RESEARCH PAPER DOI: 10.22080/cste.2026.32861.1160

Contributions of Science and Technology for Engineering

Journal Home page: https://cste.journals.umz.ac.ir/

ECPO: Promoting the Rollback Buffer to an Explicit Memory in the Crested Porcupine Optimizer

Zahrasadat Askari [1, 2], Hanif Aghagoli Kasgari [1, 2], Seyyed Ali Zendehbad [1, 2], Jamal Ghasemi [1, 2*]

[1] *Faculty of Engineering and Technology, University of Mazandaran, Babolsar, Iran.*

[2] *Artificial Intelligence Research Laboratory (AIRL), Faculty of Engineering & Technology, University of Mazandaran, Babolsar, Iran.*

ORCID: Zahrasadat Askari: 0009-0007-6907-9780; Hanif Aghagoli Kasgari:0009-0005-2395-4303; Seyyed Ali Zendehbad:0000-0002-5988-7043; Jamal Ghasemi: 0000-0002-9573-0107.

**Corresponding Author: j.ghasemi@umz.ac.ir*

Article Info

Received 20 October 2026

Accepted 21 November 2026

Available online 28 November 2026

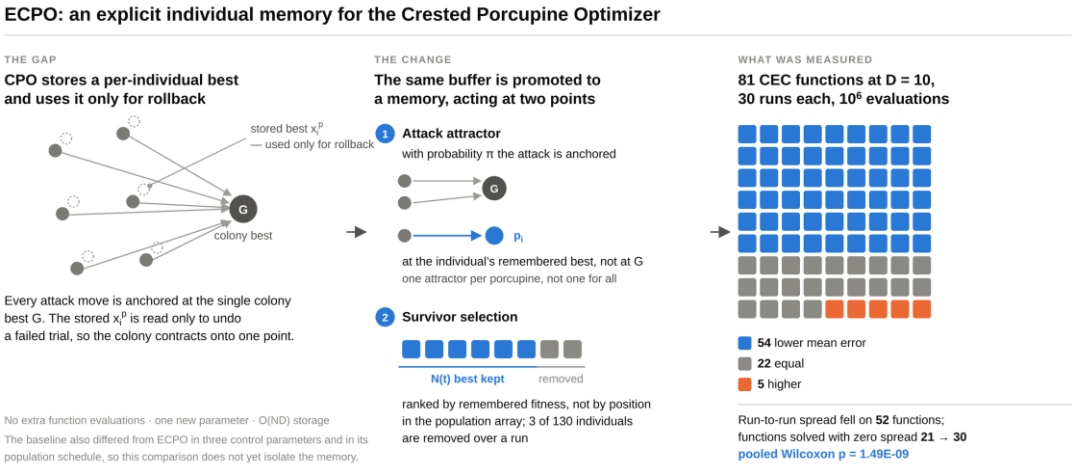
Keywords:

Crested Porcupine Optimizer;
Individual Memory;
Attractor Diversity;
Survivor Selection;
CEC Benchmark Suites;
Wilcoxon Signed-Rank Test.

Abstract:

The Crested Porcupine Optimizer (CPO) stores each individual's best accepted position for rollback. The proposed ECPO also uses this position to guide some of the physical attacks and ranks individuals by their best recorded fitness when reducing the population. These are the only two changes, adding one parameter and requiring no additional objective evaluations. We ran ECPO against an in-house CPO implementation on the 81 functions from CEC2014, CEC2017, CEC2020 and CEC2022 with 30 independent runs per function at D=10 and a nominal budget of 10^6 evaluations per run. ECPO achieved a lower mean objective value on 54 functions, was tied on 22 and had a higher mean on 5. 33 of the improvements were significant at the 5% level and 22 remained significant after Holm correction. No degradation was significant. The pooled Wilcoxon signed rank test was in favour of ECPO ($p = 1.49 \times 10^{-9}$) and the standard deviation across runs was reduced on 52 functions. The comparison with published CPO results on the 28 CEC2017 functions revealed no significant difference. The in-house baseline used different control parameters and a different population reduction schedule and both implementations used fitness to select survivors. A matched baseline and an ablation study would be required to quantify the contribution of individual memory.

Graphical Abstract



© 2026 University of Mazandaran

Copyright © 2026 by the authors. Licensee CSTE, Babolsar, Mazandaran. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (https://creativecommons.org/licenses/by/4.0/deed.en)

ISSN 3060-6578

Please cite this paper as: Z. Askari, H. Aghagoli-Kasgari, S. A. Zendehbad, and J. Ghasemi, "ECPO: Promoting the Rollback Buffer to an Explicit Memory in the Crested Porcupine Optimizer,", Contrib. Sci. & Tech Eng, vol. 3, no. 3, pp. 75–98, 2026, doi:10.22080/cste.2026.32861.1160.

1. Introduction

With the objective being expensive, non-differentiable, multimodal, or black-box, population-based metaheuristics are now the go-to approach for continuous optimization problems [1, 2]. Their attractiveness is apparent. They do not require a gradient, are linear in the number of evaluations, and can be attached to a simulator or to a legacy solver with only minor modifications. No single method dominates: none outperforms all others across every problem class. The no-free-lunch result of Wolpert and Macready [3] makes this precise, and it carries a practical implication: progress in the field should come not from devising new metaphors, but from identifying the specific step at which an already successful algorithm discards information. The Crested Porcupine Optimizer, proposed by Abdel-Basset et al. [4], is an appropriate topic for such a study. It mimics the graduated defense strategy of the crested porcupine: initially displaying its spines, then vibrating them to produce a rattling sound, then releasing an odor, and finally charging at its predators as a last resort. These four behaviors are mapped onto four position-update operators of increasing exploitative character, and, stochastically, one of these is chosen. The number of active individuals decreases in cycles and then increases several times during a run. The reported results demonstrate that it beats a wide range of contemporary swarm algorithms on the CEC suites and on engineering design problems. The architecture is simple enough to allow some degree of reasoning about its behavior beyond mere performance measurement. That reasoning leads to two instances in which CPO is, in a very literal sense, forgetful. The first concerns its core exploitation mechanism. The physical attack operator pulls an individual towards the best position G in the colony, and the amount of perturbation that offsets this pull diminishes as the run progresses. If this is done repeatedly, then an update of this type is a contraction towards a single point, and as the population shrinks, the set of values of the exploratory operators also shrinks. That is to say, the loss of diversity is in itself a force; the more the colony collapses onto G, the fewer steps there are to get away from it. The second concerns the population reduction. CPO cycles the number of active individuals, but it doesn't choose which individuals to deactivate. The main loop simply iterates over the first few items in the population array, and an individual survives a reduction only if it is at a low index. The index position contains no information about the solution's quality. It is fixed at the time of initialization, so a porcupine that has just stumbled upon a good basin will be deactivated or retained purely based on its arbitrary position in the population array. The second observation is worth pursuing, as CPO already holds the information required to perform better. For each CPO, it maintains a vector x_i^p and will restore the individual to that vector whenever the trial move fails. The stored fitness is monotonically non-increasing, as a trial is only accepted if it is no worse than the incumbent and x_i^p is therefore the best position the individual has ever reached. CPO thus maintains a per-individual best, but it is not used in any capacity other than during rollback. It is not used to select a move direction or survivors. The information exists, it is free to obtain, and it is thrown away. That is precisely the space this paper fills. We propose an extension of CPO called ECPO (explicit individual memory) in which the rollback buffer is moved into the individual's own memory and used at two decision points where it is not currently used. With a small fixed probability, the physical attack operator is anchored to the individual's own remembered best rather than to the colony best. The fixed points of the most contractive operator of the algorithm will now be different for each individual rather than shared by the entire colony, whilst the exploitative component is unchanged: a remembered personal best is, by construction, a good point. The individuals retained at each reduction step will be those with the best-remembered fitness, which turns an arbitrary deactivation into a truncation selection operator. The two changes together require no additional function evaluations and introduce just one new parameter. The contribution is deliberately minimal. It is very difficult to determine which of the various changes to an algorithm made when it is modified in 5 places has actually made the difference: it is very hard to attribute performance gains in complex extensions of simple metaheuristics. Here the intervention is confined to two specific decisions, is small in scope, and is precisely specified so that it can be reproduced. If the reduction step is, in fact, a source of unnecessary randomness, then correcting it should result in a tighter distribution of results across independent runs (not just a tighter average), and the effect should be most pronounced for functions on which CPO fails occasionally rather than systematically. Section 4 reports results which follow this pattern. Section 3.3.2 explains why they cannot be taken as a test of it: the CPO baseline we ran was different from ECPO in more than just the two modifications. Section 5 discusses this and the one function on which ECPO is clearly worse. A second contribution is methodological and stems from a question we were unable to resolve from the literature. When evaluating a modified algorithm on the CEC suites, an improvement can be expressed either in terms of the globally known optimal solution for the benchmark problem or in terms of the numbers that have been published for the benchmark algorithm, and the two are often conflated. We show that the question can be separated: the reference is no longer a statistical question, as it algebraically cancels out in any paired significance test. The baseline does not cancel and must be a matched re-execution. With the same ECPO results, the impact on our own data is shown in Section 4.9: p = 9.82E-05 for the re-executed CPO and p = 0.865 for the published table. As Section 3.3.2 reports our own re-execution was not in fact a matched re-execution in terms of its control parameters and Section 4.9 shows how important that is. The paper is structured as follows: Section 2 reviews how memory, diversity control, and survivor selection have been addressed in comparable algorithms and what remains to be done. Section 3 restates CPO in the notation used throughout, develops ECPO and describes the experimental protocol and statistical framework. The results from the four suites are reported in Section 4. These are then interpreted and analyzed for failure, with limitations detailed in Section 5, followed by a conclusion.

