Manta Ray Foraging Optimization: A Reimplementation and Analysis

Vedant Tolia
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
vt5285@rit.edu

Landon Spitzer
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
lbs9440@rit.edu

Max Frohman
Department of Computer Science
Rochester Institute of Technology
Rochester, NY 14623
mbf1102@rit.edu

Abstract

In this paper, we discuss our efforts to implement both the Manta Ray Foraging Optimization algorithm, designed by Zhao et al., 2019, and the Improved Manta Ray Foraging Optimization algorithm, designed by Qu et al., 2024. Both algorithms were successfully implemented in Python, which allowed for the testing of both on several benchmark test suites and a real-world combinatorial problem. We compare our results to other common metaheuristics and previous papers to understand the efficacy of MRFO/IMRFO and attempt to validate the claims presented by the previous papers. Our results are a mixed bag, showing that IMRFO is generally better than other metaheuristics, but not without weak spots. We end with a brief discussion of our results and next steps for IMRFO.

1 Introduction

The Manta Ray Foraging Optimization algorithm (MRFO), introduced in 2019 by Zhao et al. [1], translates the coordinated foraging behaviors of manta rays into a population-based optimization algorithm. The algorithm is built upon three primary movement strategies:

- *Chain Foraging*: individuals follow the leader to a food source;
- *Cyclone Foraging*: the population spirals inwards in a coordinated sweep;
- *Somersault Foraging*: individuals flip around the current best position to explore regions nearby.

Across many different engineering problems, MRFO performed competitively against established baselines, drawing interest in the algorithm itself and potential improvements to it.

Though the original MRFO algorithm demonstrated promising results, it was revealed to be vulnerable to slow and premature convergence, and be prone to getting trapped in local optima, specifically in high-dimensional multi-modal landscapes. In order to address these issues, Qu et al. proposed the Improved Manta Ray Foraging algorithm (IMRFO) in 2024 [2], which alters the baseline algorithm with three main enhancements:

- *Tent Chaotic Mapping*: used as an initialization strategy to improve population diversity;

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

- *Bidirectional Search*: strategy used in order to simultaneously attract individuals toward the best-known solution while repelling them from the worst;
- *Lévy Flight*: used to form perturbations that introduce occasional large random steps to escape local optima.

Evaluated against nine competing algorithms on 23 classical benchmark functions, the CEC2017 and CEC2022 benchmark suites, and several engineering design problems, IMRFO was reported to outperform all baselines across the majority of test cases.

The original MRFO implementation by Zhao et al. is written in MATLAB, and as far as we know, no Python implementation of IMRFO has been made publicly available. This limits accessibility for researchers and developers who work primarily in Python. Beyond accessibility, reproducibility is a known challenge with metaheuristic comparisons, small differences in implementation, parameter choices, or initialization can meaningfully change results, making independent replication important for validating the claims made in these papers.

In this work, we provide a Python implementation of both MRFO and IMRFO, translated from the original algorithmic descriptions and evaluated against their reported benchmarks. Both algorithms are built into a shared interface with several baseline optimizers including Particle Swarm Optimization (PSO) [3], Grey Wolf Optimizer (GWO) [4], and the Genetic Algorithm (GA), among others. We evaluate performance across the 23 classical benchmark functions from Yao et al. [5], examine two of our own variants of IMRFO, and apply all algorithms to a practical combinatorial problem, *three-dimensional cargo container packing*.

2 Related Work

Many population based algorithms have been produced in the past few decades within the field of biology-inspired metaheuristic optimization. These algorithms include but are not limited to Particle Swarm Optimization (PSO) [3], Grey Wolf Optimizer (GWO) [4], and Genetic Algorithms (GA). Qu et al. benchmark IMRFO against several of these, including Black Widow Optimization (BWO), Chimp Optimization (ChOA), and the Slime Mould Algorithm, and report that IMRFO outperforms all of them across the majority of test cases [2]. The original MRFO paper by Zhao et al. performs a similar comparison against PSO, GWO, and other optimizers [1].

The benchmark suites used to evaluate these algorithms are also well established in this bio-inspired metaheuristic optimization field. The 23 classical functions from Yao et al. remain the standard baseline for measuring convergence and the ability to avoid local optima across both uni-modal and multi-modal problems [5]. The CEC2017 [6] and CEC2022 [7] suites provide more challenging and standardized evaluations with augmented functions meant to stress-test modern optimizers. MRFO and IMRFO are evaluated on these suites in their respective papers, and we use the same suites to gauge the results in regards to reproducibility in this study.

3 Problem Description

Currently, the MRFO and IMRFO algorithms present two significant problems for the computing community. The first is that Zhao et al. nor, even more so, Qu et al., provide sufficient evidence to demonstrate the success of MRFO/IMRFO. There exists no reproducibility studies in the literature to corroborate their claims, and the authors have not made data available to see all of the results from the papers. However, reproducing the results of either paper is not trivial. The second issue at hand is that Zhao et al. include only a MATLAB implementation of the Manta Ray Foraging Optimization algorithm, and no publicly-available implementation of IMRFO is known to exist at this time. Without accessible implementations of these algorithms, independent researchers and programmers will have greater difficulty both confirming their efficacy and deploying the algorithms in their own work

4 Proposed Solution

We aim to solve both problems with MRFO/IMRFO. To begin, we have reproduced both MRFO and IMRFO according to the specificied algorithms by Zhao et al. (2019) [1] and Qu et al. (2024) [2],

respectively. Our implementations are in Python, which makes them accessible to both the academic community and metaheuristic users in industry.

