Journal on Interactive Systems, 2026, 17:1
ISSN: 2763-7719 • doi: 10.5753/jis.2026.6773

Research Paper

Pokémon GO Team Optimization: A Comparative Study of Classic Metaheuristic Algorithms

Gabriel Bueno Guimarães [Independent Researcher | *gabrielengaut15@gmail.com*]
Rodrigo Colnago Contreras [Federal University of São Paulo | *contreras@unifesp.br*]
Arthur Costa Gorgonio [Federal University of Rio Grande do Norte | *arthur@gorgonio.com.br*]
Anne Magály de Paula Canuto [Federal University of Rio Grande do Norte | *anne@dimap.ufrn.br*]

Department of Science and Technology, Institute of Science and Technology, Federal University of São Paulo (UNIFESP), São José dos Campos, SP, 12247-014, Brazil

Abstract. Pokémon GO team selection can be formulated as a combinatorial optimization problem in which the goal is to generate a three-Pokémon counter-team that maximizes performance against a given rival team under simulated battles. In this study, we establish and evaluate a compact set of classical metaheuristic baselines for this task, namely Simulated Annealing (SA), Tabu Search (TS), Variable Neighborhood Search (VNS), Genetic Algorithm (GA), Ant Colony Optimization (ACO), and Cuckoo Search (CS), using a fixed benchmark of 1,000 rival teams adopted from [da Silva Oliveira *et al*., 2020]. All methods are assessed under the same dataset, fitness function, and simulator, and we report solution quality (fitness) and elapsed time over repeated runs. The results reveal distinct quality and cost regimes among the evaluated baselines. VNS achieves the highest mean fitness under the adopted stopping conditions, whereas the SA variants provide the lowest runtimes with competitive fitness. We further analyze convergence behavior on the hardest rival teams and characterize the local-search effort of VNS, providing evidence on stabilization patterns and per-iteration workload. These findings deliver reproducible optimization baselines and convergence evidence for Pokémon GO team generation, supporting method selection under different computational budgets and providing reference points for future work on faster convergence and hybrid search strategies.

Keywords: Metaheuristic Algorithms; Team Composition Optimization; Pokémon GO; Combinatorial Optimization; Game Analytics

Edited by: Tadeu Moreira de Classe | **Received:** 30 October 2025 • **Accepted:** 02 March 2026 • **Published:** 30 March 2026

1 Introduction

The rapid expansion of the gaming industry, driven by technological advancements, has transformed video games into a multi-billion-dollar market. According to [PwC, 2023], the industry is expected to generate revenues of up to US$312 billion by 2027.

Among the many successful games, Pokémon GO has stood out as a global phenomenon, captivating millions of players since its release in 2016 [Niantic, Inc., 2016]. The game's competitive nature, particularly in battles involving teams of Pokémon, has led players to seek optimal team compositions to maximize their chances of victory. Each Pokémon GO battle consists of teams comprising three Pokémon selected from a pool of 1008 registered creatures, resulting in approximately 166.2 million possible team combinations. Evaluating all potential teams manually is infeasible, which underscores the need for computational methods capable of identifying near-optimal solutions efficiently. This motivates the application of heuristic optimization algorithms, which are well-suited to navigate large and complex solution spaces, as stated in [Almufti, 2025].

Previous studies, such as the works of [da Silva Oliveira *et al*., 2020] and [Reis *et al*., 2023a], have explored the use of Genetic Algorithms (GA) to recommend optimal Pokémon GO teams. Inspired by these works, this study aims to extend the investigation by evaluating and comparing the performance of six meta-heuristic optimization methods in the context of Pokémon GO team composition.

The algorithms considered are Genetic Algorithms (GA) [Gen and Lin, 2023], Simulated Annealing (SA) [Van Laarhoven *et al*., 1987], Tabu Search (TS) [Gomes, 2009], Ant Colony Optimization (ACO) [Dorigo *et al*., 1996], Variable Neighborhood Search (VNS) [Mladenović and Hansen, 1997], and Cuckoo Search (CS) [Yang and Deb, 2009]. It is important to note that the GA was primarily employed in this study to replicate the original work serving as a baseline for comparison.

The goal of this study is to determine which algorithm yields the best performance in terms of team fitness and computational efficiency. By comparing these six approaches, this work seeks to provide insights into the suitability of different metaheuristic techniques for team composition problems, particularly in the context of Pokémon GO. The findings are expected to contribute not only to the gaming community but also to the literature on combinatorial optimization for team composition in competitive games.

To address this challenge and ensure methodological fairness, all six optimization techniques are evaluated under identical experimental conditions using a standardized configuration of 1,000 randomly generated rival Pokémon teams, adopted consistently with those adopted from [da Silva Oliveira *et al*., 2020], with thirty-five repetitions to mitigate stochastic variance. Two evaluation metrics are adopted: (i) the fitness performance of the recommended teams, based on adjusted combat attributes and type effectiveness, and (ii) the average execution time required to pro-

duce solutions.

From the experimental analysis, Simulated Annealing with exponential cooling emerged as the method achieving the highest average fitness results, indicating superior consistency in identifying strategically advantageous Pokémon combinations. Meanwhile, Variable Neighborhood Search achieved the lowest execution time, making it an attractive alternative in scenarios where rapid decision-making is prioritized over maximum performance. Population-based methods such as Genetic Algorithms also delivered competitive fitness values, but at the cost of greater computational effort due to the evaluation of multiple individuals per iteration.

The main scientific contribution of this work consists in establishing a systematic and methodologically sound comparison of classic metaheuristic optimization techniques for the Pokémon GO team composition problem. This comparative perspective clarifies the trade-offs between solution accuracy and computational efficiency, thereby offering practical evidence for selecting adequate techniques depending on application requirements, performance constraints, and available computational resources. Moreover, the findings enhance the understanding of how each algorithm behaves when applied to this specific combinatorial scenario, contributing to the literature on optimization in competitive game contexts. The scope of the contribution is intentionally restricted to combinatorial optimization applied to Pokémon GO, without extending claims beyond optimization-based team generation, ensuring conceptual precision and alignment with the actual outcomes of the study.

The remainder of this paper is organized as follows. Section 2 reviews related works on optimization techniques applied to team composition and competitive game environments. Section 3 presents an overview of the theoretical background of the metaheuristic algorithms investigated in this study. Section 4 describes the proposed methodology, including the experimental setup, dataset, fitness function, and algorithm configurations. Section 5 reports and discusses the experimental results, providing a comparative analysis of solution quality and computational performance. Finally, Section 6 concludes the paper and outlines directions for future work.

2 Related Works

In this section, we present relevant studies in the field of optimization, particularly focusing on heuristic techniques applied to team/deck composition and recommendation in competitive games.

For instance, [da Silva Oliveira *et al*., 2020] proposed an optimization-based approach for Pokémon GO team generation using heuristic optimization techniques. Their study compared three main approaches: Iterated Local Search (ILS), Genetic Algorithms (GA), and Memetic Algorithms (MA). The results indicated that MA achieved the best team fitness scores, although it required more computational resources.

For scenarios demanding a balance between efficiency and accuracy, the authors recommended GA as a suitable alternative. The application of genetic algorithms to team composition in League of Legends was explored in [Costa *et al*., 2019]. The study introduced different fitness functions tailored to three strategic playstyles: poke, engage, and team fight. By running multiple iterations, the authors found that expanding the search space improved the quality of team recommendations, with the best configuration achieving a fitness score of 97.48% after 1000 iterations.

Extending the use of genetic algorithms to card-based competitive games, [Haumann and Höppner, 2025] investigated the adaptivity of deck-building methods in Legends of Code and Magic. Their study compared four recommendation models based on logistic regression, evaluating how well each system adapts to dynamic game scenarios such as the introduction of new cards, balance changes to existing cards, and shifts in opponent behavior. A genetic algorithm was employed to simulate an evolving competitive environment, where players iteratively improved their decks based on recommendation outputs and match outcomes. The authors demonstrated that methods with higher predictive accuracy were generally more robust to environmental changes, although they often required larger amounts of training data to adapt effectively.

Similarly, [Balbinot *et al*., 2024] proposed the use of genetic algorithms to model competitive strategies in Legends of Runeterra. The authors formalized three distinct competitive strategies commonly adopted in the game's metagame and incorporated them into a genetic algorithm framework. Two different fitness functions were designed to evaluate these strategies, each emphasizing distinct performance criteria derived from large-scale match statistics. Their empirical results demonstrated that the proposed approach is capable of generating competitively viable deck lineups. Furthermore, the experimental analysis reported by the authors showed favorable execution times alongside high fitness values, underscoring the practical viability of the approach for real-world competitive scenarios.