2. Related Work

The literature relevant to this paper is not that of nature-inspired optimizers in general, which has been reviewed adequately elsewhere [1]. It is the much narrower literature on three design decisions: what an individual is allowed to remember, how many attractors a search maintains, and how a population is reduced. Each is examined critically rather than chronologically.

2.1. Memory in Population-Based Search

Memory entered metaheuristics with tabu search [5] (in which a list of recently visited configurations is used to prevent re-visits) and ant colony optimization [6] (in which a distributed list of the components recently found in good solutions is recorded in a pheromone matrix). Both are recollections of the search generally. A complementary concept appeared in particle swarm optimization [7]: the individual's own memory. Each particle moves towards its personal best position and its neighborhood best, and the balance between the two is the algorithm's principal control parameter. Three decades of PSO variants show that the personal-best term is not decorative. Removing it collapses the swarm onto the global best; this term is what keeps a swarm a swarm and resists premature convergence, as social-only variants demonstrate. Differential evolution [8] itself does not keep such a record, but its success-history variants do. SHADE [9] and its linear-population-reduction descendant L-SHADE [10] keep historical records of the values of the control parameters that have produced successful offspring in the recent past and use these records to sample new ones. The lesson generalizes beyond DE: an algorithm that records which of its options have succeeded in the past can adapt without user tuning. This is formalized as a credit-assignment problem in the context of adaptive operator selection and is treated on a principled basis by bandit schemes [11]. Such schemes are, however, easier to describe than to tune: credit assignment introduces its own learning rate, window length, and exploration constant, so an algorithm that was difficult to tune becomes difficult to tune in a new way. Most recent swarm optimizers have absorbed neither lesson. The Grey Wolf Optimizer [12], the Whale Optimization Algorithm [13] and the Salp Swarm Algorithm [14] guides individuals toward one or a few globally best individuals. In contrast, the Equilibrium Optimizer [15], the Gradient-based Optimizer [16], the Slime Mould Algorithm [17], and the African Vultures Optimization Algorithm [18] maintain absolutely no record of the individual that has influenced their movement. These are clearly attractive algorithms, as they converge very rapidly on easy landscapes and are easy to implement. The same characteristic that makes them so attractive in this respect is also what makes them so fragile in the case of composite functions, where the leader may well end up in the wrong basin and the whole population then follows it there. CPO is in this category and has the additional property mentioned in Section 1: it computes a per-individual best incidentally and uses it only for rollback.

Memory-based metaheuristics, such as tabu search [5] and the Swarm Algorithm [14], guide individuals toward one or a few globally best individuals. In contrast, the Equilibrium Optimizer [15], the Gradient-based Optimizer [16], the Slime Mould Algorithm [17], and the African Vultures Optimization Algorithm [18] maintain absolutely no record of the individual that has influenced their movement. These are clearly attractive algorithms, as they converge very rapidly on easy landscapes and are easy to implement. The same characteristic that makes them so attractive in this respect is also what makes them so fragile in the case of composite functions, where the leader may well end up in the wrong basin and the whole population then follows it there. CPO is in this category and has the additional property mentioned in Section 1: it computes a per-individual best incidentally and uses it only for rollback.

2.2. Attractor Multiplicity and Diversity Maintenance

The exploration/exploitation trade-off is the standard way of putting this issue [2]. Still, it's rather crude in that it treats diversity as a scalar to be scheduled rather than as a structural property of the update rule. A better question is: how many attractors does the update rule have? Any rule of the form $x \leftarrow x+c(A-x)$, with a single common A, is a contraction mapping towards A; i.e., repeated application will cause the diameter of the population to shrink geometrically regardless of the schedule on c. If, instead, each individual is assigned its own attractor and is only attracted to that attractor, then the diameter of the population is bounded below by the diameter of the set of attractors. This is why the usual fixes for single attractor swarms are so ineffective. Random restarts, mutation operators, and diversity-driven perturbations all work by injecting some amount of spread, which then has to be tuned against the contraction: the rule is constantly contracting the population whilst the perturbation is constantly trying to push it out. A multi-attractor design is fundamentally different in that it requires no injection. PSO obtains this for free under its personal-best term, and CMA-ES [19] obtains something similar by updating a full covariance matrix rather than a point estimate. To our knowledge, the minimal change (replacing the shared attractor with the individual attractor on a small number of trials) has not been studied in an algorithm that retains its single-leader design. This is the gap that the first modification proposed in this paper addresses.

2.3. Population-size Control and Survivor Selection

Varying the population during a run is an established idea, and the form it takes in L-SHADE [10] has become a de facto standard: the population shrinks as the budget is consumed, and the individuals removed at each step are the worst currently held. The two halves of that design are adopted together, and the second is what makes the first defensible. A schedule decides how many individuals remain active; a selection rule decides which ones. Taken together, reduction becomes a selection operator, and shrinking the population tightens selection pressure in a controlled way. CPO separates the two halves. Its schedule is cyclic rather than monotonic: the number of active individuals falls within a cycle. It is restored at the start of the next cycle, with the stated intention of alternating between exploratory and exploitative regimes without restarting the search. The schedule is explicit and deliberate. The accompanying selection rule, however, is absent. Reducing the active count only shortens the main loop, so the individuals that remain active are those occupying the first $N(t)$ positions of the population array (an order fixed at initialization and uncorrelated with solution quality). An individual which has just discovered a good basin therefore survives or is deactivated depending on where it happens to be in the population array. We are not aware of a study that isolates this particular omission. Work on population sizing focuses on the schedule — how large the population should be and how quickly it should shrink — and treats quality-based survivor selection as given, as is common in the differential evolution and evolution strategy literature. In a single-leader swarm optimizer that reduces its population without ranking it, that choice is not given, and the quantity required to make it is the same per-individual record discussed in Section 2.1. This is the gap that the second modification proposed in this paper addresses.

2.4. Positioning

The three threads all come to the same conclusion. CPO is a competitive single-leader swarm optimizer that already has exactly the diversity the PSO literature has identified as the cure for single-leader collapse, and exactly the amount that an L-SHADE-style reduction scheme needs to select survivors sensibly. ECPO is the minimum change that closes both gaps when used. It is not a hybrid at all. It makes no use of PSO or DE operators. It retains all four defensive mechanisms, the acceptance rule, and the reduction schedule of Equation 2 as CPO computes them.