We then ran our new implementations, alongside additional standard metaheuristic algorithms and two author-created variants of IMRFO, on several benchmarks, described below in Section 5.1. We also tested our set of algorithms on a simplified real-world logistics problem, three-dimensional cargo container packing (also described in Section 5.1).

Our results allow us to analyze MRFO and IMRFO performance across a variety of problems, dimensionalities, and modalities. In the following sections, we discuss our experiment, its results, and how our results compare to the prior works mentioned above.

Beyond replicating the experiments reported in Qu et al. [2], we developed two original variants of IMRFO, each targeting a distinct structural weakness identified during our reproducibility study. The first variant, DV-IMRFO, addresses IMRFO's exploration failure by introducing a diversity-adaptive Lévy magnitude that prevents the algorithm's escape mechanism from collapsing as the population converges. The second, NT-IMRFO, addresses IMRFO's exploitation failure by replacing the single global attractor in chain foraging with a ring-topology neighbourhood best, and augmenting the final search phase with coordinate-wise golden-section refinement. Both variants are described in full below.

4.1 DV-IMRFO — Diversity-Valve IMRFO (Exploration Variant)

4.1.1 Motivation

IMRFO's Lévy somersault step is defined as:

$$x_i^{t+1} = x_{\text{som}} + \alpha \cdot \text{levy} \cdot |x_{\text{som}} - x_{\text{best}}| \tag{1}$$

where x_{som} is the tentative somersault position and x_{best} is the current global best. The magnitude term $|x_{\text{som}} - x_{\text{best}}|$ represents the distance between an agent's post-somersault position and the global best. As the population converges toward x_{best}, this distance shrinks monotonically toward zero. Consequently, the Lévy jump — designed to be the primary escape mechanism from local optima — becomes negligible at exactly the moment stagnation is most severe. This constitutes a structural self-defeating loop: convergence progressively disables the only mechanism capable of reversing it.

4.1.2 Design

DV-IMRFO makes one surgical modification to IMRFO: the Lévy magnitude term $\alpha \cdot |x_{\text{som}} - x_{\text{best}}|$ is replaced by a diversity-adaptive scale $\sigma(t)$, giving:

$$x_i^{t+1} = x_{\text{som}} + \text{levy} \cdot \sigma(t) \tag{2}$$

where the adaptive scale is defined as:

$$\sigma(t) = (\text{ub} - \text{lb}) \cdot \delta(t) \cdot \sqrt{1 - \frac{t}{T}} \tag{3}$$

The three multiplicative components each serve a distinct role. The term $(\text{ub} - \text{lb})$ normalises the step to the actual search space extent, making the behaviour scale-invariant across benchmark functions whose fitness ranges differ by many orders of magnitude. The term $\sqrt{1 - t/T}$ provides a square-root temporal decay, ensuring the amplitude still contracts late in the run but far more slowly than the linear collapse produced by Equation 1.

The central component is the diversity factor $\delta(t)$, computed from the coefficient of variation (CV) of the current population's fitness values:

$$\text{CV}(t) = \frac{\sigma_{\text{fit}}}{|\mu_{\text{fit}}| + \varepsilon} \tag{4}$$

$$\delta(t) = \delta_{\min} + (1 - \delta_{\min}) \cdot \exp\left(-\frac{\text{CV}(t)}{\tau}\right) \tag{5}$$

where σ_{fit} and μ_{fit} are the standard deviation and mean of the population fitness, ε prevents division by zero, τ is the stagnation threshold, and $\delta_{\min}$ is the minimum Lévy scale. The CV is dimensionless

and therefore scale-invariant — the same threshold τ is effective regardless of whether the fitness landscape has values near 10^{-20} or 10^{5}. When CV $\ll \tau$ (population stagnated), $\delta(t) \to 1$ and jumps are large. When CV $\gg \tau$ (population healthy), $\delta(t) \to \delta_{\min}$ and jumps remain small, preserving exploitation.

4.1.3 Elite-Preserving Tent Re-Seeding

When CV$(t) < \tau$ persists for p_{wait} consecutive iterations, stagnation is considered confirmed rather than transient. At this point, the worst ρ fraction of agents are re-initialised using the tent chaotic map (consistent with IMRFO's initialisation), which distributes new positions more uniformly across the search space than pseudo-random sampling. Before re-seeding, the top K elite agents are frozen. After re-seeding, if any frozen agent remains superior to its re-seeded counterpart, the frozen position is restored. This prevents the diversity injection from destroying solutions already discovered. The stagnation counter resets to zero following each re-seed event.

All remaining components of IMRFO — chain foraging (Eq. 1–2), cyclone foraging (Eq. 3–6), bidirectional search (Eq. 10–11), and tent initialisation (Eq. 8–9) — are preserved without modification. The four implementation corrections identified in our reproducibility study are retained in full.

4.2 NT-IMRFO — Neighbourhood Topology IMRFO (Exploitation Variant)

4.2.1 Motivation

In IMRFO's chain foraging, every agent in the population is attracted toward the single global best position x_{best}:

$$x_i^{t+1} = x_i^t + r_v \cdot \left(x_{i-1}^t - x_i^t\right) + a_v \cdot \left(x_{\text{best}} - x_i^t\right) \tag{6}$$

Because the pull toward x_{best} is applied uniformly to all N agents, the entire population collapses into a tight neighbourhood of a single point by mid-run. No structural population diversity remains: every agent performs redundant search in the same region with the same attractor. On functions with a smooth, precise basin near the optimum — such as the Rosenbrock valley, Hartmann landscapes, and Shekel basins — this collapse prevents accurate local refinement. The algorithm identifies the correct broad region but cannot walk the basin geometry precisely, because all agents are occupied with the same coarse pull.