The study by [García-Sánchez *et al*., 2016] applies evolutionary algorithms to automatically build Hearthstone decks drawn from 743 unique cards across nine heroes with at most two copies per card and one for Legendaries, enforcing mana constraints that begin at one crystal on turn one and increase by one each round up to ten and incorporating hero-specific abilities that shape overall strategy. To address the game's stochastic nature, the authors evaluated each candidate deck by playing at least 16 matches against a suite of human-designed decks and applied a lexicographical fitness measure that considers deck-construction errors involving illegal duplicates, total wins and the standard deviation of wins across opponents. Limited by computational resources, the experiment focused on two hero classes and in both cases the best evolved decks achieved higher win rates than professional players' builds, demonstrating that the algorithm can autonomously uncover effective strategies without any prior domain knowledge.

The application of genetic algorithms is discussed in [Yang *et al*., 2021] for deck construction in the game Legends of Code and Magic, which features an initial card selection phase consisting of 30 rounds, during which the player chooses one card from three random options. The authors proposed three scoring strategies: a direct approach, an attribute-based evaluation, and a method that also considers

the desired distribution of mana costs. The mana system follows the model used in games such as Hearthstone, where players start with limited mana and gain one additional point each turn. The evaluation was based on the win rate achieved across 50 simulated matches against a fixed opponent. The results indicated that the attribute-based approach achieved the best performance with lower computational cost, while incorporating mana distribution led to more diverse decks without consistent performance improvements. Nevertheless, the generated agents still underperformed compared to top human players and specialized heuristic-based solutions.

Expanding to the universe of other Pokémon-based games with similar mechanics and the Chef's Hat card game, [Barros and Sciutti, 2022] proposed an opponent-aware reinforcement learning framework based on contrastive optimization to enable individualized adaptation in competitive environments. The model designed by the authors, known as WINNE, combines a global policy for general gameplay with opponent specific local policies guided by a contrastive strategy prediction module, allowing the agent to learn and disrupt recurrent opponent behaviors in an online setting. The approach was evaluated in two competitive scenarios, namely a Pokémon duel environment and the multiplayer Chef's Hat card game, demonstrating superior performance compared to standard reinforcement learning, imitation learning, and competitive baselines, particularly when facing the same opponents repeatedly. The results further indicate that although the model may experience a temporary drop in effectiveness after extended periods without encountering the same opponent, it is able to retain learned knowledge over time and effectively outperform specialized agents while accurately predicting opponent actions.

Still focusing on Pokémon battles and leveraging a battle simulation platform, [Sarantinos, 2022] presented an AI agent that employs a short-term look-ahead algorithm to predict the opponent's next moves. In addition, the authors propose a novel approach to estimate the composition and strength of rival teams under uncertainty. The reported results show that the AI agent was able to reach the 33rd position in the world ranking and maintain a stable ELO rating over 643 battles against human players.

The analysis and integration of multiple datasets related to Pokémon were investigated in [Khare *et al*., 2023]. The main objective of that study was to identify which Pokémon characteristics, regardless of Generation or of the specific game in the franchise, are most relevant and how these attributes interact and influence predictive algorithms designed to estimate the probability of victory in 1v1 battles between Pokémon. As a result, the authors found that characteristics such as Pokémon speed and rarity may strongly influence this probability. In addition, the study reported relevant performance metrics for the evaluated machine learning algorithms, including Random Forest, Support Vector Machine, and XGBoost. Among the tested models, the best result achieved an approximate Mean Absolute Error (MAE) of 0.04.

More closely related to the present work, [Reis *et al*., 2023b] introduced an adversarial framework for automated Pokémon team building and metagame balance, applied to the VGC AI Competition template presented in [Reis *et al*., 2021]. The authors compare diverse strategies for team construction, combining genetic algorithms, Nash equilibrium analysis, and deep learning techniques to predict the win rates of pairwise Pokémon and team matchups. The experimental results reported by the authors show that the proposed approach is capable of identifying dominant strategies while preserving metagame diversity.

Unlike the aforementioned studies, our work focuses specifically on recommending optimal Pokémon GO teams using multiple heuristic algorithms, including Simulated Annealing, Tabu Search, Ant Colony Optimization, Variable Neighborhood Search, and Cuckoo Search. Unlike prior approaches, we aim to compare multiple optimization techniques to determine which method provides the best balance between fitness score and execution time, contributing valuable insights to the application of heuristics in game-related optimization problems.

3 Theoretical Background

This section covers the theoretical foundations of the six optimization techniques explored in this paper.

3.1 Genetic Algorithms

Genetic Algorithms (GAs) are optimization techniques that are based on evolution and genetics, as described by [Gen and Lin, 2023]. An initial population of N solutions is first generated, typically in a random manner. These solutions are represented as chromosomes and subsequently evaluated through a fitness function that quantifies the objective to be maximized or minimized [Alhijawi and Awajan, 2024]. Based on the assessment of individuals' fitness within the population, the algorithm initiates an evolutionary process aimed at improving solutions over successive generations. To achieve this, GAs employ genetic operators inspired by the mechanisms of natural selection.

The first of these operators is selection, which is responsible for determining which individuals will be chosen as parents and will have the opportunity to pass on their characteristics to the next generation. There are several parent selection methods such as linear ranking and roulette wheel which probabilistically favor individuals with higher fitness scores.

Once the parents have been selected, GA performs the crossover operation, a recombination mechanism that combines the genetic material of two paired individuals to generate new offspring. Various crossover techniques can be used, each influencing the genetic diversity of the population and the convergence rate of the algorithm in different ways [Kora and Yadlapalli, 2017]. One of the most widely used methods is single-point crossover, in which a cut point is randomly selected along the chromosome and, from this point onward, segments of the two parents are swapped, thus producing two new individuals.

Following the crossover phase, the algorithm proceeds to the mutation stage, which is responsible for enhancing genetic diversity within the population of newly generated individuals by introducing alterations to their genetic makeup. Mutation is applied with a given probability p, indicating the likelihood that an individual will undergo this process. Several mutation strategies have been proposed and employed in genetic algorithms, including binary bit-flipping mutation

and directed mutation. Nevertheless, one of the most widely adopted approaches is the uniform random mutation. This technique consists of randomly selecting a gene from a given chromosome and replacing it with a new value, also randomly sampled from the predefined admissible range for that gene.

An illustrative example of a complete iteration of a Genetic Algorithm is presented in Figure 1. The figure begins with the initial population, represented as a set of binary chromosomes. Individuals are evaluated through a fitness function, and a selection mechanism called roulette wheel is applied to probabilistically choose parents with higher fitness. The selected individuals then undergo crossover via a single-point recombination strategy, generating two new offspring. Subsequently, a mutation operator is applied to a randomly selected gene of one of the offspring, resulting in a modified chromosome. Finally, the new population is formed with the offspring, including any mutations introduced during this generation. This cyclical process continues over multiple generations until a termination criterion is met.

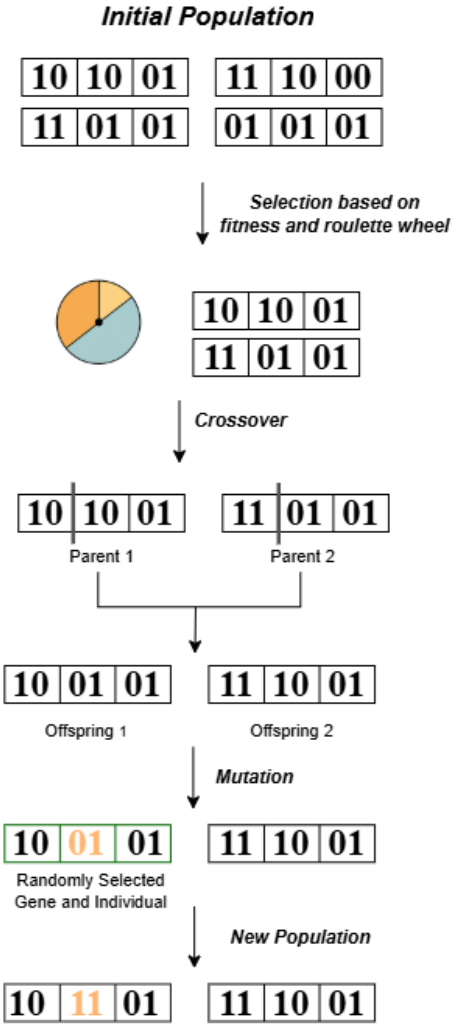

Figure 1. Illustration of a single iteration in the execution flow of a Genetic Algorithm, including selection, crossover, mutation, and formation of the new population.

