagnation

$$q = \min\left(1, \frac{stagnation}{R}\right) \tag{1}$$

- q - The coefficient of stagnation.
- *stagnation* - The number of consecutive global iterations where the colony failed to find a state with a better heuristic function than previously recorded.

Ant steps number adaptation (when colony finds a better heuristic value)

$$s = s_{\min} \tag{2}$$

- s - The actual amount of steps each ant makes in a global iteration.

Ant steps number adaptation (when colony struggles to find a better heuristic value)

$$s = \lfloor s_{\min} + (s_{\max} - s_{\min} \cdot q^2) \rfloor \tag{3}$$

- q - The coefficient of stagnation.

5.2. Pheromones management

Pheromone value bounds

$$\tau_{\min} = \tau_0 \tag{4}$$

$$\tau_{\max} = \frac{1}{\frac{1+h_{gb}}{scale}} \tag{5}$$

- $scale$ - Total number of tiles in the sliding puzzle (including the blank space).
- h_g - The global best heuristic function value found so far.

Finding distance This distance function is an adaptation of the original ACS metric, specifically tailored to satisfy the constraints of the sliding puzzle rather than traditional spatial routing.

$$distance = \frac{1}{\frac{h_b}{scale}} \tag{6}$$

- $scale$ - Total number of tiles.
- h_b - The best heuristic value currently observed among all ants in the iteration.

Enhancement of pheromones

$$\tau_n = (1 - \rho) \cdot \tau_c + distance \tag{7}$$

- τ_n - New amount of pheromone.
- τ_c - Current amount of pheromone.
- ρ - Global evaporation coefficient.

Pheromones impact on ant decision

$$\tau = \tau_0 + \sum_{i=1}^{n} \sum_{j=0}^{3} P(t_0, t_i, j) \tag{8}$$

- t_0 - The tile that switched places with the blank space.
- t_i - The corresponding neighbor of the t_0 tile.
- n - The number of neighbors of t_0.
- j - Relation identifier (0 - up, 1 - right, 2 - down, 3 - left).
- P - The relation-pheromone dictionary, where the ordered triple (t_0, t_i, j) returns the current amount of pheromone.

Local pheromones actualization

$$\tau_n = (1 - \xi) \cdot \tau_c + \xi \cdot \tau_0 \tag{9}$$

- $\xi \cdot \tau_0$ is an addend ensuring relaxation back to the initial pheromone value.
- τ_n - New amount of pheromone.
- τ_c - Current amount of pheromone.

5.3. Disturbance Management

Maximal heuristic function value change The maximal theoretical heuristic change in a single move. In the worst case, moving one tile can create a maximum of $n - 1$ new linear conflicts. Because resolving a linear conflict requires two additional moves, we multiply by 2. Moving a tile towards its destination inherently subtracts 1 from the Manhattan distance, yielding the final equation:

$$\delta_0 = 2 \cdot (n - 1) - 1 \tag{10}$$

- n - Size of one side of the sliding puzzle.

Determining lower bound of colony disturbance coefficient Derived from the equation $\exp(-\log_{D_{\min}} \delta_0)^{\beta} = a_{\min}$, instead of power of β there is multiplication of β. It was experimentally observed that choosing multiplication instead of a power function led to better algorithmic effectiveness:

$$D_{\min} = \exp\left(-\log_{a_{\min}} \delta_0\right) \cdot \beta \tag{11}$$

Determining upper bound of colony disturbance coefficient Derived from the equation $\exp(-\log_{D_{\max}} \delta_0)^{\beta} = a_{\max}$ in the same way as previous equation:

$$D_{\max} = \exp\left(-\log_{a_{\max}} \delta_0\right) \cdot \beta \tag{12}$$

Determining disturbance coefficient initial value The square root function initializes the disturbance coefficient using the geometric mean, keeping D closer to $D_{\min}$ than $D_{\max}$ initially.

$$D = \sqrt{D_{\min} \cdot D_{\max}} \tag{13}$$

Adjusting colony disturbance (when finding a better solution)

$$D = D_{\min} + (1 - D_c) \cdot (D - D_{\min}) \tag{14}$$

Adjusting colony disturbance (during stagnation)

$$D = \min\left(D_{\max}, \max\left(D_{\min}, (1 - D_p) \cdot D + D_p \cdot (D_{\min} + (D_{\max} - D_{\min} \cdot q^2))\right)\right) \tag{15}$$

- q - The coefficient of stagnation.