3. Method and Materials

3.1. The Crested Porcupine Optimizer

This section simply re-states CPO [4] in the notation that we'll be using for the rest of the paper and concentrates on the information required to describe the proposed modification. Throughout the following, we will assume that D is the problem dimension, L and U are the lower- and upper-bound vectors, respectively, f is the objective function to be minimized, and t is the number of objective function evaluations already performed (the budget is T_max). The colony at time t is a set of $N(t)$ position vectors x_i in R^D, r, r_1, r_2, ... are independent random variables drawn from the uniform distribution $U(0,1)$, u_1 and u_2 are binary masks in $0,1^D$, redrawn at each trial (Section 3.2.3 details how the reference code generates u_2). The symbol $\odot$ denotes the elementwise product. G^t does the colony find the best position up to time t. When r_1, r_2 or r_3 appear as subscripts of x; they denote the index of a colony member selected uniformly at random; otherwise, they are one of the uniform variables described above.

index of a colony member selected uniformly at random; otherwise they are one of the uniform variables described above.

3.1.1. Initialization and Cyclic Population Control

The colony is initialized uniformly in the box, with N' individuals:

$$x_i = L + r \odot (U - L), i = 1, \ldots, N' \tag{1}$$

CPO then varies the number of active individuals cyclically between an initial value N' and a floor N_{min}, completing T cycles over the run:

$$N(t) = \lfloor N_{\min} + (N' - N_{\min})\big(1 - \operatorname{rem}(t, T_{\max}/T)/(T_{\max} \cdot T)\big) \rfloor \tag{2}$$

where rem(a,b) is the remainder of a divided by b. Within a cycle $N(t)$ will decrease as evaluations accumulate and will be reset when the remainder resets. The idea is to alternate between a wide exploratory regime and a narrow exploitative regime without restarting the search. Equation 2 is as implemented in the reference implementation of CPO [4] (it computes the remainder as rem (remainder, $T_{\max} \cdot T$) rather than rem (remainder, $T_{\max}/T$) which is what the division is meant to represent). The ratio therefore never exceeds $1/T^2$ and with the settings in Table 1 (T=2) the count never drops below 127 even though N_min=120. Written with the division in parentheses, the ratio reaches 1 at the end of each cycle, and the count drops to N_min. Both forms are used in the two programs compared in this paper, and they produce very different schedules; Section 3.3.2 indicates which program computed which, and Table 1 provides the settings for each. Equation 2 governs the number of active individuals. What happens to the rest depends on the implementation: the reference code simply shortens the loop so that individuals beyond the bound are idle when the bound is low and are reused when the remainder resets, which is what gives the schedule its cyclic nature. Section 3.2.4 returns to this point, as the proposed modification changes it.

Table 1. Control parameters of ECPO. The top block is the CPO parameters [4]; the bottom block is the one introduced in this paper. The final column gives the settings actually used for the CPO baseline of Section 3.3.2. They differ from the ECPO settings in three parameters and Section 3.3.2 discusses the implications of that

Parameter	Symbol	ECPO	CPO baseline	Role
Initial population	N'	130	130	Initial colony size; upper limit of Equation 2
Population floor	N_{min}	120	80	Parameter of Equation 2; realized floor 127 for ECPO and 80 for the baseline (Section 3.3.2)
Reduction cycles	T	2	2	Number of cycles of Equation 2 per run
Evaluation budget	T_{max}	1,000,000	1,000,000	Termination criterion
Convergence rate	α	0.2	0.1	Lower bound of the attack coefficient, Equations 7, 13
Defense threshold	T_f	0.8	0.2	Probability of odor rather than physical attack
Independent runs	R	30	30	Repetitions per function
Memory attractor rate	π	0.2	–	Probability of using p_i instead of G, Equation 12

3.1.2. Exploratory Defenses: Sight and Sound

With probability one half an individual will apply one of the two exploration mechanisms. The first models the visual display, in which the porcupine raises its quills and the predator retreats without contact; algorithmically, the individual takes a normally scaled step away from the midpoint between itself and a random companion, where τ is a single normal variate applied to every coordinate, as the reference code draws it:

$$y_i^t = \left(x_i^t + x_{r_1}^t\right)/2, x_i^{t+1} = x_i^t + \tau \odot |2r_2 G^t - y_i^t|, \tau \sim N(0,1) \tag{3}$$

The second model is the acoustic warning, in which the animal rattles its quills. The update recombines the individual with the same midpoint perturbed by a scaled difference of two further companions, under a binary mask:

$$x_i^{t+1} = u_1 \odot x_i^t + (1 - u_1) \odot \left(y_i^t + r_3\left(x_{r_2}^t - x_{r_3}^t\right)\right) \tag{4}$$

Both updates take their step length from differences between members of the current colony. This is the property, referred to repeatedly below, that ties the exploratory power of CPO to the spread of its population: as the colony contracts, so do the steps these operators can produce.

3.1.3. Exploitative Defenses: Odor and Physical Attack

The remaining half of the trials use one of the two exploitative mechanisms, chosen by a threshold T_f. Both are modulated by a convergence factor that decays over the run and by a defense intensity that depends on the individual's fitness relative to the colony:

$$Y^t = 2r_4(1 - t/T_{\max})^{t/T_{\max}}, \delta_i^t = \exp\left(f(x_i^t)/\left(\sum_{k=1}^{N(t)} f(x_k^t) + \varepsilon\right)\right), S_i^t = r_5 Y^t \delta_i^t u_2 \tag{5}$$

where ε is a small constant that prevents division by zero, the three quantities of Equation 5 play distinct roles: Y^t decays the perturbation as the budget is consumed, δ_i^t scales it up for individuals whose fitness is poor relative to the colony, and the binary vector u_2 confines it to a random subset of coordinates. The third mechanism models odor emission, which repels the predator at a distance; it is applied with probability T_f and updates a masked subset of coordinates from a random companion and a scaled difference:

$$x_i^{t+1} = (1 - u_1) \odot x_i^t + u_1 \odot \left(x_{r_1}^t + \delta_i^t\left(x_{r_2}^t - x_{r_3}^t\right) - S_i^t\right) \tag{6}$$

The fourth mechanism models the physical charge, the last resort of the animal and the most exploitative operator of the algorithm. It is applied with probability $1 - T_f$ and anchors the move at the colony best:

$$\begin{aligned} &F_i^t = r_6 \odot \left(\delta_i^t\left(x_{r_1}^t - x_i^t\right)\right), \\ &\tilde{S}_i^t = r_5 Y^t (u_2 \odot F_i^t), x_i^{t+1} = G^t + (\alpha(1 - r_7) + r_7)(u_2 \odot G^t - x_i^t) - \tilde{S}_i^t \end{aligned} \tag{7}$$

The coefficient $c = \alpha(1 - r_7) + r_7$ is in [α,1] and is non-zero. In the coordinates chosen by u_2, Equation 7 maps the distance to G^t onto c times itself with a flip of sign; on the other coordinates it maps $x_{i,j}^t$ to $G_j^t - cx_{i,j}^t$, whose fixed point is $G_j^t/(1 + c)$ (Section 3.2.3 treats both cases). In either case the operator is pinned at G^t, which is the point that the whole colony has in common. This is the contraction referred to in Section 2.2 and it is the one point at which the whole colony is pulled towards a single location.

3.1.4. Boundary Handling and Acceptance

Coordinates that leave the box are resampled uniformly within it, which preserves feasibility without clustering individuals on the boundary:

$$x_{ij}^{t+1} = L_j + r\left(U_j - L_j\right) \text{ if } x_{ij}^{t+1} \notin \left[L_j, U_j\right] \tag{8}$$

The trial is then evaluated and accepted or rejected greedily. Acceptance updates the stored position x_i^p; rejection restores the individual to it:

$$\text{if } f(x_i^{t+1}) \le f_i^t : x_i^p \leftarrow x_i^{t+1}, f_i^{t+1} \leftarrow f(x_i^{t+1}) \text{ otherwise: } x_i^{t+1} \leftarrow x_i^p, f_i^{t+1} \leftarrow f_i^t \tag{9}$$

Two consequences of Equation 9 matter for what follows. First, f_i is monotonically non-increasing by construction, so x_i^p is the best position individual i has ever occupied: CPO computes an individual best whether or not it calls it one. Second, x_i^p appears nowhere else in the algorithm. It is written by Equation 9 and read by Equation 9, and no update rule and no selection step consults it.