3.2 Simulated Annealing

Simulated Annealing (SA) is an extension of the Monte Carlo Method, applied to the optimization of complex functions. The method starts with a random solution in the search space and, at each iteration, generates a "neighbor" that corresponds to a modification in the current solution [Delahaye *et al.*, 2018]. SA uses the Metropolis acceptance criterion so that, even if solution j is worse according to the cost function, it can still be accepted, following the probability:

$$P_{\text{accept}}(j \mid i) = \begin{cases} 1, & \text{if } f(j) < f(i), \\ \exp\left(-\frac{f(j)-f(i)}{T}\right), & \text{otherwise} \end{cases} \quad (1)$$

where:

- $f(i)$ and $f(j)$ are the cost functions of the current and neighboring solution, respectively.
- T is the temperature parameter.

When $f(j) < f(i)$, the solution j is always accepted. On the other hand, the acceptance of worse solutions occurs with a probability that decays exponentially as $f(j) - f(i)$ increases. This acceptance criterion originates from the Boltzmann distribution which defines the probability of a given solid being in the state i of energy E_i at the temperature T [Delahaye *et al.*, 2018] and can be defined as:

$$\Pr\{X = i\} = \frac{1}{Z(T)} \exp\left(-\frac{E_i}{k_B\, T}\right), \quad (2)$$

where:

1. $Z(T)$ is the partition function, ensuring the probabilities sum to 1;
2. E_i denotes the energy of state i;
3. k_B is the Boltzmann constant;
4. T is the temperature.

In each iteration of the algorithm, the temperature T is lowered, simulating the annealing process. The rate at which the temperature is decreased directly influences the acceptance probability of worse solutions according to equation (1). Different cooling schedules have been proposed to control this decay, each with a distinct balance between exploration and exploitation:

- **Exponential Cooling**: $T = T_0 \cdot \alpha^k$,
- **Linear Cooling**: $T = T_0 - \beta \cdot k$,
- **Logarithmic Cooling**: $T = \dfrac{T_0}{1 + \gamma \cdot \ln(k+1)}$,

where:

- T_0 is the initial temperature, k is the iteration number,
- α is the exponential cooling rate with $0 < \alpha < 1$,
- β is the temperature decrement for linear cooling,
- γ is the logarithmic cooling factor.

As the temperature decays according to a cooling schedule, the algorithm increasingly favors moves to better solutions, shifting from exploration to exploitation. The algorithm stops when a stopping criterion is met, which is typically a certain temperature threshold or a predefined number of iterations. At that point, the best solution found throughout the run is reported as the approximate global optimum.

3.3 Tabu Search

Tabu Search (TS) is a local search algorithm that prevents cycling and escaping local optima by maintaining a tabu list of previously visited solutions [Gomes, 2009]. The algorithm evaluates neighboring solutions and accepts the best non-tabu solution, unless an aspiration criterion is met, allowing a tabu solution if it yields an improvement over the best-known solution. While TS is particularly effective for combinatorial optimization problems, its reliance on memory structures and the evaluation of multiple neighboring solutions can lead to high computational costs.

3.4 Ant Colony Optimization

Ant Colony Optimization (ACO) [Dorigo *et al.*, 1996] is a population-based metaheuristic based on the behavior of ants. Artificial ants construct solutions based on pheromone trails and heuristic information, with pheromones being updated to reinforce promising paths. Evaporation prevents premature convergence. The balance between pheromone influence and heuristic guidance is controlled by parameters α and β. The probability that an ant chooses the path between nodes i and j is defined as:

$$p_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in N_i}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}}, \tag{3}$$

where:

- p_{ij} is the probability of choosing the path from node i to node j;
- τ_{ij} is the pheromone level on the path between nodes i and j;
- η_{ij} is the heuristic information (e.g., inverse of distance) associated with the path;
- α controls the influence of pheromone levels;
- β controls the influence of heuristic information;
- N_i is the set of neighbors of node i, representing all the possible paths that can be taken from node i;

3.5 Variable Neighborhood Search

Variable Neighborhood Search (VNS) is a local search method that consists of making changes in the neighborhood structure during the search [Mladenović and Hansen, 1997]. The search alternates between perturbing the current solution (shaking), performing a local search, and updating the solution if an improvement is found. The neighborhood structures are ordered, and the algorithm cycles through them to enhance the exploration capability [Hansen *et al.*, 2019]. Let $\mathcal{N}(x) = \{N_1(x), N_2(x), \ldots, N_{k_{\max}}(x)\}$ denote the family of neighborhood structures associated with the current solution x, where $N_k(x)$ denotes the k-th neighborhood of x. The search begins with $k = 1$ and proceeds as follows:

- Shaking: A random solution x' is selected from the k-th neighborhood of x.
- Local Search: A local search is applied starting from x', yielding a new solution x''.
- Move or Not: If x'' is better than the current solution x, the search returns to $k = 1$ with $x = x''$. Otherwise, k is incremented.

3.6 Cuckoo Search

The Cuckoo Search (CS) [Yang and Deb, 2009] is an optimization technique that is based on the behavior of the cuckoo bird. Some of the cuckoo species lay their eggs in the nests of other host birds, removing some of the host's eggs. If the host bird detects the intrusion, it may abandon the nest. In this algorithm, each cuckoo represents a candidate solution, and a new solution is generated using a Lévy flight as follows:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \cdot \text{Lévy}(\lambda), \tag{4}$$

where:

- $\alpha > 0$ is a step size scaling factor;
- Lévy(λ) is a random step drawn from a Lévy distribution with exponent λ.

Poor solutions have a probability $p_a \in [0, 1]$ of being abandoned and replaced by new randomly generated solutions. This combination of Lévy flights and randomization allows CS to perform both local and global searches efficiently.

4 Methodology

In order to select the best solution for the team recommendation problem, a fair comparison of different aspects of each algorithm is required. In this analysis, two main aspects will be evaluated: fitness and execution time.

In this work, we aim to compare six different optimization techniques: Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO), Variable Neighborhood Search (VNS), Cuckoo Search (CS), and Genetic Algorithm (GA). The Genetic Algorithm was included to ensure a fair comparison with the results reported by [da Silva Oliveira *et al.*, 2020].

In our empirical analysis, these six techniques will be evaluated considering the aforementioned aspects. Each algorithm will be applied to the problem of recommending an optimal Pokémon GO team. All methods will operate under the same experimental conditions, using the same dataset, evaluation metrics, and hardware setup, which is an AMD Ryzen 7 5700X processor and 32GB of RAM. The fitness function used to assess the quality of the recommended teams follows the same evaluation criteria proposed by [da Silva Oliveira *et al.*, 2020].

The main goal of our work is to identify the most effective algorithm for team recommendation in Pokémon GO, balancing the trade-off between solution quality and computational cost.

4.1 Algorithm selection rationale

We selected six metaheuristic algorithms—Simulated Annealing (SA), Tabu Search (TS), Ant Colony Optimization (ACO), Variable Neighborhood Search (VNS), Cuckoo Search (CS), and Genetic Algorithm (GA)—to provide a representative and methodologically sound comparison for Pokémon GO team recommendation. This choice was guided by three complementary criteria: (i) *canonical status and maturity*, (ii) *diversity of search dynamics*, and (iii) *relevance and continuity with prior work*.

First, all selected methods are well-established and widely studied metaheuristics with extensive use in combinatorial optimization and game-related optimization tasks. Their maturity offers stable baselines for comparison, reduces methodological ambiguity, and supports reproducibility through well-understood operators and parameterizations. Moreover, as evidenced in the related work, these algorithmic families frequently appear in optimization problems involving large discrete search spaces and team/deck composition, which makes them natural candidates for this study.

Third, selecting multiple classical families facilitates meaningful comparisons against prior literature and supports cumulative science. Closely related studies in competitive games and team/deck building often evaluate at least one of these paradigms (commonly GA and/or local-search-based methods).

Our selection therefore improves external comparability and helps position the results as incremental evidence within an active research line.

Third, selecting multiple classical families facilitates meaningful comparisons against prior literature and supports cumulative science. Closely related studies in competitive games and team/deck building often evaluate at least one of these paradigms (commonly GA and/or local-search-based methods). Our selection therefore improves external comparability and helps position the results as incremental evidence within an active research line.

Finally, our goal is not to exhaust the entire metaheuristic literature, but to compare a compact and representative set of mature algorithms that span complementary search dynamics under the same dataset, fitness function, and evaluation protocol. By controlling these experimental factors, we aim to attribute observed differences in performance and runtime primarily to the search dynamics of each method rather than to changes in data, modeling assumptions, or assessment procedures.

4.2 Proposed Approach and Hyperparameter Configuration

Algorithm 1 outlines the experimental procedure adopted in this study. This work makes use of the same dataset containing 1000 rival teams, each composed of three Pokémon, as used in [da Silva Oliveira *et al*., 2020]. These teams represent challenging opponents against which the optimization algorithms must recommend effective counter-teams.