5.4. Heuristic Function Value Impact on Ant Decision

When heuristic function value improves

$$\eta = e^{-\delta} \tag{16}$$

- δ - The change in heuristic function value (new state heuristic function value minus previous one).

When heuristic function value degrades This formula ensures that η penalizes transitions to degrading states, especially when the disturbance coefficient is lower than the heuristic function change ($D < \delta$). Conversely, as D increases and significantly exceeds δ, the ants' tolerance for suboptimal states broadens. This mechanism simulates a highly disturbed colony that accepts both heuristic improvements and degradations to successfully escape local minima. Mathematically, it guarantees that the ants' reliance on the heuristic diminishes as the disturbance reaches extreme levels:

$$\lim_{n \to \infty} \sqrt[n]{e} = 1 \implies \lim_{n \to \infty} \frac{1}{\sqrt[n]{e}} = 1$$

$$\eta = e^{-\log_D \delta} \tag{17}$$

5.5. Ant Determining Weight of Specific Move

$$w = \tau^{\alpha} \cdot \eta^{\beta} \tag{18}$$

- w - Weight determining the probability of a specific move being chosen.

6. Proposed Algorithms

Ant move detection algorithm This algorithm dictates how an ant compares neighboring states to its current state. It assigns a weight to each possible neighbor based on heuristic function change and the pheromone values derived from the new relations created by moving the blank tile. A higher weight increases the likelihood of selection via a roulette wheel mechanism. Local evaporation then ensures subsequent ants are slightly less likely to follow the exact same path, enhancing broad exploration.

Procedure Ant move detection

Input: ant, PD - pheromones dictionary, τ_0 - initial pheromone value, ξ - coefficient of local evaporation, α - power of pheromone impact, D - disturbance coefficient, β - power of heuristic function impact on ant move decision

Output: node object

Start

$h0$ = heuristic function value of the state

foreach *valid move of the blank element* **do**

  if *state was visited by ant or it is a parent of that state* **then**

    skip this move and continue checking other moves

  end

  h = heuristic function value of this specific state

  τ = sum of pheromone values corresponding to new relations of the switched tile

  $\delta = h - h0$

  if $\delta \leq 0$ **then**

    $\eta = e^{-\delta}$ (16)

  end

  else

    $\eta = e^{-\log_D \delta}$ (17)

  end

  $w = \tau^{\alpha} \cdot \eta^{\beta}$

  Add weight to the list of weights

  Add new state, its blank element position, and heuristic function value to the candidates list

end

if *candidates list is empty* **then**

  Revert ant to previous state

  return state, where an ant is located

end

else

  $W = \sum_{w \in weights} w$

  r = random real number from interval (0, W)

  Use a roulette wheel based on cumulative probability to identify the interval where random number is situated

  winner = candidate from candidates list corresponding to the chosen interval

end

actualize pheromones locally (9)

return new node consisting of winner state, its zero position, parent state, heuristic function value

End

Ants search algorithm The main algorithm responsible for initializing, actualizing, and managing the colony as a whole.

Algorithm: Ants search

Input: $s_{\min}$ - minimal number of steps, `top` - number of nodes chosen to initialize the next global iteration
N - total number of ants
Output: node object
Start
Initialize pheromone dictionary - PD
$stagnation = 0$
$h_g = \infty$
$s = s_{\min}$
Set maximal theoretical heuristic function change (10)
Initialize disturbance coefficient lower and higher bounds (11)(12)
Determine disturbance coefficient initial value (13)
Set pheromone value minimal bound (4)
Initialize colony
while *not solved* **do**
    foreach *step in s* **do**
        foreach *ant in colony* **do**
            ant's current node = result of procedure "Ant move detection"
            if *ant does not have best node or ant's current node heuristic function value is less than ant's best node* **then**
                ant's best node = ant's current node
            end
            if *ant's current node's state == solution state* **then**
                return ant's current node
            end
        end
    end
    Choose the best node among all ants according to lowest heuristic function value
    $h_{\min}$ = best node heuristic value
    Enhance pheromones for all correct relations of the best node's state (7)
    Set pheromone value upper bound (5)
    Actualize all other pheromones according to function $\tau_n = (1 - \rho) \cdot \tau_c$
    if $h_{\min} < h_g$ **then**
        $h_g = h_{\min}$
        $stagnation = 0$
        $s = s_{\min}$
        Reduce colony disturbance (14)
    end
    else
        Calculate coefficient of stagnation (1)
        Increase colony number of steps (3)
        Increase colony disturbance (15)
    end
    Choose `top` best nodes
    For each pack of $\lfloor \frac{N}{top} \rceil$ ants, assign one of the `top` nodes as its starting current node
end

