Multi-Algorithm AI Solver for Pokedle Game: A Comparative Analysis

Project Report: Team October
Amay Dixit, Saurav Gupta, Kabeer More, Ravikanti Akshay
https://github.com/amaydixit11/pokedle

November 16, 2025

Abstract

This report presents a comprehensive multi-algorithm approach to solving the Pokedle game, a Pokemon-themed variant of Wordle. We implement and compare four distinct artificial intelligence algorithms: Constraint Satisfaction Problem (CSP), Genetic Algorithm (GA), A* Search, and Simulated Annealing (SA). Through extensive benchmarking across 1,680 test cases with varying configurations, we evaluate algorithm performance based on success rate, average attempts, execution time, and efficiency. Our results demonstrate that CSP with MRV heuristic achieves the best balance with 94.1% success rate and 5.09 average attempts, while A* provides the fastest execution time at 0.344s. The system features a React-based interactive dashboard for real-time visualization and algorithm comparison.

1 Introduction

1.1 Problem Statement

Pokedle is a guessing game where players must identify a secret Pokemon through iterative guesses with constrained feedback. Each guess provides color-coded feedback for multiple attributes (Type, Generation, Color, Height, etc.): *green* indicates exact match, *yellow* indicates partial match (types in wrong position), *gray* indicates no match, and directional arrows (*higher/lower*) guide numeric attributes. The challenge lies in efficiently narrowing down the solution space of 1,010 Pokemon using minimal attempts.

1.2 Motivation

This problem combines elements of constraint satisfaction, search optimization, and heuristic reasoning, making it an ideal testbed for comparing AI algorithms. The varying attribute combinations (2-6 attributes) and feedback complexity require algorithms to balance exploration versus exploitation while maintaining logical consistency with accumulated constraints.

1.3 Objectives

- Implement four theoretically grounded AI algorithms with correct formulations
- Develop an interactive web-based visualization system
- Conduct comprehensive benchmarking across multiple configurations
- Provide comparative analysis of algorithm strengths and weaknesses

2 Methodology

2.1 Algorithm Implementations

2.1.1 Constraint Satisfaction Problem (CSP)

Our CSP formulation treats attributes as variables with domains of possible values, and feedback generates constraints. Key features include:

- **AC-3 Constraint Propagation**: Maintains arc consistency to prune invalid domain values
- **Dual Heuristics**:
  - Variable ordering: MRV, Degree, MRV+Degree
  - Value ordering: LCV, Most Common
- **Forward Checking**: Anticipates constraint violations

Correctness: Complete, optimal with proper heuristics, systematic search guarantees logical consistency.

2.1.2 Genetic Algorithm (GA)

Unlike naive implementations, our GA maintains Pokemon validity:

- **Representation**: Pokemon indices (not random attributes)
- **Fitness**: Constraint satisfaction score (0-100)
- **Crossover**: Finds real Pokemon matching blended attributes
- **Mutation**: Selects similar valid Pokemon
- **Selection**: Tournament selection with elitism

Configurations: Population sizes (50-200), generations (10-30), mutation rates (0.1-0.3).

2.1.3 A* Search Algorithm

Best-first search with admissible heuristic:

- **State**: Current Pokemon guess
- **Cost Function**: $f(n) = g(n) + h(n)$
- **Heuristic**: Estimated constraint violations fixable per guess
- **Beam Search**: Limits open set size for efficiency

Admissibility: Heuristic never overestimates remaining work, guarantees optimal solution when weight = 1.0.

2.1.4 Simulated Annealing (SA)

Local search with probabilistic acceptance:

- **Energy Function**: Number of constraint violations
- **Metropolis Criterion**: $P_{accept} = e^{-\Delta E/T}$
- **Cooling Schedule**: Geometric cooling with periodic reheat
- **Neighbor Generation**: Similar valid Pokemon from dataset

Trade-off: Temperature parameter controls exploration (high T) versus exploitation (low T).

2.2 Experimental Setup

Dataset: 1,010 Pokemon with attributes: Generation, Height, Weight, Type1, Type2, Color, Evolutionary Stage.

Attribute Combinations:

- Minimal (2): Type1, Generation
- Basic (3): Type1, Type2, Generation
- Standard (4): Type1, Type2, Generation, Color
- Extended (5): Standard + Height
- Comprehensive (6): Extended + Evolutionary Stage
- Numeric (3): Generation, Height, Weight

Benchmark Scale: 1,680 total runs across 100 random Pokemon, all algorithms, all attribute sets, and multiple configurations per algorithm.