3.2. Proposed Method: ECPO

3.2.1. Design Principle

ECPO is built on a single principle: the quantity CPO already maintains should be allowed to influence the two decisions that most obviously depend on individual history, and nothing else about the algorithm should change. The two decisions are the choice of attractor in the physical-attack operator (Equation 7) and the choice of survivors during a population reduction event (Equation 2). We take them in turn after making the memory explicit.

3.2.2. Individual Memory

Each porcupine is equipped with a memory pair (p_i, φ_i) that stores the best position it has visited and its corresponding objective value. The memory is initialized from the individual's starting position and updated after every evaluation:

$$p_i^0 = x_i^0, \varphi_i^0 = f(x_i^0); (p_i^{t+1}, \varphi_i^{t+1}) = \left(x_i^{t+1}, f(x_i^{t+1})\right) \text{if } f(x_i^{t+1}) < \varphi_i^t, (p_i^t, \varphi_i^t) \text{ otherwise} \tag{10}$$

The update is strict, whereas the acceptance rule of Equation 9 is not. The distinction is not cosmetic. On a plateau, where the trial and incumbent share a fitness value, Equation 9 accepts the move and lets x_i^p drift across the plateau, while Equation 10 pins p_i to the first position that attained the value. The memory is therefore well defined on flat regions, where the rollback

buffer is not, and it is exactly on such regions (the flat outer basins of composition functions for instance) an attractor which wanders is of minimal use. From the memory, we form a merit score that will be used for survivor selection:

$$\psi_i^t = \min(f_i^t, \varphi_i^t) \tag{11}$$

Equation 11 is written as a minimum rather than as f_i alone so that the criterion remains correct if the acceptance rule of Equation 9 is ever relaxed, for example, to a non-elitist or annealed variant, under which f_i would no longer be monotone. Under Equation 9 as published, the two quantities coincide and $\psi_i = f_i$.

Four position vectors per individual are used here, and they should be kept distinct. The current position $x_i^t$is the point from which Equations 3 to 7 and 13 generate the next trial. The rollback position $x_i^p$is given by Equation 9 and is used to restore $x_i$following a rejected trial, so that $x_i^t = x_i^p$at the beginning of each trial. The memory p_i is given by Equation 10; it is only different from x_i^p following an equal fitness move on a plateau (the move which Equation 9 accepts and Equation 10 does not). The attractor a_i^t of Equation 13 is not a separate state: it is either p_i^t or G^t as per Equation 12. Since $\varphi_i = f_i$under Equation 9, the merit of Equation 11 is equal to the current fitness in every experiment described here. Section 3.2.4 returns to the implications of this.

3.2.3. Individual Memory as an Alternative Attractor

The physical-attack operator of Equation 7 is where CPO concentrates its exploitative pressure and, for the reason given in Section 2.2, where its diversity is spent. ECPO leaves the operator's form untouched and changes only what it points at. With a fixed probability π the attractor becomes the individual's own memory:

$$a_i^t = p_i^t \text{ if } r_8 < \pi; a_i^t = G^t \text{ otherwise} \tag{12}$$

$$x_i^{t+1} = a_i^t + (\alpha(1 - r_7) + r_7)(u_2 \odot a_i^t - x_i^t) - \tilde{S}_i^t \tag{13}$$

where a_i^t is the attractor chosen by Equation 12, and all other symbols have the same meaning as in Equation 7. The modification is a strict generalization, and the meaning of the parameter becomes clear when $\pi = 0$, in which case it reduces to CPO. The expectation is structural rather than statistical, and it all depends on where the operator is anchored. Write $c = \alpha(1 - r_7) + r_7$ and set the perturbation aside for the moment. On a coordinate with $u_{2,j} = 1$, Equation 13 gives $x_{i,j}^{t+1} - a_{i,j}^t = -c(x_{i,j}^t - a_{i,j}^t)$, a contraction with factor c which may overshoot the anchor but has a $a_{i,j}^t$ as its fixed point. On a coordinate with $u_{2,j} = 0$, it gives $x_{i,j}^{t+1} = a_{i,j}^t - cx_{i,j}^t$, a contraction with the same factor whose fixed point is $a_{i,j}^t/(1 + c)$. The memory is therefore the exact fixed point only on the coordinates that u_2 selects. The perturbation $\tilde{S}_i^t$ acts on those same coordinates, with a scale proportional to Y^t which goes to zero as the budget is consumed. The two fixed points thus describe where the deterministic part settles in the latter part of a run. Equation 7 has the same two cases, with G^t in place of a_i^t.

The argument in Section 2.2 doesn't require a_i^t to be the fixed point (it only requires the fixed points to be individual). In either case, the fixed point of a trial of Equation 13 which is anchored at p_i is determined by the individual's own memory whereas every fixed point of Equation 7 is determined by G^t which is common to the whole colony. Under Equation 7 all the attack trials are driven towards a single common location. Under Equation 13, with probability π, a trial is drawn towards a location that differs from one porcupine to the next. The operator therefore has one anchor per individual rather than one per colony and the trials no longer drive the colony diameter down to zero. The spread of the memories bounds the colony diameter from below, and it shrinks only when the individuals actually converge on the same region.

Two points regarding the implementation reinforce this interpretation. Firstly, as Equation 9 restores x_i to x_i^p following a rejected trial and overwrites x_i^p following an accepted trial, $x_i^t = x_i^p$ whenever Equation 13 is applied, and therefore $x_i^t = p_i^t$ save in the case of an equal fitness move on a plateau (Section 3.2.2). The memory branch of Equation 13 therefore produces its trial from the individual's own best point rather than from a point being pulled towards it. Secondly, in the reference implementation of CPO [4] which both of the algorithms here use, the mask is generated by the statement U2=rand (1, dim)<0.5*2-1, with dim = D. MATLAB evaluates the arithmetic before the comparison so the threshold is 0 and u_2 is the zero vector in every trial. In all of the experiments reported here for both CPO and ECPO the alike, S_i^t and $\tilde{S}_i^t$ therefore vanish and every coordinate falls into the second case: the attack operators that are actually executed are $x_i^{t+1} = G^t - cx_i^t$ and $x_i^{t+1} = a_i^t - cx_i^t$, with fixed points $G^t/(1 + c)$ and $a_i^t/(1 + c)$. We retain the masked forms of Equations 5 to 7 and 13, as these are the forms in which CPO is defined. The comparison in Section 4 is unaltered, as both algorithms use the same operator and differ only in the anchor point; whether the conclusions would still hold in the case of an implementation where u_2 is a genuine random mask is one of the limitations listed in Section 5.3.

The pressure to exploit is absolutely not reduced by this change. p_i by construction is the best point that individual i has found so a trial anchored there is anchored at a locally good point. What changes is that the pressure to exploit is distributed across many points rather than focused on one. There is also an effect on exploration. The steps taken by Equations 4 and 6 are derived from the differences between colony members, so their size increases with the colony diameter. If Equation 13 helps maintain the diameter, then it also helps maintain the range of the exploratory operators as a by-product, again without any explicit diversity mechanism. The amount of the budget that is redirected is controlled by π and is deliberately very small. An attack trial occurs on a fraction $(1 - T_f)/2$ of all trials so with $T_f = 0.8$ and π=0.2 the memory attractor is being used on approximately $\pi(1 - T_f)/2 = 2\%$ of all trials. The intervention is therefore a very small perturbation to the search behavior of CPO, rather than a complete redesign. Any improvement it might produce has to be viewed as the result of a very small structural bias accumulating over the long run rather than as a change of regime.