Table 1
System Specifications

Category	Details
CPU	Intel Core i7-14700KF (20 cores and 28 threads, base frequency 3.4 GHz, up to 5.6 GHz boost)
RAM	Patriot Viper Venom 32 GB DDR5-6000 (operating at 4800 MT/s during tests), CL36
OS	Windows 11 Pro
Programming language	Python 3.11.15
Interpreter	PyPy 7.3.21 (64bit)
Environment	Visual Studio Code

Table 2
Metrics for probabilistic nature test conducted 30 times on 4×4 state

Metrics	Steps	Time (seconds)
Minimum	71	0.058
Maximum	111	0.311
Median	89	0.102
Mean	89	0.107
Sample standard deviation	9	0.045
Coefficient of variation	10.36%	41.56%

Table 3
Metrics for probabilistic nature test conducted 20 times on 7×7 state

Metrics	Steps	Time (seconds)
Minimum	730	1.911
Maximum	1508	46.77
Median	883	2.867
Mean	938	5.826
Sample standard deviation	185	10.062
Coefficient of variation	19.73%	172.71%

7. Performance

Performance testing was conducted on a system with the specifications shown in Tab. 1. The conducted tests demonstrate the algorithm's probabilistic behavior and its dependence on parameter configurations. Apart from the mean number of steps and time required to solve a given $N \times N$ sliding puzzle, additional metrics—such as minimum, maximum, median, sample standard deviation, and coefficient of variation—were included to provide a deeper insight into the ACS-inspired algorithm. In total, the algorithm was initialized 145 times, with a cumulative testing duration of 28 minutes and 54 seconds.

7.1. Probabilistic Nature Test

Tests were conducted on two states - 30 times on 4×4 state and 20 times on 7×7 sliding puzzles (Tab. 2 and Tab. 3, respectively). The specific 4×4 state used requires at least 56 steps to be solved optimally. The shared parameters for both states were: $N = 120$, $\texttt{top} = 20$, $\tau_0 = 0.1$, $\rho = 0.2$, $\xi = 0.4$, $\alpha = 1$, $R = 10$, $a_{\min} = 0.1$, $a_{\max} = 0.85$, $D_c = 0.6$, $D_p = 0.05$, $\beta = 2$. The varied parameters were $s_{\min} = 12$ and $s_{\max} = 16$ for the 4×4 state, and $s_{\min} = 35$ and $s_{\max} = 49$ for the 7×7 state.

Table 4
Metrics heuristic-driven search conducted 15 times on same 8×8 state

Metrics	Steps	Time (seconds)
Minimum	1391	1.752
Maximum	2547	8.165
Median	1877	3.292
Mean	1949	3.973
Sample standard deviation	328	1.983
Coefficient of variation	16.83%	49.90%

Table 5
Metrics for pheromone-driven search conducted 15 times on same 8×8 state

Metrics	Steps	Time (seconds)
Minimum	2129	7.27
Maximum	3725	127.816
Median	2565	19.83
Mean	2673	32.725
Sample standard deviation	391	34.075
Coefficient of variation	14.64%	104.13%

Table 6
Metrics for balanced search conducted 15 times on same 8×8 state

Metrics	Steps	Time (seconds)
Minimum	1671	3.939
Maximum	3429	96.446
Median	2249	18.865
Mean	2334	31.374
Sample standard deviation	427	29.958
Coefficient of variation	18.32%	95.49%

7.2. Different Sets of Parameters, Same State Test

In this test, an 8×8 state was the same starting point while three different parameter sets were applied, highlighting three distinct colony behaviors: heuristic-driven search (Tab. 4), pheromone-driven search (Tab. 5), and a balanced search (Tab. 6). Shared parameters: $N = 40$, $\texttt{top} = 10$, $s_{\min} = 50$, $s_{\max} = 100$, $\tau_0 = 0.1$, $\rho = 0.2$, $\xi = 0.4$, $R = 10$, $a_{\min} = 0.1$, $a_{\max} = 0.85$, $D_c = 0.6$, $D_p = 0.05$. The varying parameters were α and β. First set of parameters: $\alpha = 1$, $\beta = 3$, for second: $\alpha = 2$, $\beta = 1$ and for third: $\alpha = 1$, $\beta = 1$.