This is achieved by generating and evaluating multiple team compositions based on a fitness function that measures performance in simulated battles. Each recommended team is tested against all three Pokémon of the rival team, and its fitness is computed by summing the adjusted combat power differences, considering type effectiveness and individual stats. All Pokémon currently available in the game are listed and maintained by the community at Pokémon Game Info[1]. In addition, all Pokémon information, including type, combat power, and attributes, was retrieved from a publicly available API[2].

[1] `https://pokemon.gameinfo.io/pt-br` - Accessed on 22 March 2026

[2] `https://pogoapi.net/` - Accessed on 22 March 2026

By applying each optimization method to this standardized problem setup, we can directly compare their abilities to recommend competitive Pokémon teams under identical conditions.

We use the same 1,000 rival teams as in [da Silva Oliveira *et al*., 2020], each composed of three Pokémon, that were generated through a random process of combination with repetition, enabling the simulation of Pokémon battles across multiple scenarios against a team generated by the algorithm. The optimization process is executed using each of the selected heuristic algorithms. Since these optimization techniques are non-deterministic, the process is repeated thirty-five times for each algorithm to ensure reliable performance assessment. During each execution, fitness scores and execution times are recorded. The overall procedure is outlined in Algorithm 1.

Algorithm 1 Team Recommendation Experiment

```
Load Pokémon database
for each of the 1000 rival teams do
    for each algorithm in {GA, SA, TS, ACO, VNS, CS} do
        for iteration = 1 to 35 do
            Solve the problem using the selected algorithm
            Compute fitness, execution time, and recommended team
        end for
    end for
end for
Return final results for all rival teams
```

Table 1 presents the parameter configurations used for each optimization algorithm. The Genetic Algorithm (GA) follows the recommendations from [da Silva Oliveira *et al*., 2020], particularly for crossover and mutation rates. For the remaining algorithms, key execution parameters, such as the number of iterations and other operational criteria, were kept as similar as possible to maintain comparability. However, certain values, such as temperature, pheromone evaporation rate, number of neighborhood structures, and other algorithm-specific hyperparameters, were empirically determined through preliminary testing.

4.3 Team Composition and Pokemon's Combat Points

In this study, we assume that each team in Pokémon GO consists of three Pokémon, selected based on their attributes and overall effectiveness in battle. It is important to emphasize that, as mentioned before, although Pokémon GO currently features over 1000 registered Pokémon, this study adopts a reduced set of 651 Pokémon. This choice is deliberate and follows the experimental protocol established by [da Silva Oliveira *et al*., 2020; Harris, 2016], whose dataset is reused to ensure methodological consistency and fair comparison. The dataset employed in that work reflects the state of the game at the time of data collection and includes only the Pokémon available in that version of Pokémon GO. Given the 651 available Pokémon in dataset, this results in 65,194,526 possible team combinations, making the selec-

Table 1. Algorithm configuration in the experiments.

Algorithm	Parameters
Genetic Algorithm (GA)	Individuals = 50, Generations = 20, Crossover = 80%, Mutation = 20%, Selection: Biased roulette (top 20%), Stopping criterion: 20 generations
Simulated Annealing (SA)	Cooling functions tested: - Exponential (Exp.), Linear (Lin.), and Multiplicative Logarithmic (Log.), Acceptance criterion: Metropolis rule, Number of iterations varies according to temperature
Tabu Search (TS)	Initial Solution: Random, Stopping Criterion: 50 iterations, Tabu List Size: 100 solutions, Aspiration Criterion: Accepts if fitness is greater than the best global, Neighbor generation based on type (80%) and CP (20%)
Ant Colony Optimization (ACO)	Number of Ants = 50, Iterations = 20, Pheromone Evaporation Rate = 0.1, Probability of choice proportional to relevance in previous battles
Variable Neighborhood Search (VNS)	Maximum Neighborhood Structures = 3, Maximum Iterations = 50, Number of neighbors in local search = 5, Acceptance Criterion: First improvement, Shaking Mechanism: Dynamic replacement of 1 to 3 Pokémon
Cuckoo Search (CS)	Population Size: 10 nests, Maximum Iterations: 50, Nest Abandonment Probability: 25%, Lévy Flight Parameters: Scale step = 1.0, Long jump probability = 30%

tion process highly complex.

Each Pokémon possesses key attributes that directly impact its battle performance. The type of a Pokémon determines its strengths and weaknesses in combat, as Pokémon can have one or two types that influence their effectiveness in battles.

Bulbasaur, for example, is a Grass/Poison type, which grants it resistances against Water, Electric, Fairy, and Fighting-type moves, while making it more vulnerable to Fire, Flying, Ice, and Psychic-type attacks. It is particularly resistant to Grass-type moves, receiving only 25

On the other hand, it is twice as weak to Fire, Flying, Ice, and Psychic-type moves, making encounters with Pokémon of these types more challenging.

For the purpose of this study, we rely on Combat Points (CP) to quantify and compare Pokémon strength. CP serves as a general measure of a Pokémon's overall power, influenced by its individual stats and level.

Although CP is not a deterministic indicator of battle outcomes in Pokémon GO, it is widely adopted to measure the strength of a Pokémon in large-scale analytical and optimization studies due to its ability to aggregate core combat attributes [da Silva Oliveira *et al.*, 2020; Harris, 2016]. It is important to highlight that higher CP does not guarantee victory in all scenarios. Pokémon with lower CP may outperform stronger opponents depending on factors not explicitly modeled here, such as moveset composition, damage-per-second (DPS), energy generation, attack timing, and player's performance.

To partially mitigate this limitation, type effectiveness is incorporated into the fitness evaluation, adjusting CP values to reflect known strengths and weaknesses between Pokémon types and their movesets. Nevertheless, aspects related to specific move builds, damage-per-second (DPS), energy generation, attack timing, and player skill are intentionally excluded from the model to maintain methodological consistency and tractability.

Future work may extend this formulation by incorporating moveset-dependent attributes or build-specific evaluations to better approximate real battle dynamics.

For instance, Exeggtor has a CP of 3014. Its performance is further determined by:

- **Attack (ATK)** – Influences the damage dealt by a Pokémon's moves.
- **Defense (DEF)** – Reduces incoming damage during combat.
- **Stamina (STA)** – Determines total health points (HP), which affect how long a Pokémon can remain in battle before fainting.

Equation 5 illustrates how the Combat Power (CP) is computed by combining the Attack, Defense, and Stamina attributes with level-dependent modifiers, providing a standardized measure of a Pokémon's battle potential. In this study, these modifiers are fixed at 15 and 0.7903, ensuring that all Pokémon are evaluated under identical and optimal conditions. All base attribute values follow the official Pokémon GO game data provided by Niantic, Inc.[3]

[3] `https://pokemongo.com/en-US` - Accessed on 22 March 2026

The value 15 corresponds to the maximum possible Individual Value (IV) assigned to each attribute, resulting in a total IV range from 0 to 45 across all combinations. The value 0.7903 represents the Combat Power Multiplier (CPM) associated with Pokémon at level 40, as defined by the official CP formulation in Pokémon GO [Pokémon GO Wiki, 2024]. The CPM is a level-dependent scaling factor that adjusts base attributes to reflect a Pokémon's effective battle strength.

By fixing all attribute modifier values at their maximum levels and the CPM at the level-40 multiplier, the search space is simplified while preserving methodological consistency across all Pokémon. Although each Pokémon may exhibit different intrinsic base attributes, the CP calculation formula is publicly documented and embedded within the game mechanics, ensuring transparency and reproducibility of the experimental setup.

$$\text{CP} = \frac{(\text{Attack}+15)\cdot\sqrt{(\text{Def}+15)}\cdot\sqrt{(\text{Stam}+15)}\cdot(0.7903)^2}{10} \tag{5}$$

Figure 2 presents a simulated battle between a Charizard and a Exeggutor. Analyzing the attributes of each Pokémon, we can state that the Exeggutor has a superior base-CP value. If the evaluation is conducted solely on the basis of this information, the resulting assessment may be inaccurate, as Pokémon types significantly influence battle outcomes. In the illustrative example, Charizard is a Fire-type Pokémon, whereas Exeggutor is a Grass-type Pokémon. When Charizard performs a Fire-type attack, Exeggutor receives double damage due to type effectiveness. Conversely, when Exeggutor uses a Grass-type attack, Charizard receives only one-quarter (0.25) of the normal damage. Consequently, despite having a lower base Combat Power (CP), Charizard exhibits a high probability of winning this battle, since its attacks benefit from substantially higher effective damage.

Figure 2. Illustration of a battle between Charizard and Exeggtor.

4.4 Representation of Individuals

For this study, the representation of individuals in all algorithms follows the same structure proposed by [da Silva Oliveira *et al.*, 2020]. Each individual is represented as a team composed of three Pokémon, which can be either distinct or repeated within the same team. This means that a valid solution may consist of three different Pokémon or include duplicates, depending on the selection process.