3.2.4. Memory-Informed Survivor Selection

The second modification concerns the consequence of reducing the active population in Equation 2. In CPO, only the bound of the loop is reduced, and thus the individuals that remain active are the first N(t) entries of the population array, where the order is pre-defined. ECPO keeps the N(t) best individuals:

$$\text{let } \omega_\psi \text{ be the permutation with } \psi_{\omega_\psi(1)} \le \cdots \le \psi_{\omega_\psi(N(t-1))};\ \text{if } N(t) < N(t-1):\ \text{ retain } \left(x_j, x_j^p, f_j, p_j, \varphi_j\right): j = \omega_\psi(1), \ldots, \omega_\psi\left(N(t)\right) \tag{14}$$

Equation 14 makes the reduction a truncation/selection operator. The change is minor: under the settings in Table 1, the schedule in Equation 2 uses N(t) over the range [130] (Section 3.1.1), so the colony will lose three of its 130 individuals over a run, one at each of the three reduction events. It is nonetheless of a different nature, as the set of individuals retained now depends on their solution quality rather than their position in the population array. Since $\psi_i = f_i$ under Equation 9 (Section 3.2.2) the criterion of Equation 14 is, in the experiments described here, plain fitness based truncation. The difference from CPO is therefore not the use of memory in place of fitness, it's the use of any quality measure at all (CPO uses position in the array). The minimum in Equation 11 is only relevant to variants with a non-elitist acceptance rule. A second consequence is what happens to the individuals that are not retained. In the reference code they are simply ignored and so remain in the array to be used again when the bound increases at the start of the next cycle. Equation 14 is a selection operator, so individuals outside the selected subset are removed, and the active set can only shrink. With the settings of Table 1 it will be 130 for the first sweep, then 129, then 128 and then 127 from about 4×10^5 evaluations onwards. The cyclic schedule of CPO thus becomes a monotonic one, which is a change the present paper does not otherwise advocate, and Section 5.3 lists it as one of the properties that remain untested. The behavior resulting from Equation 14 is predictable, and we specify it before checking the results to ensure a fair assessment. If survival by index is random noise, this should appear most clearly in the scatter of results from independent runs and less clearly (or not at all) in the mean. The effect should be most pronounced on functions where CPO succeeds in most runs but fails in a small fraction, the bimodal signature of an outcome decided by chance. It should be almost imperceptible on functions that both algorithms solve exactly in every run, where there is nothing to gain. This is the behaviour that Section 4 is looking for. It should be made clear right at the outset that the experiments described in that section cannot test it. As Section 3.3.2 makes clear, the CPO baseline we ran also selected its survivors based on fitness, so the two programs are, in fact, identical in that respect. The prediction made above is therefore a property of Equation 14 as it stands with respect to CPO as published, not something that the data presented here can be said to establish.

3.2.5. Complete Algorithm, Cost, and Parameters

Algorithm 1 gives ECPO in full, with the lines that differ from CPO marked. Figure 1 shows the same structure as a flowchart.

Algorithm 1. ECPO. Lines marked ‣ are the modifications introduced in this paper; all other lines are CPO as published by Abdel-Basset et al. [4], with the counter t expressed in evaluations (Section 3.2.5)

Input: N', N_{min}, T, α, T_f, π, T_{max}, f, L, U, D
Output: G, $f(G)$
initialize $x_1, \ldots, x_{N'}$ by Eq. (1); evaluate; set $x_i^p \leftarrow x_i$, $f_i \leftarrow f(x_i)$
‣ initialize memory $p_i \leftarrow x_i$, $\varphi_i \leftarrow f(x_i)$ for all i [Eq. (10)]
$G \leftarrow \arg\min_i f_i$; $t \leftarrow 0$
while $t \le T_{max}$ and $f(G) \neq f^*$ do
  for $i = 1$ to $N(t)$ do
    if $r < 0.5$ then ▷ exploration
      apply Eq. (3) or Eq. (4) with equal probability
    else ▷ exploitation
      if $r < T_f$ then apply Eq. (6) ▷ odor
      else
        ‣ draw a_i by Eq. (12) [memory attractor]
        ‣ apply Eq. (13) ▷ attack
      end if
    end if
    repair bounds by Eq. (8); evaluate $f(x_i)$; $t \leftarrow t + 1$
    accept or restore by Eq. (9); update G
    ‣ update (p_i, φ_i) by Eq. (10)
  end for
  ‣ $N(t) \leftarrow \min(N(t-1),$ value of Eq. (2) at $t)$
  ‣ if $N(t) < N(t-1)$ then keep the $N(t)$ best by Eq. (14) and remove the others end if
end while
return G, $f(G)$

The cost of the addition is negligible and can be given precisely. Storage increases by one position vector and one scalar per individual, a total of O(N'D), which is of the same order as the population itself. Equation 10 adds one comparison per evaluation. Equation 12 adds one uniform draw from the fraction of trials affected. At each reduction event Equation 14 will sort at most 130 merit values (an O(N log N) operation). Under the schedule of Section 3.1.1 N(t) is halved three times per run, so the sorting is performed three times and the cost is a few thousand comparisons against the 10^6 objective evaluations in the budget. None of the modifications consume a function evaluation. The two programs did not use the same budget for a reason unrelated to the modifications: the ECPO counter starts after the initial colony has been evaluated. A run of ECPO therefore used 1,000,131 evaluations against the 1,000,000 of the baseline (Section 3.3.2).

The asymptotic complexity of the algorithm is unchanged at $O(T_{\max} \cdot D)$. All the control parameters are given in Table 1. The values of α, T_f and T are those of the reference implementation of CPO. N' and $N_{\min}$ are ours, and π is the parameter we have introduced here. The baseline program settings differ from all of the above and are provided in the final column of Table 1 and in Section 3.3.2. No parameter was tuned per function or per suite. The value of π was set a priori at 0.2 because the memory attractor should perturb the attack operator rather than replace it and was not altered at any point thereafter. In practice π = 0.2 means that the attack operator is that of CPO in 4 out of 5 cases and that approximately 2% of all trials are moved to the memory (Section 3.2.3). The two extremes of the range illustrate what is at stake. At π = 0 the attack operator is that of CPO. At π = 1 no attack trial is anchored at G^t and this removes the one operator responsible for concentrating the colony on the incumbent, which should be detrimental to convergence on unimodal landscapes where a single attractor is the correct design. A value of π well below 0.5 will keep the incumbent as the dominant operator while still giving every individual an anchor of its own on a minority of trials. We did not tune π, and the sensitivity of the results to it is an open question (Section 5.3).

The correspondence among the equations, Algorithm 1, and the MATLAB code for ECPO is very direct, with five points to note. First, Equation 2 is the direct translation of the statement new_size=fix(N_min+(N-N_min)*(1-(rem(t,Tmax/T)/Tmax/T))), which is the statement that recomputes the number of active individuals after each sweep (Algorithm 1, line 19). MATLAB evaluates rem (t,Tmax/T)/Tmax/T from left to right so the remainder is divided by T_max·T and that is why the actual floor is 127 rather than N_min (Section 3.1.1). Second, the statement is guarded by the condition new_size < Pop_size, and individuals outside the retained subset are simply removed from the arrays, so the active set never grows again (Section 3.2.4). Third, the mask u_1 is drawn once per trial, as is u_2, which, as the reference code generates it, is the zero vector (Section 3.2.3); the coefficient r_7 of Equations 7 and 13, however, is drawn once per sweep and is common to all the trials of that sweep in both codes. Fourth, the evaluation counter t is incremented once per objective evaluation (line 15), so that T_max and the schedule in Equation 2 are both expressed in terms of evaluations. In contrast, in the reference code the counter is advanced once per generation. The counter is set to zero after the initial colony has been evaluated, which is why a run performs 1,000,131 evaluations rather than 10^6. Fifth, ECPO only adds or modifies the statements that implement Equations 10, 12 to 14 as marked ‣ in Algorithm 1. The baseline program is a separate file and is described in Section 3.3.2.

3.3. Experimental Setup

3.3.1. Benchmark Suites

ECPO was evaluated on the four IEEE CEC single-objective bound-constrained suites: CEC2014 [20], CEC2017 [21], CEC2020 [22] and CEC2022 [23]. Together they form 81 functions, all shifted and rotated such that the optimum is not at the origin and not on any of the coordinate axes, and each with a known optimum f_q^*. For CEC2014 and CEC2017, the optimum for function q is 100q. For CEC2020 and CEC2022 the functions in the suites are renumbered so the optima are as given in ↴ Tables 2 and 3 and are not 100q. Section 3.3.2 explains why this is important for the stopping criterion. The suites are used in conjunction rather than individually for two reasons. The first is coverage: they cover the unimodal, simple multimodal, hybrid, and composition classes. The composition functions in particular are constructed by stacking several rotated basins on top of each other and are the class of functions on which single leader algorithms are notoriously bad. This variety is consistent with recommendations to choose test problems that reflect the goals of the comparison [24]. The second reason is independence: the four suites were produced by overlapping groups over an 8-year period, so agreement across all four is a much stronger result than agreement on any one of them. We used the first 30 functions of CEC2014, the first 29 of CEC2017, all 10 of CEC2020, and all 12 of CEC2022 (81 in total). CEC2017 F30 is therefore not included. F2 of that suite is described in the technical report as being unstable in higher dimensions, and we did run it, including it in the results. It is simply omitted from the comparison in Table 4 as the table we are comparing to doesn't include it.