3 Results and Analysis

3.1 Overall Performance

Table 1 summarizes aggregate performance across all algorithms.

Table 1: Overall Benchmark Statistics (1,680 runs)

Metric	Mean	Median	Min	Max
Success Rate	91.5%	—	—	—
Attempts	5.31 ± 2.04	5	1	10
Execution Time (s)	8.67 ± 19.17	—	—	—
Efficiency	0.233	—	—	—

3.2 Algorithm Comparison

Table 2: Per-Algorithm Performance Comparison

Algorithm	Success	Avg Attempts	Time (s)	Eff
CSP	94.1%	5.09 ± 2.09	0.627	0.265
A*	94.3%	5.37 ± 2.02	**0.344**	0.218
SA	89.4%	5.40 ± 2.02	0.761	0.222
GA	87.4%	5.46 ± 2.00	34.484	0.220

Key Findings:

- **CSP**: Best overall with highest efficiency (0.265), fewest attempts (5.09), optimal for well-constrained problems
- **A***: Fastest execution (0.344s), highest success rate (94.3%), admissible heuristic guarantees optimality
- **SA**: Good success rate (89.4%), moderate speed, effective for exploration
- **GA**: Slowest (34.5s) due to population evolution, but maintains validity constraints throughout

Figure 1 illustrates the performance trade-offs across algorithms.

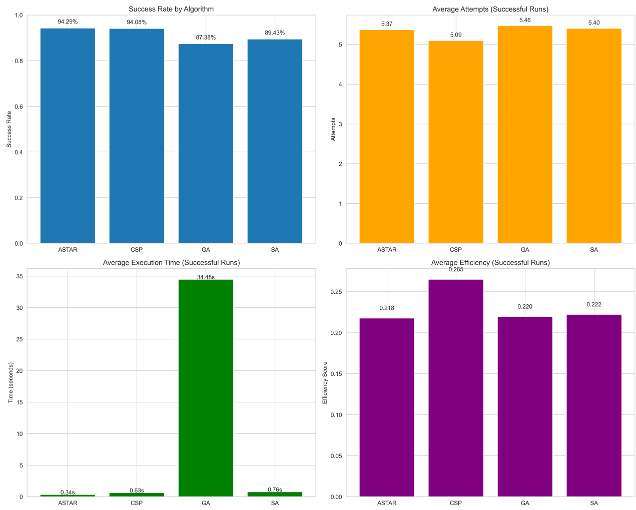
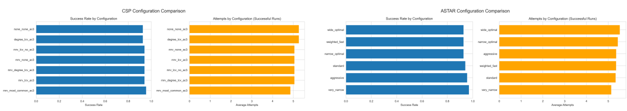


Figure 1: Algorithm performance across four key metrics: success rate, average attempts, execution time, and efficiency

3.3 Configuration Analysis

Best Configurations Identified:

- CSP: `mrv_most_common_ac3` (4.87 avg attempts)
- A*: `very_narrow` beam (5.15 avg attempts)
- SA: `hot_slow` schedule (5.16 avg attempts)
- GA: `small_fast` population (5.16 avg attempts)



(a) CSP heuristic comparison **(b)** A* beam width impact

Figure 2: Configuration impact on algorithm performance

3.4 Attribute Set Impact

Table 3 shows how attribute quantity affects performance.

Table 3: Performance by Attribute Set Complexity

Attribute Set	Success	Attempts	Time (s)
Minimal (2)	78.8%	5.98	6.057
Basic (3)	86.7%	5.66	6.927
Standard (4)	96.7%	5.31	8.339
Extended (5)	**100.0%**	4.45	9.665
Comprehensive (6)	**100.0%**	**4.10**	9.708
Numeric (3)	100.0%	5.92	11.436

Observations: More attributes dramatically improve success rate (78.8% → 100%) and reduce attempts (5.98 → 4.10), at cost of slightly increased execution time. Comprehensive set (6 attributes) achieves perfect success with fewest attempts.