4.5 Fitness

In this work, the fitness function measures the performance of a selected team against predefined rival teams. The calculation considers each Pokémon's individual attributes and the effectiveness of their types relative to their opponents. The final fitness value of a team is determined by summing the individual performances of its members in direct battles against all rival Pokémon.

Formally, the fitness of a team is computed as follows: each Pokémon in the selected team faces each Pokémon in the rival team, and their performances are adjusted by an effectiveness factor ε, which reflects type interactions during combat. This coefficient determines whether a Pokémon has an advantage or disadvantage against its opponent.

For example, if a Grass-type Pokémon battles against a Fire-type Pokémon, the effectiveness coefficient ε would be 0.5, as Grass-type moves are less effective against Fire-types. Conversely, if the Pokémon were Water-type instead, the coefficient ε would be 2.0, as Water-type moves have a super-effective advantage over Fire-types. The formula used to compute a Pokémon's adjusted strength is:

$$CP_{adj} = CP \times \varepsilon. \tag{6}$$

After computing the adjusted values for each individual battle, the team's fitness is determined by summing the differences between the adjusted CP values of the selected Pokémon and the rival Pokémon. Since type effectiveness is applied to both sides, the CP of each Pokémon in the rival team is also adjusted based on the battle matchup. This ensures that the evaluation reflects not only the raw strength of the Pokémon but also the strategic advantages and disadvantages present in the matchups. The fitness value is then computed by considering all possible pairwise battles between the two teams:

$$Fitness_{team} = \sum_{i=1}^{3}\sum_{j=1}^{3}(CP_{adj_{sel,i}} - CP_{adj_{riv,j}}), \tag{7}$$

where:

- $CP_{adj_{sel,i}}$ is the adjusted CP of the i-th Pokémon in the selected team, considering type effectiveness.
- $CP_{adj_{riv,j}}$ is the adjusted CP of the j-th Pokémon in the rival team, also adjusted for type effectiveness.
- The outer sum ($\sum_{i=1}^{3}$) iterates over all three Pokémon in the selected team.
- The inner sum ($\sum_{j=1}^{3}$) iterates over all three Pokémon in the rival team, ensuring that each selected Pokémon is evaluated against every opponent.

4.6 Optimization Operations Across Algorithms

The optimization algorithms employed in this study operate through distinct mechanisms. However, some fundamental processes overlap where one of the key shared operations is the replacement of Pokémon within a team.

4.6.1 Pokémon Replacement

Algorithms such as GA, SA, TS, VNS, and CS incorporate random Pokémon replacement to explore alternative team compositions. This process allows the algorithms to escape local optima and find better team configurations. The replacement mechanism is applied in different ways depending on the algorithm:

- GA and CS: A Pokémon in the team is randomly replaced by another from the dataset, ensuring genetic diversity (GA) or facilitating exploration through Lévy flights (CS).
- SA: A Pokémon is randomly swapped, and the new team's fitness is evaluated. The move is accepted based on the Metropolis criterion, which allows occasional acceptance of weaker teams to avoid stagnation.
- TS and VNS: In TS, Pokémon can be replaced based on type advantage (80%) or Combat Power similarity (20%), while in VNS, the number of replaced Pokémon varies dynamically between one and three.

Figure 3 illustrates the Pokémon replacement process, which is responsible for the exploration component in multiple algorithms.

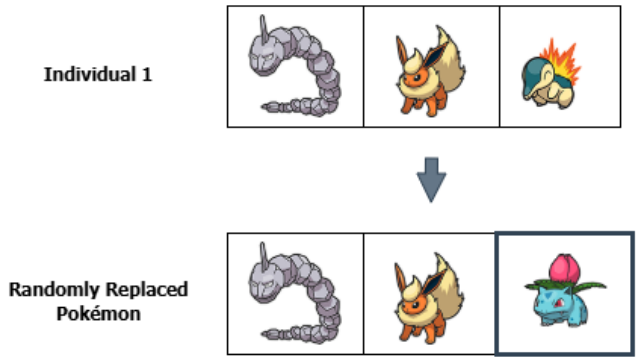

Figure 3. Illustration of the Pokémon replacement process in optimization algorithms.

4.6.2 Individuals Crossover

Crossover is a particular operation applied to GAs that allows the exchange of genetic material between individuals in a population, mimicking the natural process of reproduction. In the context of this study, crossover is applied to Pokémon teams, enabling the combination of characteristics from two parent teams to generate new offspring teams.

The crossover process used in this work follows a one-point crossover strategy, which is a common approach in genetic algorithms. The operation consists of selecting a random position within the team structure and swapping the Pokémon between two parent teams at that point. This ensures genetic diversity while maintaining structural integrity within the generated teams.

Formally, given two parent teams $P_1 = (p_1, p_2, p_3)$ and $P_2 = (q_1, q_2, q_3)$, a crossover point k is randomly chosen between positions 1 and 2. The offspring teams O_1 and O_2 are then generated by swapping the elements after the crossover point:

- If $k = 1$, then:

$$O_1 = (p_1, q_2, q_3)$$
$$O_2 = (q_1, p_2, p_3)$$

- If $k = 2$, then:

$$O_1 = (p_1, p_2, q_3)$$
$$O_2 = (q_1, q_2, p_3)$$

The probability of performing crossover in this study was set to 80%, meaning that in most iterations, parent teams undergo crossover to produce offspring.

Figure 4 illustrates an example of the crossover operation applied to two parent teams.

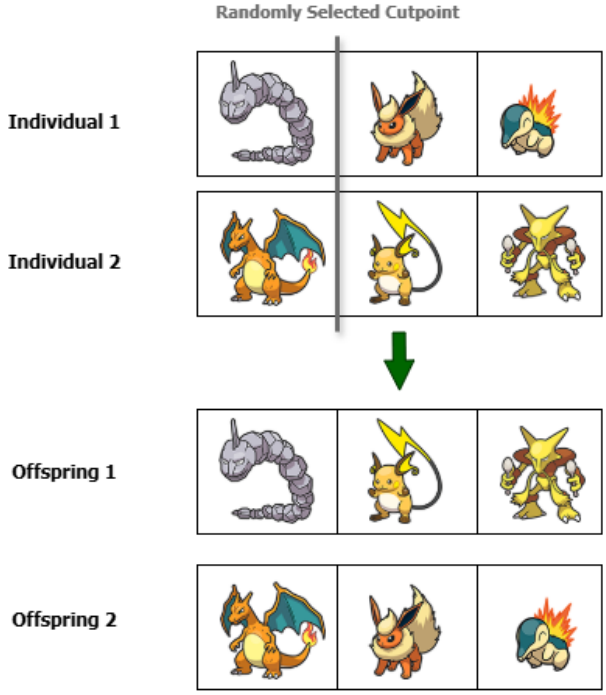

Figure 4. Illustration of the crossover operation in genetic algorithms applied to Pokémon teams.

4.7 Algorithm-specific Search Procedures

Although Section 4.6 defines the core operators used across methods such as Pokémon replacement and crossover, each metaheuristic employs distinct search dynamics. For the sake of clarity and reproducibility, we detail below how each algorithm executes the search process, explicitly defining: (i) what constitutes an iteration, (ii) how candidate teams are generated, (iii) the acceptance/replacement rule, and (iv) the stopping criterion and returned output. All hyperparameters follow Table 1.

4.7.1 Genetic Algorithm (GA)

In GA, one iteration corresponds to one generation. The algorithm starts by initializing a population of 50 individuals, each composed of three randomly selected Pokémon. In each

generation, fitness is evaluated for all individuals, after which parent selection is performed using a biased roulette strategy restricted to the top 20% of the population. One-point crossover is then applied with a probability of 80% to generate offspring, followed by mutation with a probability of 20%, implemented through the Pokémon replacement operator. The next population is subsequently formed from the resulting individuals. The optimization process terminates after 20 generations and returns the best-performing team identified during the search.

4.7.2 Simulated Annealing (SA)

In SA, one iteration corresponds to generating and evaluating one neighbor team. Starting from a random initial team, SA produces a neighbor by applying Pokémon replacement to the current team, as illustrated in Figure 3. After evaluating the neighbor, the algorithm decides whether it should be accepted. To mitigate the risk of convergence to local optima, the Metropolis acceptance criterion is applied to non-improving neighbors. A neighbor solution that represents a fitness decrease may still be accepted with a given probability depending on the current temperature, enabling controlled exploration of the search space. This behavior is gradually changed as the temperature decreases according to the cooling schedule. The algorithm terminates when the stopping condition associated with the cooling schedule is met and returns the best team encountered during the search.