4.2.2 Design

NT-IMRFO replaces the global attractor in chain foraging with a ring-topology neighbourhood best (x_{lbest}). At each iteration, agents are sorted by fitness. Agent at sorted rank i is attracted toward the best agent within its ring neighbourhood $\{i - \lfloor k/2 \rfloor, \ldots, i + \lfloor k/2 \rfloor\} \pmod N$, where $k = \max(3,\, \lfloor N/10 \rfloor)$:

$$x_i^{t+1} = x_i^t + r_v \cdot \left(x_{i-1}^t - x_i^t\right) + a_v \cdot \left(x_{\text{lbest}}^{(i)} - x_i^t\right) \tag{7}$$

The sorting step is essential and non-trivial. Without it, adjacent indices in the population array share no meaningful property — they are neighbours only by initialisation order. After sorting by fitness, rank-adjacent agents are similar in quality, and $x_{\text{lbest}}^{(i)}$ is a genuinely productive local attractor. This construction creates multiple simultaneous exploitation basins: top-ranked agents refine the best region found; mid-ranked agents exploit a related but distinct region; lower-ranked agents explore further afield. The population retains structural diversity and is never fully collapsed to a single attractor.

4.2.3 Adaptive Somersault Factor

IMRFO applies a fixed somersault factor $s = 2$ throughout the run. NT-IMRFO replaces this with an exponentially decaying schedule:

$$s(t) = 2 \cdot \exp\left(-2\,\frac{t}{T}\right) \tag{8}$$

This decays from $s = 2$ at $t = 0$ to $s \approx 0.27$ at $t = T$. Early iterations produce wide somersaults that cover a broad neighbourhood of x_{best}; late iterations produce tight somersaults suited to precise local convergence. The exponential form is deliberately faster than the square-root decay used in DV-IMRFO, consistent with the exploitation focus of this variant.

4.2.4 Golden-Section Coordinate Refinement

In the final ρ_{refine} fraction of iterations, NT-IMRFO applies a coordinate-wise golden-section search (GSS) to the current global best agent following the main population update. GSS is the theoretically optimal interval-reduction method for smooth unimodal one-dimensional functions: it reduces the search bracket by the golden ratio $\varphi = (\sqrt{5} - 1)/2 \approx 0.618$ per evaluation, requires no gradient, and cannot be improved upon for its problem class [**?**]. Applied per-coordinate, it requires exactly $2 \times n_{\text{steps}}$ evaluations per coordinate refined. With $n_c = 3$ coordinates per iteration and $n_{\text{steps}} = 4$ steps per coordinate, the overhead is 12 additional function evaluations per iteration — negligible relative to the $2N$ evaluations required by the population update.

The bracket for each coordinate j is set to $[x_{\text{best},j} \pm 0.25\,(\text{ub}_j - \text{lb}_j)]$, tightening the search to the local geometry near the current best rather than the full coordinate range. This refinement phase directly targets the precision failures observed on low-dimensional functions where IMRFO was outperformed by GWO and Chimp: Branin (2D), Goldstein-Price (2D), Hartmann 3 (3D), and the Shekel family (4D), all of which have smooth, well-defined basins amenable to coordinate-wise polishing.

All remaining components of IMRFO — cyclone foraging, bidirectional search, tent initialisation, and the Lévy somersault formula — are preserved without modification. The four implementation corrections identified in our reproducibility study are retained in full.

5 Experimental Design and Results

5.1 Experiment Design

All of our implementations were tested on three benchmark test suites: 23 classical benchmarks [5], CEC2017 [6], and CEC2022 [7]. We reimplemented Yao et al.'s 23 classical benchmarks, all of which our available in our repository (see Section 7). These functions were tested for all 13 metaheuristics we considered (MRFO, IMRFO, our two variant IMRFO algorithms, and the eight comparison algorithms). Each complete test suite was run a total of 30 times per algorithm for statistical power.

We also tested our algorithms on a three-dimensional cargo container-packing problem. For tractability, we reduced the problem to maximizing boxes (rectangular prisms) packed into a single cargo container (cube). Three benchmark test scenarios were established:

1. *Small Quantity of Boxes*: 15 boxes in a container with side ranges from 3.0 to 6.0 units, and an expected total volume of approximately 1,370 cubic units;
2. *Medium Quantity of Boxes*: 25 boxes in a container with side ranges from 2.5 to 5.0 units, and an expected total volume of approximately 1,320 cubic units;
3. *Large Quantity of Boxes*: 40 boxes in a container with side ranges from 2.0 to 4.0 units, and an expected total volume of approximately 1,080 cubic units.

Boxes were generated randomly for each of the three scenarios using a set seed (for consistency across algorithms and runs). The container itself had a constant size of 1,000 cubic units. The expectation was that maximizing packing would increase with the quantity of boxes, though overflow (the ratio of total volume to container volume) did decrease. Each algorithm was run five times per scenario.