7.3. Random States Test

Here, three sets of parameters, which differs only with steps, were applied to 25 random 5×5 states (Tab. 7), 15 random 8×8 states (Tab. 8) and also to 10 random 11×11 states (Tab. 9) to test the algorithm's ceiling on highly complex problems. Shared parameters: $N = 60$, $\texttt{top} = 20$, $\tau_0 = 0.1$, $\rho = 0.2$, $\xi = 0.4$, $\alpha = 1$, $R = 10$, $a_{\min} = 0.1$, $a_{\max} = 0.85$, $D_c = 0.6$, $D_p = 0.05$, $\beta = 3$. The difference lies in the step bounds: $s_{\min} = 10$, $s_{\max} = 20$ for 5×5 states, $s_{\min} = 40$, $s_{\max} = 90$ for 8×8 states and $s_{\min} = 100$, $s_{\max} = 180$ for 11×11 states

Table 7
Metrics for search conducted 25 times on random 5 × 5 states with same algorithm parameters

Metrics	Steps	Time (seconds)
Minimum	182	0.146
Maximum	380	0.489
Median	242	0.229
Mean	249	0.243
Sample standard deviation	52	0.094
Coefficient of variation	21.20%	38.73%

Table 8
Metrics for search conducted 15 times on random 8 × 8 states with same algorithm parameters

Metrics	Steps	Time (seconds)
Minimum	1343	2.27
Maximum	2466	16.354
Median	1811	3.572
Mean	1805	5.613
Sample standard deviation	355	4.379
Coefficient of variation	19.70%	78.02%

Table 9
Metrics for search conducted 10 times on random 11 × 11 states with same algorithm parameters

Metrics	Steps	Time (seconds)
Minimum	6831	17.232
Maximum	10371	128.004
Median	8083	38.891
Mean	8252	50.576
Sample standard deviation	1266	34.931
Coefficient of variation	15.34%	69.07%

7.4. Overall Performance Analysis

Analyzing the collected statistics, it becomes evident that for every test scenario, the number of steps remains relatively consistent, oscillating with a coefficient of variation between approximately 10% and 21%. In contrast, execution time is a highly unstable metric; in extreme cases (e.g., the 7x7 probabilistic test), the sample standard deviation significantly exceeds the mean value. Furthermore, when comparing the 8x8 configurations, the pheromone-driven search exhibits the lowest coefficient of variation for steps (14.64%). This suggests that this parameter set makes the algorithm more stable in terms of path length, although it suffers from the worst time stability. Conversely, the heuristic-driven search proved to be the most efficient approach overall, succeeding both in terms of the lowest mean steps and execution time. Interestingly, as the problem scale increases (up to the 11x11 state), the algorithm maintains its step consistency, proving its robustness. All of these statistics can also be observed in graphical representations as box plots in Fig. (2) and Fig. (3), which illustrate the distribution of results for each experimental setup. They provide direct insights into the algorithm's stability (represented by the height of the boxes) and its convergence reliability (indicated by the presence of outliers).

Figure 2: The first five plots illustrate the results of the algorithm's probabilistic nature tests (4x4, 7x7) and search strategies (heuristic, pheromone, balanced for 8x8) across multiple runs on predefined states. The remaining three evaluate random 5x5, 8x8, and 11x11 configurations. For each box, the dashed line is the median, the red cross is the mean, and the box shows the interquartile range. Whiskers indicate min-max values; circles denote outliers.