3.3.2. Protocol

All experiments were run at D=10 with the search domain $[-100,100]^D$ as specified by the suites. The budget of function evaluations for each algorithm was set to $T_{\max} = 10^6$ and the algorithms were run 30 times from independent initializations. The reduction schedule is on the same scale as the budget (in terms of the number of evaluations, not generations), as per Equation 2. A run terminates when either the budget is exhausted or the incumbent reaches the value 100q for function q (tested for exact equality).

That test is predicated on the assumption (as stated in Section 3.3.1) that 100q is the optimum of every function in every suite. It is the optimum of CEC2014 and CEC2017, and of CEC2020 F1, and on those 60 functions, a run that reaches the optimum will stop there; the equality test is valid in those cases, as the optima are integer multiples of 100 and are attained bit exactly if attained at all. It is not the optimum of CEC2020 F2 to F10 or of any of the CEC2022 functions (Tables 2 and 3) where the suites place the optima elsewhere; on those 21 functions 100q is below the optimum so the test can never trigger and every run will consume the full budget.

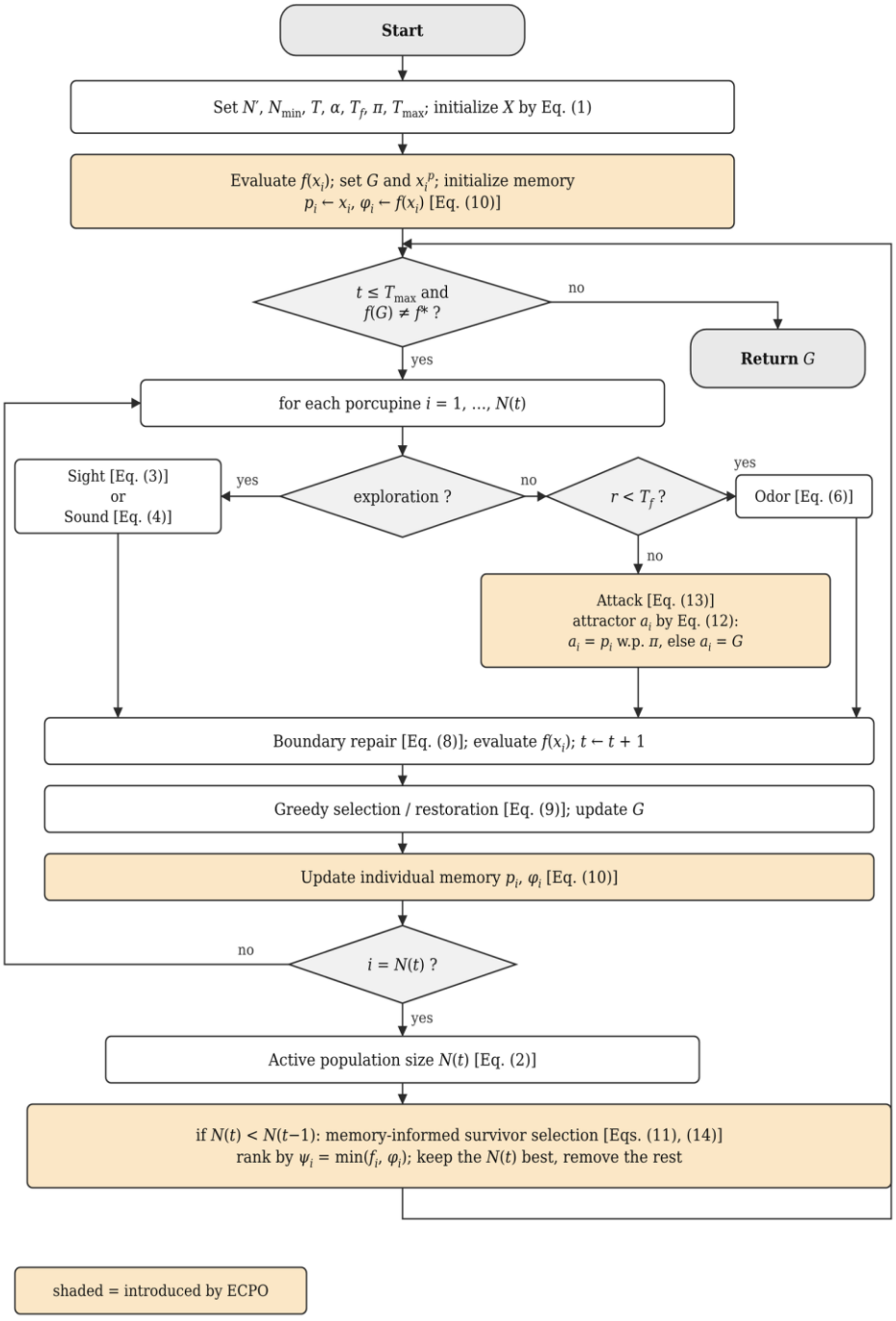


Figure 1. Flowchart of ECPO. Shaded blocks are the components introduced in this paper: the individual memory of Equation 10, the memory-based attractor of Equations 12 and 13, and the memory-informed survivor selection of Equation 14. All remaining blocks are unchanged from CPO [4]

Table 2. Results on the CEC2020 suite (D = 10, 30 runs, $T_{max} = 10^6$)

F	f*	CPO Mean	CPO Std	ECPO Mean	ECPO Std	p (Welch)	Sig.
F1	100	100.0000	0.00E+00	100.0000	0.00E+00	-	=
F2	1100	1116.4315	32.03	1104.6396	7.21	5.79E-02	=
F3	700	711.5789	0.77	710.8418	0.60	1.13E-04	+
F4	1900	1900.3280	0.24	1900.4067	0.14	1.24E-01	=
F5	1700	1700.7944	0.83	1700.2030	0.13	5.49E-04	+
F6	1600	1615.9720	51.69	1600.0291	0.04	1.02E-01	=
F7	2100	2100.0014	0.00	2100.0004	3.67E-04	3.97E-05	+
F8	2200	2283.3333	37.90	2286.6667	34.57	7.23E-01	=
F9	2400	2610.3033	105.35	2560.9188	107.15	7.70E-02	=

F10	2500	2922.5941	22.76	2906.9464	18.52	5.04E-03	+

Table 3. Results on the CEC2022 suite (D = 10, 30 runs, $T_{max} = 10^6$)

F	f*	CPO Mean	CPO Std	ECPO Mean	ECPO Std	p (Welch)	Sig.
F1	300	300.0000	0.00E+00	300.0000	0.00E+00	-	=
F2	400	403.0867	13.00	400.2658	1.01	2.45E-01	=
F3	600	600.0000	0.00E+00	600.0000	0.00E+00	-	=
F4	800	809.6511	4.86	802.8854	1.18	1.84E-08	+
F5	900	900.0000	0.00E+00	900.0000	0.00E+00	-	=
F6	1800	1800.0720	0.06	1800.0822	0.09	6.04E-01	=
F7	2000	2000.0000	0.00E+00	2000.0000	0.00E+00	-	=
F8	2200	2200.2763	0.09	2200.0802	0.04	3.98E-13	+
F9	2300	2529.2844	0.00E+00	2529.2844	0.00E+00	-	=
F10	2400	2513.6967	35.06	2500.1601	0.03	4.32E-02	+
F11	2600	2626.6667	101.48	2600.0000	0.00E+00	1.61E-01	=
F12	2700	2865.7127	6.55	2864.2255	1.00	2.28E-01	=