5.2 Results

5.2.1 Benchmarks

Our results indicated mixed performance for both MRFO and IMRFO. To perform easy comparison, Wilcoxon tests were performed between IMRFO against all other algorithms for the 23 Yao et al. benchmarks (see Figure 1). In all cases, the Sailfish Optimizer (SO), the Slime Mold Algorithm (SMA), and the Sunflower Optimizer (SFO) metaheuristics fail to outperform IMRFO. However, some other algorithms, particularly Particle Swarm Optimization (PSO), Black Widow Optimization (BWO), and the Chimp Algorithm (Chimp) perform better, with IMRFO failing Wilcoxon tests against

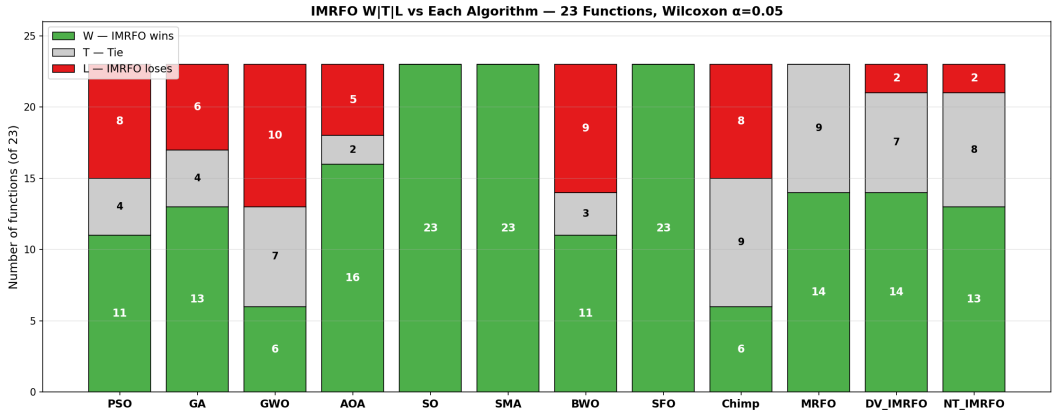

Figure 1: Comparisons of IMRFO with all other algorithms in Wilcoxon tests for each of the Yao et al. benchmark functions, with $\alpha = 0.05$. Green means that IMRFO wins, gray represents a tie, and red means that IMRFO loses the test.

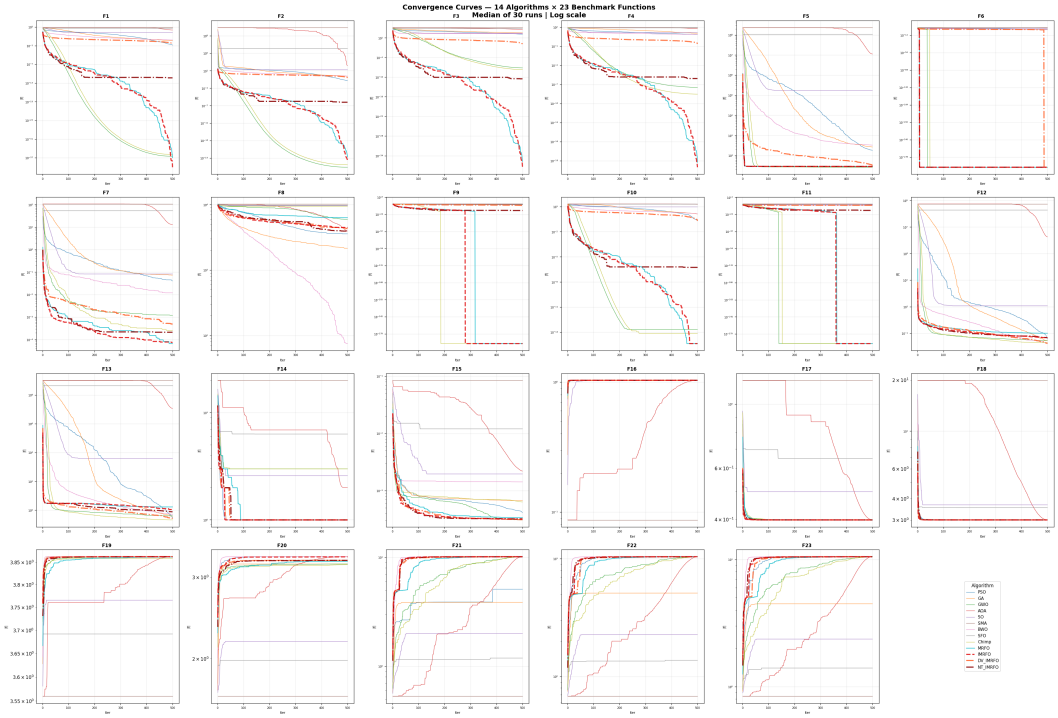

Figure 2: Convergence plots for all 23 of Yao et al.'s functions [5]. MRFO is cyan, IMRFO is dashed red, DV-IMRFO is dashed orange, and NT-IMRFO is dashed maroon.