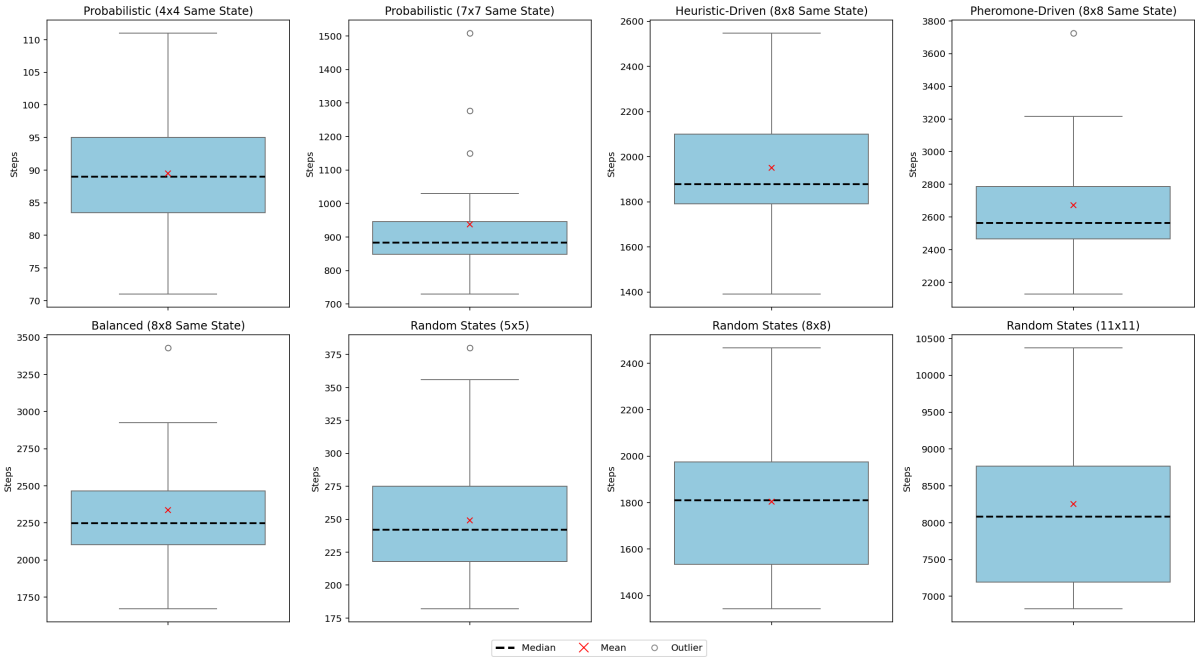
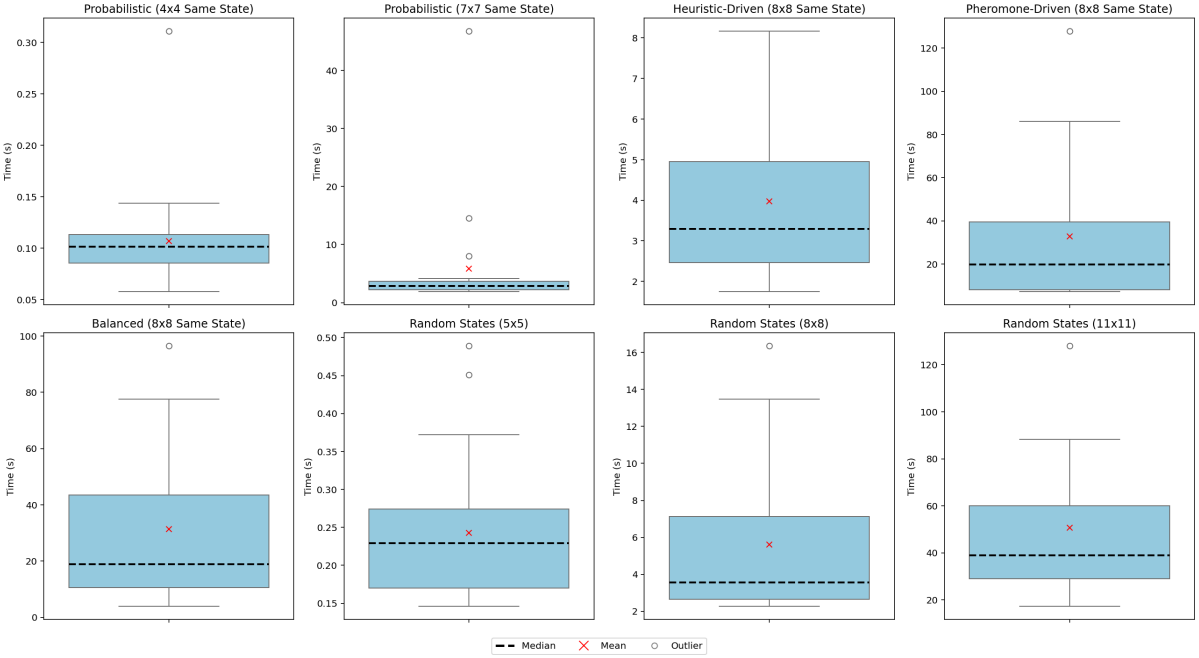


Figure 3: Box-plots of execution time (s) for all conducted tests.