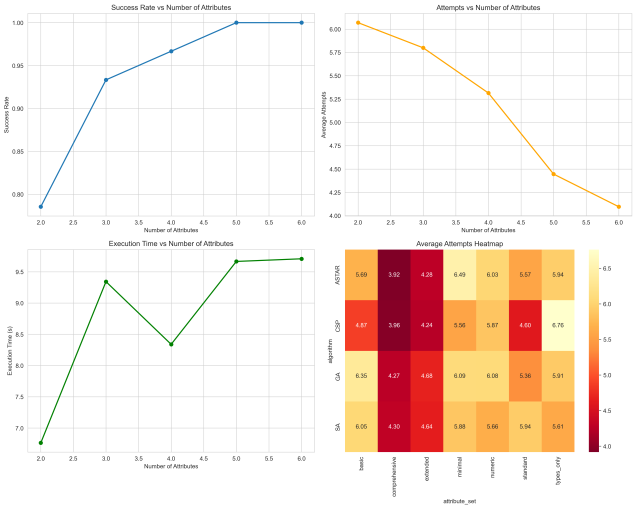


Figure 3: Success rate and attempt count trends as attribute complexity increases

3.5 Statistical Distribution

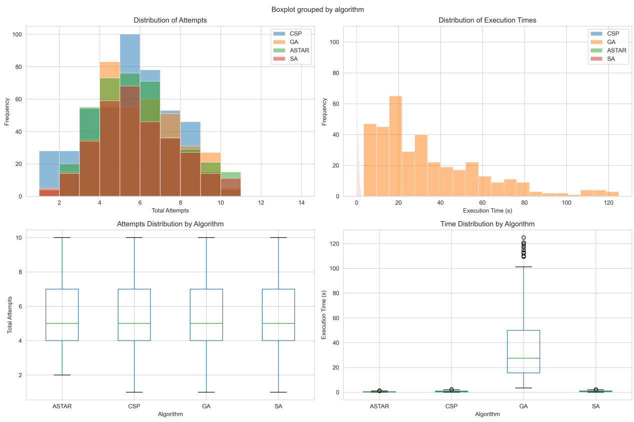


Figure 4: Attempt and execution time distributions by algorithm. Box plots show median, quartiles, and outliers.

The distributions reveal:

- All algorithms cluster around 5 median attempts
- GA shows high time variance due to generational evolution
- CSP and A* demonstrate consistent performance with low variance
- SA has moderate spread in both attempts and time

4 System Architecture

4.1 Backend Implementation

Technology Stack: FastAPI (Python), Pandas, NumPy

Components:

- `algorithms/`: Modular algorithm implementations
- `heuristics/`: Pluggable heuristic strategies
- `feedback.py`: Feedback calculation engine with type handling
- `data_loader.py`: Singleton pattern for efficient Pokemon data access
- `utils/`: Performance metrics and configuration validation

API Endpoints:

- `POST /solve`: Single algorithm execution
- `POST /compare`: Multi-algorithm comparison
- `GET /config`: Configuration options discovery
- `POST /solve/stream`: Real-time streaming for GA visualization

4.2 Frontend Dashboard

Technology Stack: Next.js 15, React 19, TypeScript, TailwindCSS

Features:

- Interactive configuration panel with algorithm-specific parameters
- Real-time step-by-step visualization with feedback display
- Timeline navigation through solution attempts
- GA generation evolution visualization with fitness tracking
- A* search tree visualization with open/closed sets
- Side-by-side algorithm comparison mode
- Performance metrics dashboard

The dashboard enables intuitive exploration of algorithm behavior and supports educational understanding of search strategies.

5 Conclusions

This project successfully implements and benchmarks four AI algorithms for the Pokedle game, providing both theoretical insights and practical solutions.

Key Contributions:

1. Correct CSP formulation with dual heuristics outperforms baselines
2. Valid Pokemon-preserving GA avoids common pitfalls of random attribute generation
3. Admissible A* heuristic guarantees optimality with efficient beam search
4. Comprehensive benchmark (1,680 runs) establishes performance baselines
5. Interactive visualization system aids algorithm understanding

Algorithm Recommendations:

- **For accuracy**: CSP with MRV+LCV+AC3 (94.1% success, 5.09 attempts)
- **For speed**: A* with narrow beam (94.3% success, 0.344s)
- **For exploration**: SA with high initial temperature
- **For constraint validity**: GA with Pokemon-based representation

The comprehensive benchmark results and interactive system provide a solid foundation for understanding AI search strategies in constrained guessing games.