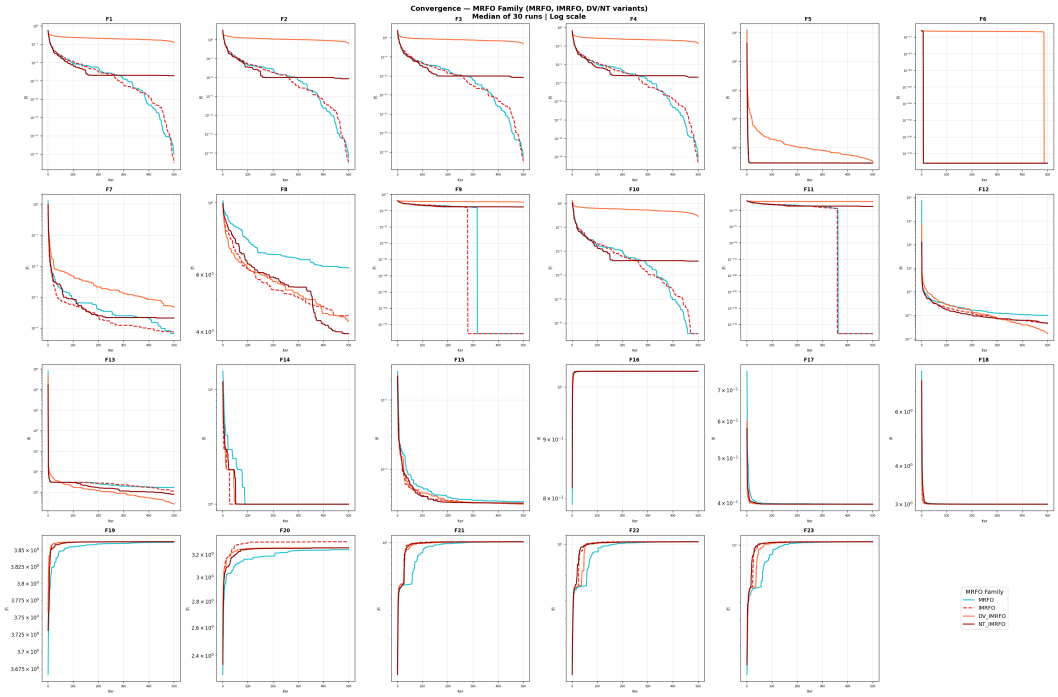

Figure 3: Convergence plots for all 23 of Yao et al.'s functions [5]. MRFO is cyan, IMRFO is dashed red, DV-IMRFO is dashed orange, and NT-IMRFO is dashed maroon.

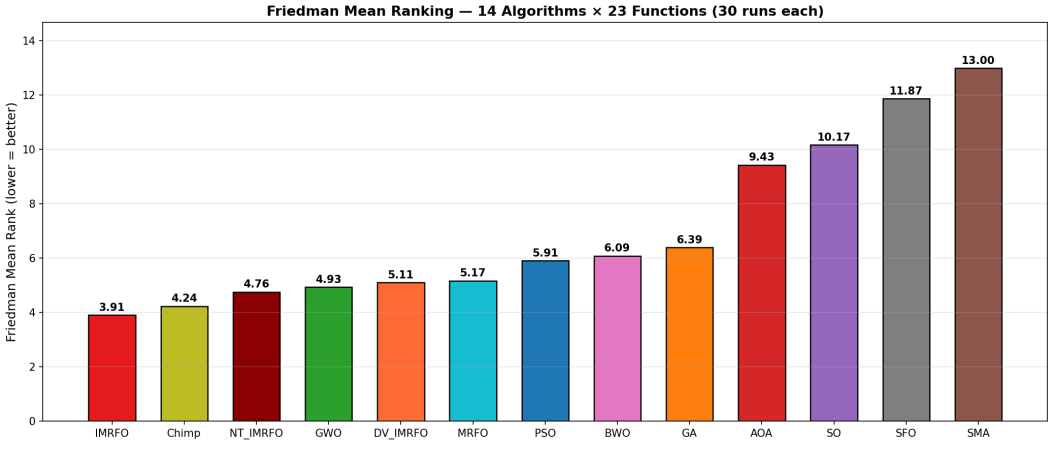

Figure 4: Friedman mean rankings of our 13 tested algorithms. Lower ranks are better.

them for eight to ten of the benchmark functions. Only one algorithm, the Grey Wolf Optimizer (GWO), is able to truly challenge IMRFO, having more wins than losses (10-7-6 wins-ties-losses for IMRFO). MRFO similarly to IMRFO, tieing on 9 benchmarks, but lost to IMRFO on the majority (14-9-0 wins-ties-losses for IMRFO).

Our convergence plots for the benchmark functions reflected these performances (see Figure 2), with most benchmarks converging at similar rates but IMRFO often converging at one of the fastest rates. Figure 3 isolates MRFO, IMRFO, and our two variant IMRFO algorithms, showing that they are all closely aligned. However, as indicated by our Wilcoxon tests (1), the two variants occasionally outperform IMRFO (DV-IMRFO wins for functions 12 and 13; NT-IMRFO wins for functions 8 and 17).

Finally, we calculated Friedman mean rankings for each of our 13 tested algorithms. Figure 4 shows these rankings; IMRFO overall has the lowest ranking, which means it has performed the best. In alignment with our other figures, the rest of the MRFO family performs quite well, though the Chimp and GWO algorithms have slightly lower ranks than DV-IMRFO and MRFO, and the Chimp algorithm outperforms NT-IMRFO.

5.2.2 Container-Packing Problem

Despite the strong results of the benchmark test suites, neither MRFO nor IMRFO appeared to surpass other metaheuristics in the container-packing problem. Neither MRFO, IMRFO, nor the two variants managed to perform the best in the small, medium, or large packing problems.

In the small quantity scenario, IMRFO manages to achieve the best utility score, but overall has a lower mean utility score than the Gray Wolf Optimizer (see Table 1). All manta ray algorithms do worse for the medium quantity scenario (Table 2), and the Genetic Algorithm actually performs the best. Finally, in the large scenario (Table 3), the Particle Swarm Optimization algorithm is the best performer, and the manta ray algorithms have mediocre utility scores.

Overall, manta ray foraging does not appear well-suited to the container-packing problem. This could be for many reasons. It may be that the problem itself is just better-suited to other metaheuristics, like PSO. It could also be that our simplifications of container packing constrained the search space in a way that was less advantageous for a complex algorithm like IMRFO. Finally, it could also be that our limited number of runs and chosen problem hyperparameters mean that this specific test leads to poor performance, but other instances of container-packing have high IMRFO utility scores.