8. Discussion

This algorithm's greatest strength is its ability to navigate massive sliding puzzles in a reasonably short time. An 8×8 sliding puzzle contains $\approx 10^{89}$ possible states (billions of times more than the estimated atoms in the observable universe). For standard AI algorithms, getting out from "plateaus" in this massive search space is a distinct challenge.

As Korf noted [6], A* is impractical for large state spaces due to its exponential memory requirements. While Disjoint Pattern Databases [7] revolutionized optimal solving for the 15-puzzle and 24-puzzle, their memory requirements scale exponentially, making them impossible to compute or store for extreme scales like an 8×8 or 11×11 puzzle. By 2004, Felner, Korf, and Hanan concluded that finding optimal solutions to random instances of the Thirty-Five Puzzle (6×6) was beyond the reach of existing techniques and machines [8].

The logical workaround is adopting a suboptimal solution search that finds *any* solution while allowing the user to tune for optimality. However, even the famous weighted A* can fail on larger scales. Wilt and Ruml [1] demonstrated that weighted heuristic performance degrades severely in domains where the true distance to the goal, $d^*(n)$, correlates poorly with $h(n)$.

9. Conclusion

This research proves the viability of an ACS-inspired algorithm for solving extreme-scale state space problems. It successfully solved different random 11×11 sliding puzzle states (over 10^{200} states), albeit requiring more time. Despite the success of this concept, a significant drawback remains: the difficulty in selecting the optimal set of parameters for a given puzzle size. This article also demonstrates that swarm algorithms, if finely tuned to the specific mechanics of a problem, can act as powerful solvers for unusual or highly complex puzzles. Ultimately, the hope is that this research inspires other scientists, students, and professors to give swarm algorithms a fresh outlook.

Acknowledgments

The author would like to express his gratitude to Professor Marcin Woźniak for the initial inspiration regarding the application of Ant Colony Systems to sliding puzzle problems, which laid the foundation for this research.

Declaration on Generative AI

During the preparation of this work, the author used Gemini 3.1 Pro: in order to: Grammar and spelling check, Paraphrase and reword, Plagiarism detection. After using this service, the author reviewed and edited the content as needed and takes full responsibility for the publication's content.

References

[1] C. Wilt, W. Ruml, When does weighted a* fail?, Proceedings of the International Symposium on Combinatorial Search 3 (2021) 137–144. URL: https://ojs.aaai.org/index.php/SOCS/article/view/18250. doi:10.1609/socs.v3i1.18250.

[2] A. Tuncer, 15-puzzle problem solving with the artificial bee colony algorithm based on pattern database, JUCS - Journal of Universal Computer Science 27 (2021) 635–645. doi:10.3897/jucs.65202.

[3] M. Dorigo, L. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1997) 53–66. doi:10.1109/4235.585892.

[4] O. Hansson, A. Mayer, M. Yung, Criticizing solutions to relaxed models yields powerful admissible heuristics, Information Sciences 63 (1992) 207–227. URL: https://www.sciencedirect.com/science/article/pii/002002559290070O. doi:https://doi.org/10.1016/0020-0255(92)90070-O.

[5] M. Dorigo, V. Maniezzo, A. Colorni, Ant system: optimization by a colony of cooperating agents, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 26 (1996) 29–41. doi:10.1109/3477.484436.

[6] R. E. Korf, Depth-first iterative-deepening: An optimal admissible tree search, Artificial Intelligence 27 (1985) 97–109. URL: https://www.sciencedirect.com/science/article/pii/0004370285900840. doi:https://doi.org/10.1016/0004-3702(85)90084-0.
[7] R. E. Korf, A. Felner, Disjoint pattern database heuristics, Artificial Intelligence 134 (2002) 9–22. URL: https://www.sciencedirect.com/science/article/pii/S0004370201000923. doi:https://doi.org/10.1016/S0004-3702(01)00092-3.
[8] A. Felner, R. E. Korf, S. Hanan, Additive pattern database heuristics, Journal of Artificial Intelligence Research 22 (2004) 279–318. URL: http://dx.doi.org/10.1613/jair.1480. doi:10.1613/jair.1480.

A. Online Resources

The source code of the project, implemented in Python, can be found here:

- GitHub