4.7.3 Tabu Search (TS)

In TS, one iteration consists of generating a set of candidate neighbors and moving to the best admissible solution. The search starts from a randomly initialized team and runs for 50 iterations. At each iteration, neighboring teams are generated using the Pokémon replacement operator illustrated in Figure 3. The replacement process follows a policy, in which 80% of the replacements prioritize type-based candidates, while the remaining 20% prioritize CP-based similarity. A tabu list with a fixed capacity of 100 teams is maintained to store recently visited solutions, preventing the search from revisiting previously explored regions of the solution space. An aspiration criterion is adopted to override the tabu restriction when a tabu move yields an improvement over the global best fitness found so far. Once the stopping criterion is met, the algorithm returns the best-performing team identified throughout the entire search process.

4.7.4 Ant Colony Optimization (ACO)

In ACO, one iteration consists of constructing candidate solutions with all ants, evaluating their fitness, and updating the pheromone trails accordingly. In our experiments, the algorithm employs 50 ants and is executed for 20 iterations. Each ant constructs a team by sampling Pokémon according to a probability distribution derived from the current pheromone levels, such that Pokémon associated with higher pheromone intensity at a given team position have a higher probability of being selected.

Once all ants have generated their candidate teams, fitness values are computed using the fitness function defined in Section 4.5. The resulting team fitness is then decomposed into individual Pokémon contributions, reflecting the relative impact of each team member on the overall battle performance. These individual contribution scores are subsequently used to guide pheromone reinforcement: Pokémon that contribute more positively to high-quality teams receive stronger pheromone reinforcement at their corresponding team positions, while less influential Pokémon receive proportionally smaller updates. Prior to reinforcement, pheromone levels undergo uniform evaporation at a rate of 0.1 to prevent premature convergence and encourage exploration. Throughout the optimization process, the algorithm continuously tracks the best-performing team and returns the best solution found over the entire set of iterations.

4.7.5 Variable Neighborhood Search (VNS)

In VNS, one iteration applies a shaking step followed by a local search. We define up to 3 neighborhood structures (Table 1), implemented through a dynamic replacement of 1 to 3 Pokémon during shaking. After shaking, the algorithm performs a local search by exploring up to 5 neighbor candidates per local-search, using Pokémon replacement, and adopting a first-improvement acceptance rule: the search moves as soon as an improving neighbor is found. The procedure runs for up to 50 iterations and returns the best team observed.

4.7.6 Cuckoo Search (CS)

In CS, one iteration consists of generating new candidate solutions from nests, applying replacement if improvement occurs, and abandoning a fraction of nests with probability 25%. We use a population of 10 nests and run for 50 iterations. New candidates are generated by applying a Lévy-flight-inspired perturbation in the discrete team space, implemented as Pokémon replacement with scale step 1.0, and a long-jump probability of 30% . When a generated candidate improves upon its current nest, it replaces it. Additionally, nest abandonment introduces diversification by reinitializing some nests. The algorithm returns the best team found throughout the run.

5 Results and Discussions

This section presents the results of the empirical analysis using various optimization techniques, such as: GA, SA, TS, ACO, VNS, and CS. In the experimental explorations, as mentioned previously, we created a database containing 1000 different Pokémon teams that were randomly generated.

5.1 Overall Results

Tables 2 and 3 present the numerical results of fitness and elapsed time, respectively. Each table describes the type of algorithm (Population or Single) and the algorithm alias. The next column presents the average and standard deviation, and each algorithm's best and worst results are presented in the last two columns. In addition, the best result in each column is highlighted in bold. Meanwhile, the italic results represent the best value in the group.

Table 2 summarizes the fitness results obtained by the evaluated algorithms. Although the logarithmic and exponential variants of Simulated Annealing produced nearly identical outcomes, the Variable Neighborhood Search (VNS) achieved the best overall performance, reaching a mean fitness of $53,886.81 \pm 12,782.24$. Notably, even its worst-case result outperformed the minimum values obtained

by all other methods, highlighting the robustness of VNS across the benchmark.

Before presenting the statistical results, it is important to emphasize that, in order to support the adequacy of the adopted number of repetitions, the sample size was defined *a priori* through a statistical power analysis for a one-way ANOVA with eight groups, assuming a medium effect size (Cohen's $f = 0.25$), significance level $\alpha = 0.05$, and statistical power of 0.80. Under this configuration, 35 independent runs per algorithm were considered sufficient to detect practically relevant differences among the evaluated methods. This choice is particularly relevant in the present study because all analyzed metaheuristics are stochastic, and repeated runs help reduce the influence of random variation on performance estimates.

To verify whether these differences were statistically meaningful, we conducted a one-way ANOVA using the best fitness values as the response variable and the optimization algorithm as an eight-level categorical factor. All runs for each method were pooled (n = 35 per group), and the model compared between-group and within-group variability, to evaluate the performance differences across multiple independent algorithms while avoiding inflated Type-I error rates. Subsequently, a Tukey HSD post-hoc test was applied to identify which algorithm pairs differed significantly, with family-wise error properly controlled.

The ANOVA revealed a significant effect of algorithm on best fitness, $F(7, 279{,}992) = 7036.73$, $p < 0.001$, with a large effect size ($\eta^2 = 0.1496$, Cohen's $f = 0.419$). Tukey's HSD showed that all pairwise comparisons were statistically significant ($p < 0.001$), except GA vs. TS and SA-Exp vs. SA-Log. As expected from the descriptive results, VNS obtained the highest mean performance and differed significantly from every other method.

It is essential to highlight that two main factors contribute to this high standard deviation: i) Pokémon teams that are "difficult" to counter (Pokémons with few or no weaknesses); ii) random teams that have naturally strong Pokémon. Such factors make it harder for the algorithms to find reasonable solutions, and the average fitness value of the individuals is low. In contrast, the highest fitness results were caused by teams with Pokémon with multiple weaknesses, for example, a Bug and Grass-type Pokémon has four times weakness to a Fire-type Pokémon - twice for each individual type and/or a lower CP. Thus, the evaluated algorithms found several teams with very high fitness values.

Table 2. Fitness results.

type	Alg.	Mean ± SD	Best	Worst
Pop.	ACO	38 793.55 ± 10 617.82	104 928.00	7 255.00
	CS	39 579.63 ± 10 415.80	106 842.00	9 370.50
	GA	*48 233.18* ± 12 029.06	***114 249.00***	*19 674.00*
Single	SA – Exp.	49 898.58 ± 12 143.23	***114 249.00***	19 269.00
	SA – Lin.	43 717.90 ± 12 721.62	***114 249.00***	12 136.50
	SA – Log.	49 830.19 ± 12 165.33	***114 249.00***	19 719.00
	TS	48 186.86 ± 11 804.16	109 396.50	18 034.00
	VNS	***53 886.81*** ± 12 782.24	***114 249.00***	***26 776.50***

Table 3 reports the elapsed time results. TS exhibited the longest runtime (1.38 ± 0.03), whereas the SA variants were the fastest, all around 0.05 ± 0.00 (SA-Exp., SA-Lin., and SA-Log.). VNS showed intermediate runtime (0.53 ± 0.03), which is expected because it explores multiple neighborhoods to escape local minima.

In Pokémon team optimization, each neighborhood change implies rebuilding and re-evaluating team performance, increasing computational cost compared to simpler single-solution heuristics. Among population-based algorithms, GA achieved the best runtime (0.09 ± 0.01), followed by CS (0.08 ± 0.00) and ACO (0.37 ± 0.01).

This gap between population-based and single-solution methods is consistent with their computational structure. Population-based algorithms evaluate a set of candidate teams each iteration, whereas single-solution approaches evaluate only one candidate before updating the current solution. As a result, population size directly increases the number of fitness evaluations per iteration, creating a trade-off between computational cost and search breadth.

Table 3. Elapsed time results.

type	Alg.	Mean ± SD	Best	Worst
Pop.	ACO	0.370 ± 0.010	0.360	0.740
	CS	*0.080 ± 0.000*	*0.070*	*0.210*
	GA	0.090 ± 0.010	0.080	0.250
Single	SA – Exp.	***0.050 ± 0.000***	***0.050***	0.090
	SA – Lin.	***0.050 ± 0.000***	***0.050***	0.090
	SA – Log.	***0.050 ± 0.000***	***0.050***	***0.080***
	TS	1.380 ± 0.030	1.280	2.810
	VNS	0.530 ± 0.030	0.430	0.880

5.2 Specific Results

In addition to the overall performance comparisons, this section provides a more in-depth analysis of specific behaviors observed in selected algorithms during the optimization process. Based on the aggregated results in Tables 2 and 3, we highlight VNS, ACO, and SA-Exp because they represent distinct performance regimes in both fitness and execution time. VNS achieved the highest mean fitness ($53{,}886.81 \pm 12{,}782.24$), making it the strongest performer in solution quality and a natural reference for convergence behavior.