Table 1: Small quantity three-dimensional container-packing results. Best algorithm is bolded.

Algorithm	Mean util	Std util	Best util	Boxes packed
PSO	82.60%	0.0092	84.30%	8/15
GA	82.20%	0.0228	84.30%	8/15
GWO	**83.70%**	**0.0065**	**84.30%**	**8/15**
MRFO	80.40%	0.0148	82.00%	8/15
IMRFO	83.20%	0.0063	84.30%	8/15
DV_IMRFO	81.60%	0.0161	83.20%	8/15
NT_IMRFO	82.50%	0.0057	83.30%	8/15
AOA	82.20%	0.0057	82.80%	8/15
BWO	82.60%	0.0084	83.20%	8/15
SFO	81.80%	0.0128	83.10%	8/15
SMA	76.90%	0.0367	81.60%	8/15
SO	79.60%	0.0137	81.30%	8/15

5.2.3 Reproducibility Comparison

In order to evaluate the reproducibility of both the MRFO and IMRFO algorithms, our reproduced Python-based implementation of each algorithm was run across all 23 benchmark functions using the same experimental conditions reported in the original paper:

- *Population size*: 50
- *Number of iterations*: 500
- *Number of independent runs*: 30
- *Problem dimensionality*: 30

Table 2: Medium quantity three-dimensional container-packing results. Best algorithm is bolded.

Algorithm	Mean util	Std util	Best util	Boxes packed
PSO	75.20%	0.0149	76.80%	16/25
GA	**76.10%**	**0.0126**	**78.40%**	**16/25**
GWO	75.60%	0.0097	77.30%	15/25
MRFO	73.90%	0.0267	76.40%	15/25
IMRFO	75.10%	0.0091	76.20%	15/25
DV_IMRFO	75.80%	0.0084	76.60%	16/25
NT_IMRFO	76.20%	0.0071	76.90%	16/25
AOA	74.50%	0.0147	76.20%	16/25
BWO	76.50%	0.02	77.70%	15/25
SFO	72.80%	0.0046	73.60%	15/25
SMA	62.00%	0.0279	67.60%	15/25
SO	73.10%	0.0189	76.30%	15/25

Table 3: Large quantity three-dimensional container-packing results. Best algorithm is bolded.

Algorithm	Mean util	Std util	Best util	Boxes packed
PSO	**80.80%**	**0.0249**	**84.80%**	**32/40**
GA	79.10%	0.0116	80.70%	31/40
GWO	80.10%	0.004	80.60%	27/40
MRFO	79.30%	0.0198	81.70%	30/40
IMRFO	79.70%	0.0095	81.50%	29/40
DV_IMRFO	80.70%	0.0109	82.20%	29/40
NT_IMRFO	79.80%	0.0137	81.50%	32/40
AOA	80.00%	0.007	80.70%	28/40
BWO	80.70%	0.006	81.80%	30/40
SFO	76.70%	0.0163	78.40%	28/40
SMA	67.10%	0.0316	70.20%	27/40
SO	78.70%	0.0062	79.50%	30/40

For MRFO, our implementation was led by publicly available reference code [1], while IMRFO was reproduced from the pseudocode and mathematical descriptions by Qu et al. [2]. Figure 5 reports the relevant paper's mean fitness values next to our reproduced means and the absolute error between them for each benchmark function.

One notable pattern is that our IMRFO results diverge more from the paper than our MRFO results do. We do not have a definitive explanation, but there are a few possible reasons. First, unlike MRFO, IMRFO has no publicly available reference implementation, so our version was built directly from the paper's pseudocode. Any ambiguity in how the Tent mapping, bidirectional search, or Lévy flight were described could compound into implementation differences. Second, IMRFO's additional strategies, particularly the Lévy flight random walk, likely make it more sensitive to random seeding than MRFO, meaning run-to-run variance is inherently higher. It is also worth pointing out that this kind of reproducibility gap is a known issue in the metaheuristics literature where results are often difficult to replicate exactly, even with identical parameters, which is part of why reproducibility studies like this one are valuable. This reproducibility gap was consistent across CEC2017 and CEC2022 as well, where IMRFO again showed larger divergence from the paper-reported values than MRFO, further supporting this idea that the added complexity of IMRFO makes it more difficult to reproduce faithfully from pseudocode alone. Full CEC benchmark results are reported in Tables 4 and 5, respectively.

6 Discussion

Overall, the results of our experimentation are positive, but not overwhelming. As expected, the Improved Manta Ray Foraging Optimization algorithm does perform better than its predecessor, Manta Ray Foraging Optimization. However, IMRFO does not consistently perform better than other metaheuristics in benchmark test suites, nor in our container-packing problem. These results are reaffirmations of the No Free Lunch Theorem, demonstrating that no metaheuristic can outperform in all cases.

As shown by generally small absolute errors in Figure 5, our implementation of MRFO is fairly accurate to Zhao et al.'s original 2019 paper. However, our IMRFO implementation varies more significantly from Qu et al.'s 2024 paper. This is likely a result of increased randomization in IMRFO alongside more ambiguous code in the IMRFO paper as compared to the MRFO paper