Table 4. Reproducibility check on CEC2017. Mean error with respect to the benchmark optimum over the 28 functions common to both tables (F2 is excluded, as it is from the published table), and the outcome of comparing each in-house algorithm against the published CPO values

Algorithm	Mean error to f*	Better / Tie / Worse vs published	Wilcoxon p
Published CPO [4]	82.1989	-	-
Re-implemented CPO	92.0442	1 / 9 / 18	8.38E-04
ECPO (proposed)	82.7350	6 / 13 / 9	8.65E-01

No run terminates before reaching the optimum, and early termination is available for 60 of the 81 functions, not for the other 21. That is what gives rise to the pattern in Table 5. The stopping rule is the same in both programs, so neither is at an advantage. Both programs are derived from the reference implementation of CPO [4] and use its boundary repair (Equation 8), its acceptance rule (Equation 9), MATLAB's random number generator, and an initialization routine which draws the colony uniformly in the box as per Equation 1 (although the baseline has its own copy of that routine rather than calling the shared one); ECPO differs from that reference only in the statements marked ‣ in Algorithm 1. They are, however, two separate files, and the paragraphs below report where the baseline departs from ECPO in ways that the modifications do not account for. This is important both for the argument in Section 3.4.3 and for the interpretation of Section 4.9. Both algorithms were run in MATLAB R2024a on a laptop with an Intel Core i5-5200U processor (2.20 GHz), 6GB of RAM, and 64-bit Windows 10, with no other user load. Their run times are therefore comparable with each other but not with those reported elsewhere.

Table 5. Mean wall-clock time per run in seconds, averaged over the functions of each suite. The final column counts functions on which ECPO was faster. Runtimes include the effect of early termination at the known optimum (Section 3.3.2)

Suite	CPO (s)	ECPO (s)	Change %	ECPO faster
CEC2014	13.171	12.344	-6.28	11/30
CEC2017	8.364	7.789	-6.88	21/29
CEC2020	8.465	8.352	-1.34	4/10
CEC2022	10.069	9.961	-1.07	8/12
All suites	10.410	9.867	-5.21	44/81

The baseline implementation. The paragraph above describes what the two programs have in common. They differ in more than just the two modifications in Section 3.2, and that difference is relevant to every comparison in this paper, so we lay it out in full. The baseline is a second MATLAB file, derived from the reference implementation of CPO [4] and used exactly as it is for all the runs reported in this paper. Having gone through both files line by line as part of preparing this revision, we identified six differences beyond the memory attractor of Equations 12 and 13, the first three of which are relevant to the search.

(i) Control parameters. The baseline was run with $N_{\min} = 80$, $\alpha = 0.1$ and $T_f = 0.2$, against $N_{\min} = 120$, $\alpha = 0.2$ and $T_f = 0.8$ for ECPO (Table 1). The value of T_f is the most consequential of the three: the physical-attack operator is applied on a fraction $(1 - T_f)/2$ of the trials, so the baseline attacks on 40% of its trials and ECPO on 10%. Given that this is the operator which pulls the colony towards the single incumbent, it's not surprising that the baseline is the more convergent of the two in

that respect. The value of α is the lower bound on the attack coefficient, so the baseline also takes slightly less aggressive attack steps.

(ii) Population schedule. The baseline works out the ratio of Equation 2 with the division in the parentheses so that the ratio is 1 at the end of a cycle and the count drops to $N_{\min}$. Its active population therefore drops from 130 to 80 over the first 5x10^5 evaluations in fifty steps. Then it remains at 80 for the remainder of the run, whereas the active population of ECPO is around 127 throughout (Section 3.2.4). For most of a run, the baseline is therefore searching with about two-thirds of the individuals that ECPO has.

(iii) Survivor selection. The baseline doesn't exclude individuals based on their position in the array. When its count falls, it will sort the colony by fitness, keep the fittest, and eliminate the rest, which is precisely what Equation 14 does, and, for the reason given in Section 3.2.4, the two criteria are in fact the same under the acceptance rule of Equation 9. The experiments described in Section 4 therefore don't represent a comparison between memory-informed survivor selection and selection by array position; in both cases, selection is based on quality, and the second modification is not applied in those runs.

(iv) Budget. The baseline counts the initial colony against the budget and terminates at exactly 10^6 evaluations; ECPO commences its counter after the initial colony and performs 1,000,131 (Section 3.2.5). The excess is 0.013% and is reported for completeness.

(v) Ties at the incumbent. The baseline replaces the colony best only on strict improvement; ECPO also on equal value, so on a plateau the two move their incumbent differently.

(vi) Form of the stopping test. ECPO tests the incumbent against 100q for equality, and the baseline test checks whether it still exceeds 100q. They're equivalent in this instance, as the objective of a CEC function can't fall below its optimum, so the baseline can only leave its loop at a value it could have attained via equality. We record the difference as the two tests are not in the same form.

Taken together (i) to (iii) mean that this is not a matched comparison and we make it very clear what follows. Two of those factors, on their own, would be expected to preserve diversity in ECPO relative to the baseline: a larger active population throughout the run and an operator rate that concentrates the colony on a single point, which is four times lower. The third factor entirely removes a modification from the comparison. The differences reported in Section 4 are therefore real measurements of the two programs. Still, they are not attributable to individual memory, and we make no such attribution anywhere in what follows. The comparison that would isolate the memory is the one described in Section 5.3: the baseline re-run with the same parameters and schedule as ECPO, together with the ablation that separates Equations 13 and 14. Neither of those could be completed in the revision period. We report the mismatch rather than hide it because the code will be deposited with the paper and a reader who opens it would see it immediately and because a measurement whose cause is unidentifiable is still a measurement worth recording if you label it as such.

3.4. Performance Measures and Statistical Methodology

This subsection defines quantities that are reported in Section 4. It also addresses a question that is rarely discussed in comparative studies of metaheuristics, and that we could not resolve from the literature: should an improvement be measured against the global optimum defined by the benchmark, or against the values published for the baseline algorithm? Section 3.4.3 shows that the question has two distinct parts, each with a different answer.

3.4.1. Error with Respect to the Benchmark Optimum

For algorithm A running on function q, the best objective value reached in run k is $f_{\mathrm{A,q}}^{(k)}$, and the sample mean, median, best, and worst over the runs are denoted accordingly for $f_{\mathrm{A,q}}^{(k)}$. The most important measure of accuracy is the mean error to the optimum, as defined by the suite:

$$E_{\mathrm{A,q}} = (1/R)\sum_{k=1}^{R} f_{\mathrm{A,q}}^{(k)} - f_q^* \tag{15}$$

Equation 15 is an affine transformation of the mean fitness and preserves its ordering, so no comparison based on it can be affected by the choice. Its merit is comparability: f_q^* is fixed by the benchmark's definition and is identical for every author who uses that benchmark, so E is a quantity that can be read across papers.

3.4.2. The Error Floor and the Corrected Error

Equation 15 has one flaw and on the CEC suites that is a major one. For many of the functions (particularly the composition functions) in our experiments, no run of either algorithm reached the suite optimum at D=10, and much of the residual seems to be due to the function as instantiated at that dimension rather than to any actual difference between the two algorithms. Hence reporting E alone effectively embeds a fixed offset into every comparison and squashes the apparent difference between the algorithms. To separate the two effects, we define, for each function, a best-observed reference and the associated floor:

$$f_q^{\mathrm{ref}} = \min_{A\in\{\mathrm{CPO},\,\mathrm{ECPO}\}} \min_k f_{A,q}^{(k)}, \rho_q = f_q^{\mathrm{ref}} - f_q^* \geq 0 \tag{16}$$

$$\tilde{\mathrm{E}}_{\mathrm{A,q}} = (1/R)\sum_{k=1}^{R} f_{\mathrm{A,q}}^{(k)} - f_q^{\mathrm{ref}} = E_{\mathrm{A,q}} - \rho_q \tag{17}$$

The floor ρ_q is zero in all cases where at least one run of either algorithm achieved the nominal optimum and is positive otherwise. In our experiments, $\rho_q > 0$ on 41 of the 81 functions, and in some cases, it is quite large (397.74 on CEC2017 F25, 300.00 on CEC2017 F23, F26 and F28 and 229.28 on CEC2022 F9). For such functions, the difference between two

algorithms can be an order of magnitude smaller than the floor, so E effectively compresses the visible effect of any modification. The reference f_q^{ref} is, of course, the best value attained in these experiments (not an independently established optimum or a theoretical bound) and could, in fact, be lowered by a third algorithm, a larger budget, or simply more runs. $\tilde{E}$ is therefore a descriptive, experiment-dependent measure of how much of the distance to the best observed value each algorithm closed, and it tells us nothing about what is in principle achievable. E is the primary measure and the only one that can be compared with published results; $\tilde{E}$ is reported alongside it as a secondary measure. The reference cancels out in any paired difference (Equation 18), so the signed-rank tests of Section 4.6 are identical under either measure. What the reference does change is the magnitude of the floor corrected reductions and of the PRI($\tilde{E}$) and MGI values in Table 6. Section 4.7 quantifies the gap between the two measures.