In contrast, ACO obtained one of the lowest mean fitness values among population-based methods ($38{,}793.55 \pm 10{,}617.82$) while also presenting the highest execution time within that group (0.37 ± 0.01 s), thus representing a weaker and slower baseline. Finally, SA-Exp was selected as the best among the simulated annealing variants ($49{,}898.58 \pm 12{,}143.23$) while exhibiting the fastest execution time overall (0.05 ± 0.00 s), enabling the analysis of a high-efficiency single-solution method.

Together, these three algorithms span the spectrum of behaviors observed in this study: the top-performing method (VNS), a slower and lower-performing population baseline (ACO), and the strongest SA variant with minimal computational cost (SA-Exp). This selection supports a focused discussion of convergence dynamics and the trade-off between solution quality and computational effort in Pokémon GO team recommendation.

5.3 Convergence Profiles Under Method-Specific Configurations

Figure 5 illustrates the fitness evolution across iterations for five representative rival teams under the stopping conditions adopted in this study. These five teams were selected as the *hardest* instances in our benchmark, defined as the rival teams with the lowest mean fitness achieved by the best-performing algorithm (VNS). Focusing on these cases provides a conservative view of convergence, since they correspond to opponents for which even the strongest method tends to obtain lower-quality solutions on average.

In our protocol, each method is executed until its own termination criterion is met, which means the traces may span different iteration ranges across algorithms. A shorter curve therefore does not indicate premature interruption, but rather that the method reached its predefined stopping condition. Iteration counts are not directly comparable across metaheuristics because each algorithm defines an iteration differently and, crucially, each iteration can entail a different number of fitness evaluations. In our setup, SA-Exp follows a single-solution scheme, evaluating approximately one candidate team per iteration; thus, running SA-Exp for 1,000 iterations remains computationally inexpensive.

In contrast, VNS has a higher per-iteration workload because it explicitly explores multiple neighborhood structures and performs local search within each one. With a maximum of $X = 3$ neighborhood structures and a local search evaluating up to $Y = 5$ neighbors per neighborhood under a first-improvement criterion, a single VNS iteration can require on the order of $X \times Y = 15$ candidate evaluations (in addition to the shaking step, which dynamically replaces 1 to 3 Pokémon to diversify the search). Therefore, although VNS traces may span fewer iterations (up to 50 in our configuration), each VNS iteration represents substantially more search effort than an SA iteration. A similar rationale applies to population-based methods (e.g., ACO), whose iterations evaluate multiple candidates and whose cost scales with the number of constructed solutions.

Even under these method-specific configurations and different per-iteration workloads, a consistent pattern emerges across all cases. VNS reaches high-quality solutions quickly and then stabilizes, whereas SA-Exp exhibits a more gradual improvement over a longer horizon. In contrast, ACO improves early but tends to plateau at lower fitness levels compared to VNS and the later stages of SA-Exp.

Overall, the convergence profiles reinforce that VNS achieves superior mean fitness through rapid stabilization at high fitness levels, while SA-Exp is attractive under strict computational constraints due to its minimal runtime, even if it may not match VNS in final fitness under the adopted stopping conditions. Fairness in our comparison is ensured by keeping the formulation fixed (same dataset, fitness function, and simulator) and by reporting both solution quality and elapsed time, which provides a comparable basis for assessing the trade-off between fitness and computational effort.

5.3.1 VNS Local-Search Effort and Neighborhood Dynamics

Figure 6 reports the average number of local-search attempts performed by VNS per iteration with the shaded region indicating the standard deviation across runs/instances. Two regimes are evident. The first iteration exhibits a markedly higher effort, followed by a sharp drop and a stable plateau from approximately iteration 2 onward.

This behavior is consistent with VNS's mechanics under a first-improvement acceptance rule. At the beginning of the search, the incumbent solution is far from a locally good region, so the algorithm tends to explore more candidates within the local-search phase before finding an improving move. Once VNS reaches a higher-quality basin, improvements become easier to trigger under first-improvement, and the algorithm typically accepts a move after evaluating fewer candidates, which explains the rapid decrease in local-search effort.

After this transient, the average number of local-search attempts stabilizes around a nearly constant level across the remaining iterations. This plateau suggests that VNS quickly settles into a steady operational pattern in which (i) the shaking step (dynamic replacement of 1 to 3 Pokémon) consistently produces perturbations of comparable difficulty, and (ii) the local-search phase requires a similar number of neighbor evaluations to locate the first improving move across iterations.

6 Conclusions

This work investigated Pokémon GO team selection as a combinatorial optimization task, where the goal is to construct a three-Pokémon counter-team that maximizes performance against a given rival team under simulated battles. Using a fixed dataset of 1,000 rival teams adopted from [da Silva Oliveira *et al*., 2020] and a standardized evaluation protocol, we compared six classical metaheuristics (SA, TS, VNS, GA, ACO, and CS) and quantified their trade-offs in solution quality (fitness) and computational cost.

Our first contribution is a compact and reproducible set of optimization baselines for Pokémon GO team generation under a consistent problem formulation. The experimental results identify VNS as the strongest baseline in mean fitness under the adopted stopping conditions, while the SA variants provide highly efficient baselines with minimal runtime. These outcomes offer practical reference points for selecting an optimization strategy depending on whether the priority is solution quality or latency.

Our second contribution is an instance-level analysis that complements aggregated statistics. By focusing on the hardest rival teams (those with the lowest mean fitness under VNS) and analyzing convergence profiles, we provide evidence on how different search dynamics behave in difficult cases. In addition, the VNS effort characterization shows a pronounced initial refinement phase followed by a stable plateau in local-search attempts per iteration, consistent with first-improvement acceptance and the shaking mechanism employed.

Overall, rather than proposing a new model, we contribute empirically validated baselines and convergence ev-

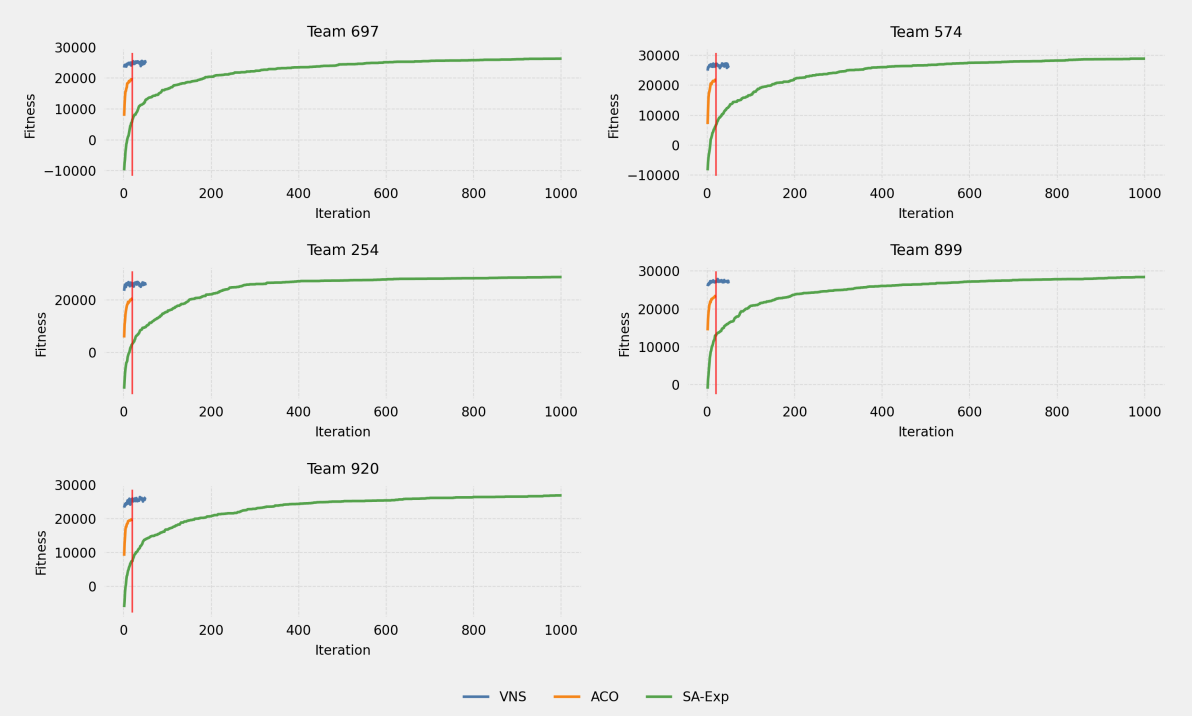

Figure 5. Fitness evolution across iterations for the five hardest rival teams, defined as those with the lowest mean fitness under VNS.