Reproducibility Study: MRFO vs IMRFO | pop=50, iters=500, runs=30, dim=30							
		MRFO			IMRFO		
Function	Group	Paper Mean	Our Mean	Abs. Error	Paper Mean	Our Mean	Abs. Error
F1 Sphere	Unimodal	0.00E+00	4.78E-08	4.78E-08	0.00E+00	4.67E+00	4.67E+00
F2 Schwefel 2.22	Unimodal	8.01E-217	3.11E-06	3.11E-06	1.80E-224	3.09E+00	3.09E+00
F3 Schwefel 1.2	Unimodal	0.00E+00	2.61E+00	2.61E+00	0.00E+00	1.60E+04	1.60E+04
F4 Schwefel 2.21	Unimodal	1.42E-211	4.47E+00	4.47E+00	2.06E-217	8.82E+00	8.82E+00
F5 Rosenbrock	Unimodal	2.16E+01	2.79E+01	6.37E+00	4.13E-01	5.19E+02	5.19E+02
F6 Step	Unimodal	1.02E-13	0.00E+00	1.02E-13	3.10E-15	9.60E+00	9.60E+00
F7 Quartic+Noise	Unimodal	1.20E-04	5.29E-03	5.17E-03	2.15E-05	1.14E-01	1.14E-01
F8 Schwefel 2.26	Multimodal	-8.49E+03	8.61E+03	1.71E+04	-1.23E+04	7.24E+03	1.95E+04
F9 Rastrigin	Multimodal	0.00E+00	1.63E-08	1.63E-08	0.00E+00	2.49E+02	2.49E+02
F10 Ackley	Multimodal	8.88E-16	8.03E-06	8.03E-06	8.88E-16	2.16E+00	2.16E+00
F11 Griewank	Multimodal	0.00E+00	5.18E-07	5.18E-07	0.00E+00	1.04E+00	1.04E+00
F12 Penalized 1	Multimodal	3.46E-03	3.96E-01	3.92E-01	8.57E-17	6.24E+00	6.24E+00
F13 Penalized 2	Multimodal	1.98E+00	2.23E+00	2.55E-01	2.38E-15	2.99E+00	2.99E+00
F14 Foxholes	Fixed-Dim	9.98E-01	1.00E+00	6.00E-03	9.98E-01	9.98E-01	3.90E-06
F15 Kowalik	Fixed-Dim	3.50E-04	8.54E-04	5.04E-04	3.07E-04	7.51E-04	4.44E-04
F16 Camel Six-Hump	Fixed-Dim	-1.03E+00	-1.03E+00	1.60E-05	-1.03E+00	-1.03E+00	2.30E-05
F17 Branin	Fixed-Dim	3.98E-01	4.01E-01	2.82E-03	3.98E-01	3.98E-01	2.80E-04
F18 Goldstein-Price	Fixed-Dim	3.00E+00	3.00E+00	2.38E-04	3.00E+00	3.00E+00	0.00E+00
F19 Hartmann 3	Fixed-Dim	-3.86E+00	-3.86E+00	7.10E-03	-3.86E+00	-3.86E+00	2.00E-05
F20 Hartmann 6	Fixed-Dim	-3.26E+00	-3.10E+00	1.62E-01	-3.32E+00	-3.25E+00	6.97E-02
F21 Shekel 5	Fixed-Dim	-9.13E+00	-4.04E+00	5.09E+00	-1.02E+01	-7.01E+00	3.14E+00
F22 Shekel 7	Fixed-Dim	-9.29E+00	-3.91E+00	5.39E+00	-1.04E+01	-8.36E+00	2.04E+00
F23 Shekel 10	Fixed-Dim	-1.00E+01	-3.95E+00	6.04E+00	-1.05E+01	-8.58E+00	1.96E+00

Abs. Error < 1 1 – 100 > 100

Figure 5: Comparison of paper-reported and reproduced mean fitness values for MRFO and IMRFO.

(pseudocode versus MATLAB). Our results underscore the importance of reproducibility studies; this paper demonstrates the results as shown in Qu et al. 2024 are potentially inaccurate or not consistent. Our work also shows why it is critical for computational research to include code when possible, to reduce the chance of ambiguities in future implementations.

7 Conclusion

Our work does support the proposal by Zhao et al. (2019) and Qu et al. (2024) that MRFO/IMRFO can perform well, on par with many high-performing metaheuristics. IMRFO's three additional strategies (Tent chaos mapping, bidirectional search, and Lévy flights) appear to address the weaknesses of MRFO substantially. While IMRFO is not always the best out of our set of tested metaheuristics, its strong performance suggests it should be considered more often alongside well-known algorithms like PSO and GA.

Having reproduced results to varying degrees of success from previous works on MRFO/IMRFO, the next step is to continue to build out MRFO tools. Our implementations of both algorithms, while usable, could benefit from continued documentation and polishing. Further refactoring of the code to make it more flexible would also benefit industry users of metaheuristics, who need easy, versatile libraries. Further work could also be done to identify key weaknesses that remain in IMRFO; though we did minor comparisons with novel improvements to IMRFO in this paper, it was not at all the main focus. The additional strategies also failed to produce significantly better results. Along the same lines, matching the strengths of IMRFO to combinatorial problems with less theoretical applications would create a stronger argument in support of its adoption.

Code

All code used in this paper, as well as our official results, is stored in a GitHub repository. The repository can be found here: `https://github.com/maxfroh/CSCI_633_MRFO`.

Table 4: CEC2017 benchmark results (dim=30, 30 runs). Mean and standard deviation are reported for MRFO and IMRFO.

ID	Description	MRFO Mean	MRFO Std	IMRFO Mean	IMRFO Std
Unimodal					
F01	Shifted & Rotated Bent Cigar	1.78e+10	2.24e+09	4.31e+10	5.48e+09
Simple Multimodal					
F03	Shifted & Rotated Rosenbrock's	1.86e+03	4.54e+02	6.44e+03	2.33e+03
F04	Shifted & Rotated Rastrigin's	1.92e+04	3.47e+03	4.24e+04	5.73e+03
F05	Shifted & Rotated Schaffer's F7	5.00e+02	5.38e-03	5.00e+02	1.17e-02
F06	Shifted & Rotated Lunacek Bi-Rastrigin	7.71e+05	1.27e+05	1.40e+06	1.62e+05
F07	Shifted & Rotated Non-Cont. Rastrigin	1.25e+03	5.81e+01	1.44e+03	5.53e+01
F08	Shifted & Rotated Levy	8.14e+02	3.06e+00	8.42e+02	7.37e+00
F09	Shifted & Rotated Schwefel's	3.76e+03	9.90e+02	3.58e+03	7.95e+02
Hybrid					
F10	Hybrid Function 1	1.44e+06	7.61e+05	2.91e+06	1.40e+06
F11	Hybrid Function 2	2.55e+09	6.80e+08	5.78e+09	9.58e+08
F12	Hybrid Function 3	1.40e+09	6.25e+08	3.88e+09	1.57e+09
F13	Hybrid Function 4	1.45e+06	1.05e+06	3.65e+06	2.21e+06
F14	Hybrid Function 5	4.03e+08	1.96e+08	1.32e+09	4.10e+08
F15	Hybrid Function 6	2.48e+07	1.20e+07	2.14e+08	1.66e+08
F16	Hybrid Function 7	8.30e+08	1.92e+09	8.48e+09	1.35e+10
F17	Hybrid Function 8	4.17e+05	3.64e+05	7.45e+05	7.64e+05
F18	Hybrid Function 9	5.08e+10	3.05e+10	9.73e+11	9.59e+11
F19	Hybrid Function 10	4.80e+03	8.21e+02	9.56e+03	1.62e+03
Composition					
F20	Composition Function 1	1.23e+04	4.64e+03	2.73e+04	1.53e+04
F21	Composition Function 2	2.50e+03	7.36e+01	2.70e+03	3.44e+02
F22	Composition Function 3	2.21e+04	3.99e+03	2.88e+04	5.92e+03
F23	Composition Function 4	1.55e+04	2.66e+03	1.79e+04	4.36e+03
F24	Composition Function 5	3.48e+03	1.30e+02	5.30e+03	6.50e+02
F25	Composition Function 6	3.67e+03	1.05e+02	3.51e+03	3.95e+01
F26	Composition Function 7	3.33e+03	6.69e+01	3.29e+03	2.94e+01
F27	Composition Function 8	3.64e+03	1.50e+02	4.05e+03	1.25e+02
F28	Composition Function 9	2.78e+09	3.93e+09	1.86e+10	2.55e+10
F29	Composition Function 10	4.29e+09	2.54e+09	2.22e+10	1.38e+10

Table 5: CEC2022 benchmark results (dim=20, 30 runs). Mean and standard deviation are reported for MRFO and IMRFO.

ID	Description	MRFO Mean	MRFO Std	IMRFO Mean	IMRFO Std
Simple Multimodal					
F01	Shifted & Rotated Zakharov	8.05e+03	1.61e+03	2.31e+04	5.24e+03
F02	Shifted & Rotated Rosenbrock's	7.05e+02	6.33e+01	1.12e+03	1.78e+02
F03	Shifted & Rotated Expanded Schaffer's	6.00e+02	5.28e-02	6.01e+02	9.03e-02
F04	Shifted & Rotated Non-Cont. Rastrigin	1.05e+03	3.56e+01	1.14e+03	2.99e+01
F05	Shifted & Rotated Levy	9.04e+02	9.60e-01	9.12e+02	3.19e+00
Hybrid					
F06	Hybrid Function 1	1.44e+08	7.79e+07	4.82e+08	2.85e+08
F07	Hybrid Function 2	2.66e+03	2.30e+02	3.65e+03	5.85e+02
F08	Hybrid Function 3	9.26e+05	3.72e+06	6.25e+07	1.19e+08
Composition					
F09	Composition Function 1	2.85e+03	8.25e+01	2.86e+03	7.16e+01
F10	Composition Function 2	3.14e+03	6.55e+02	3.10e+03	5.98e+02
F11	Composition Function 3	2.89e+03	5.18e+02	3.46e+03	7.03e+02
F12	Composition Function 4	3.00e+03	2.07e+01	3.00e+03	2.17e+01

References

[1] Weiguo Zhao, Zhenxing Zhang, and Liying Wang. Manta ray foraging optimization: An effective bio-inspired optimizer for engineering applications. *Expert Systems with Applications*, 135: 195–217, 2019.

[2] Peng Qu, Qing Yuan, Feng Du, and Qiang Gao. An improved manta ray foraging optimization algorithm and its engineering applications. *Scientific Reports*, 14(59960), 2024.

[3] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[4] Seyedali Mirjalili, Seyed Mohammad Mirjalili, and Andrew Lewis. Grey wolf optimizer. *Advances in Engineering Software*, 69:46–61, 2014.

[5] Xin Yao, Yong Liu, and Guangming Lin. Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation*, 3(2):82–102, 1999.

[6] Noor H. Awad, Mostafa Z. Ali, Jing J. Liang, Bo Yang Qu, and Ponnuthurai N. Suganthan. Problem definitions and evaluation criteria for the CEC 2017 special session on single objective bound constrained real-parameter numerical optimization. Technical report, IEEE Congress on Evolutionary Computation, 2017.

[7] Ali Wagdy Mohamed, Ali Ahrari Hadi, and Kamal Mohammed Jambi. CEC 2022 single objective bound constrained optimization benchmark functions. Technical report, IEEE Congress on Evolutionary Computation, 2022.