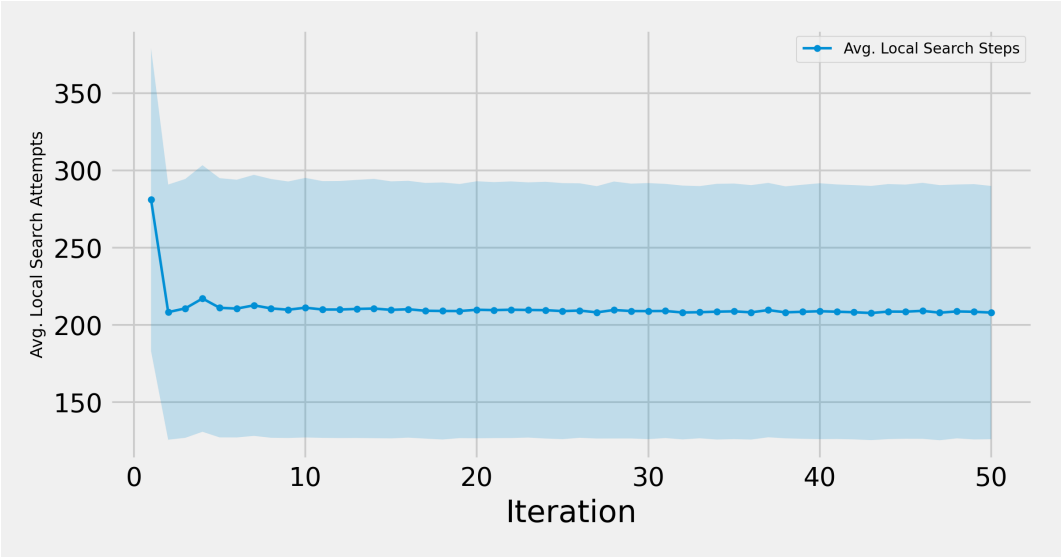

Figure 6. Average Local-Search Steps per Iteration in VNS

idence that can be reused as optimization components in Pokémon GO team-generation workflows, and as reference points for future studies aiming at faster convergence, budget-aware stopping rules, hybrid metaheuristics, and further improvements via parameter tuning. In particular, future work may extend the experimental parameterization by analyzing how tuning choices affect performance, e.g., replacing the GA parent-selection scheme with elitism.

Declarations

Authors' Contributions

Conceptualization, A.C.G, G.B.G, R.C.C and A.M.d.P.C.; methodolgy, G.B.G.; software, G.B.G, A.C.G; validation, A.C.G, G.B.G, R.C.C and A.M.d.P.C.; formal analysis, A.C.G, G.B.G, R.C.C and A.M.d.P.C.; investigation, G.B.G, R.C.C and A.C.G.; writing - original draft preparation - A.C.G, G.B.G and A.M.d.P.C.; writing - review and editing, A.C.G, G.B.G, R.C.C and A.M.d.P.C.; supervision - R.C.C.; project administration, R.C.C. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Acknowledgement

We gratefully acknowledge the grants provided by the Brazilian agencies: "Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES)" and "The State of São Paulo Research Foundation (FAPESP)", respectively through the processes 2019/21464-1 (FAPESP - RCC) and Finance Code 001 (CAPES).

Availability of data and materials

The datasets and source code used in this study are available at `https://github.com/GabrielBueno21/pokemon-go-team-optimization`

References

Alhijawi, B. and Awajan, A. (2024). Genetic algorithms: Theory, genetic operators, solutions, and applications. *Evolutionary Intelligence*, 17(3):1245–1256. DOI: https://doi.org/10.1007/s12065-023-00822-6.

Almufti, S. (2025). *Metaheuristics algorithms: Overview, applications, and modifications*. Deep Science Publishing India. https://doi.org/10.70593/978-93-7185-454-2.

Balbinot, R. A., Costa, L. M., Souza, A. C. C., and Souza, F. C. M. (2024). Developing competitive strategies for legends of runeterra using genetic algorithm. In *Simpósio Brasileiro de Jogos e Entretenimento Digital (SBGames)*, pages 513–531. SBC. DOI: https://doi.org/10.5753/sbgames.2024.241320.

Barros, P. and Sciutti, A. (2022). All by myself: Learning individualized competitive behavior with a contrastive reinforcement learning optimization. *Neural Networks*, 150:364–376. DOI: https://doi.org/10.1016/j.neunet.2022.03.013.

Costa, L. M., Souza, A. C. C., and Souza, F. C. M. (2019). An approach for team composition in league of legends using genetic algorithm. In *2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 52–61. IEEE. DOI: https://doi.org/10.1109/sbgames.2019.00018.

da Silva Oliveira, S., Silva, G. E. P. L., Gorgônio, A. C., Barreto, C. A., Canuto, A. M., and Carvalho, B. M. (2020). Team recommendation for the pokémon go game using optimization approaches. In *2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 163–170. IEEE. DOI: https://doi.org/10.1109/sbgames51465.2020.00030.

Delahaye, D., Chaimatanan, S., and Mongeau, M. (2018). Simulated annealing: From basics to applications. In *Handbook of metaheuristics*, pages 1–35. Springer. DOI: https://doi.org/10.1007/978-3-319-91086-4_1.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE transactions on systems, man, and cybernetics, part b (cybernetics)*, 26(1):29–41. DOI: https://doi.org/10.1109/3477.484436.

García-Sánchez, P., Tonda, A., Squillero, G., Mora, A., and Merelo, J. J. (2016). Evolutionary deckbuilding in hearthstone. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE. DOI: https://doi.org/10.1109/cig.2016.7860426.

Gen, M. and Lin, L. (2023). Genetic algorithms and their applications. In *Springer handbook of engineering statistics*, pages 635–674. Springer. DOI: https://doi.org/10.1007/978-1-4471-7503-2_33.

Gomes, A. (2009). Uma introdução à busca tabu. Technical report, Instituto de Matemática e Estatística, Universidade de São Paulo.

Hansen, P., Mladenović, N., Brimberg, J., and Pérez, J. A. M. (2019). *Variable neighborhood search*. Springer.

Harris, M. D. (2016). Pokémon go post-evolution cp: A model. Quantitative Archaeology blog.

Haumann, T. and Höppner, F. (2025). Adaptivity of card recommendation systems for legends of code and magic. *IEEE Transactions on Games*. DOI: https://doi.org/10.1109/cog60054.2024.10645596.

Khare, S., Callejo, S., and Espinoza, M. (2023). Pokemon team predictions: Which characteristics of a pokemon give the best winning rate and can they be used for prediction? Graduate-level course project, Florida State University, not peer-reviewed.

Kora, P. and Yadlapalli, P. (2017). Crossover operators in genetic algorithms: A review. *International Journal of Computer Applications*, 162(10). DOI: https://doi.org/10.5120/ijca2017913370.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100. DOI: https://doi.org/10.1016/s0305-0548(97)00031-2.

Niantic, Inc. (2016). Pokémon go. https://www.pokemon.com/br/app/pokemon-go. Acesso em: 22 mar. 2026.

Pokémon GO Wiki (2024). Combat power. `https://pokemongo.fandom.com/wiki/Combat_Power`. Accessed: 25 March 2026.

PwC (2023). Perspectives and insights: Global entertainment and media outlook 2023–2027. `https://www.pwc.com/gx/en/industries/tmt/media/outlook/insights-and-perspectives.html`, Acesso em: 22 mar. 2026.

Reis, S., Novais, R., Reis, L. P., and Lau, N. (2023a). An adversarial approach for automated pokémon team building and meta-game balance. *IEEE Transactions on Games*. DOI: https://doi.org/10.1109/tg.2023.3273157.

Reis, S., Novais, R., Reis, L. P., and Lau, N. (2023b). An adversarial approach for automated pokémon team building and metagame balance. *IEEE Transactions on Games*, 16(2):365–375. DOI: https://doi.org/10.1109/tg.2023.3273157.

Reis, S., Reis, L. P., and Lau, N. (2021). Vgc ai competition-a new model of meta-game balance ai competition. In *2021 IEEE Conference on Games (CoG)*, pages 01–08. IEEE. DOI: https://doi.org/10.1109/cog52621.2021.9618985.

Sarantinos, N. R. (2022). Teamwork under extreme uncertainty: Ai for pokemon ranks 33rd in the world. *arXiv preprint arXiv:2212.13338*. DOI: https://doi.org/10.48550/arXiv.2212.13338.

Van Laarhoven, P. J., Aarts, E. H., van Laarhoven, P. J., and Aarts, E. H. (1987). *Simulated annealing*. Springer.

Yang, X.-S. and Deb, S. (2009). Cuckoo search via lévy flights. In *2009 World congress on nature & biologically inspired computing (NaBIC)*, pages 210–214. Ieee. DOI: https://doi.org/10.1109/nabic.2009.5393690.

Yang, Y.-J., Yeh, T.-S., and Chiang, T.-C. (2021). Deck building in collectible card games using genetic algorithms: A case study of legends of code and magic. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–07. IEEE. DOI:

https://doi.org/10.1109/ssci50451.2021.9659984.