Table 6. Aggregate performance indices. E is the mean error with respect to the benchmark optimum, Equation 15; $\tilde{E}$ is the floor-corrected error, Equation 17; PRI values are means over the functions of the suite, Equation 22; MGI combines them with weights (0.5, 0.3, 0.2), Equation 23

Suite	E (CPO)	E (ECPO)	Red. %	$\tilde{E}$ (CPO)	$\tilde{E}$ (ECPO)	Red. %	PRI($\tilde{E}$)	PRI(σ)	PRI(t)	MGI
CEC2014	53.7231	46.0893	14.21	23.6807	16.0470	32.24	0.4631	0.3720	0.0497	0.3531
CEC2017	88.8703	79.8821	10.11	15.8623	6.8741	56.66	0.4111	0.3536	0.0609	0.3238
CEC2020	76.1337	67.0652	11.91	35.3101	26.2417	25.68	0.3605	0.3366	0.0116	0.2835
CEC2022	45.7039	41.4153	9.38	4.6958	0.4072	91.33	0.3476	0.3634	0.0029	0.2834
ALL	67.8854	60.0851	11.49	19.5047	11.7045	39.99	0.4147	0.3598	0.0421	0.3237

3.4.3. Reference and Baseline are Different Questions

Consider a paired test of the per-function differences between two algorithms, computed from errors measured relative to some reference f_q^r. The difference is

$$d_q = \left(\bar{f}_{\text{CPO,q}} - f_q^r\right) - \left(\bar{f}_{\text{ECPO,q}} - f_q^r\right) = \bar{f}_{\text{CPO,q}} - \bar{f}_{\text{ECPO,q}} \tag{18}$$

and the reference cancels identically. Every statistic computed from the d_q — signed ranks, sums of ranks, p-values — is therefore invariant to whether errors are measured against the CEC optima, against a best-observed reference, or not at all. The choice between the global optimum and any other reference is a reporting decision that affects the interpretability of the magnitudes and is not a statistical decision. Equation 18 settles that half of the question.

What does not cancel is $\bar{f}_{\text{CPO,q}}$, and this is where the substantive choice lies. Two quantities are in principle available: the mean produced by a re-execution of CPO under the conditions of the present study, and the mean printed in the source publication. They are not interchangeable. A re-execution shares the initialization routine, boundary handling, random number generator, stopping tolerance, arithmetic, and machine with the proposed algorithm, so a paired difference isolates the effect of the modification. A published value shares none of these, so a paired difference confounds the modification with every difference between two implementations. Since the point of the experiment is to estimate the effect of the modification, the baseline must be the re-execution. Most careful comparative studies do re-run their baselines, but the reason is rarely made explicit, and the practice of comparing against printed tables remains common. Section 4.9 quantifies the cost using our own data.

The published values continue to play a role, and we report them. Comparing a re-implementation against them is a reproducibility check on the implementation, which is a legitimate and useful claim, distinct from a hypothesis test about a modification. Reporting the two side by side is what allows a reader to see which claim is being made.

3.4.4. Hypothesis Tests

Two tests are used, at two levels. At the level of the suite, the multi-problem Wilcoxon signed-rank test [25] is applied to the paired per-function differences d_q of Equation 18, treating each function as one observation, as recommended for comparing algorithms over multiple problem sets [26, 27]. Let n be the number of functions with $d_q \neq 0$ and let R_q be the rank of $|d_q|$ among those:

$$R^+ = \Sigma_{d_q>0} R_q, R^- = \Sigma_{d_q<0} R_q \tag{19}$$

With R+ accumulating the evidence in favour of ECPO; functions with $d_q = 0$ are discarded before ranking as in Wilcoxon's original procedure [25]. It's a non-parametric test that makes no assumptions about the distribution of the differences (this is important, given that the errors on the CEC suites span several orders of magnitude across the functions). One sided p-values are reported against the alternative that ECPO has the smaller error. At the level of the individual function, we now have summary statistics from the 30 runs rather than the runs themselves, so a rank-sum test on them is out of the question (Section 3.4.4 returns to this). We therefore use Welch's unequal-variance t-test [28], which requires only means, standard deviations, and sample sizes:

$$t_q = \left(\bar{f}_{\text{CPO,q}} - \bar{f}_{\text{ECPO,q}}\right) / \sqrt{s^2_{\text{CPO,q}}/R + s^2_{\text{ECPO,q}}/R} \tag{20}$$

with the Welch–Satterthwaite degrees of freedom. Effect size is reported as Hedges' g [29], the bias-corrected form of Cohen's d [30]:

$$g_q = \left(\bar{f}_{\mathrm{CPO,q}} - \bar{f}_{\mathrm{ECPO,q}}\right)/s_p, s_p = \sqrt{\left(s^2_{\mathrm{CPO,q}} + s^2_{\mathrm{ECPO,q}}\right)/2}, \tag{21}$$

corrected by the factor $1 - 3/(4 \cdot (2R - 2) - 1)$

The limitation of Equation 20 has to be mentioned. The distribution of the final fitness values across the runs of a metaheuristic is generally skewed and, on functions with multiple basins, is multimodal. The normality assumption underpinning the Welch approximation is therefore not valid. We use it because the individual run values are not available and the only thing, we have is the summary statistics in Tables 2, 3, 7 and 8. We treat the verdicts of the per- function test as indicative rather than definitive. There is one caveat to note here, which was discovered whilst reviewing the code for this revision. The driver script did, in fact, carry out a Wilcoxon rank-sum test on the raw runs of each function as they were generated and wrote the p-value to the results file at that time. A Wilcoxon rank-sum test was therefore not possible at the time and is not reported here. The raw runs were not saved, so it' s not possible to recompute or verify it. We report the Welch test because it is the test we can recompute from the data the paper actually publishes. Section 5. 3 lists the rank-sum p-values, which should have been saved. The conclusions of the paper are based on the suite-level signed-rank tests of Equation 19, which make no distributional assumptions and would not change if the per-function tests were removed. As there are many per-function tests, we also apply Holm's step-down procedure to control the family-wise error rate, as recommended by Derrac et al. [27]. A test is possible for 60 of the functions; for the other 21, both algorithms return the same value in every run. These are not the 21 functions of Section 3.3.2 on which early termination is not available. The two sets are of the same size and otherwise unconnected. Of the 33 improvements that are significant at the 5% level, 22 remain significant after Holm's correction across the 60 tests, and none of the 60 are significant against ECPO, either before or after correction. Supplementary Table S1 lists for each function the mean difference with its 95% Welch confidence interval and the unadjusted and Holm adjusted p- values, all of which are computed from the summary statistics in Tables 2, 3, 7 and 8

Table 7. Results on the CEC2014 suite (D = 10, 30 runs, $T_{\max} = 10^6$). The last column gives the result of the Welch comparison of Equation 20 at the 5% level

F	f*	CPO Mean	CPO Std	ECPO Mean	ECPO Std	p (Welch)	Sig.
F1	100	100.0000	0.00E+00	100.0000	0.00E+00	-	=
F2	200	200.0000	0.00E+00	200.0000	0.00E+00	-	=
F3	300	300.0000	0.00E+00	300.0000	0.00E+00	-	=
F4	400	416.2276	16.58	404.2006	10.49	1.53E-03	+
F5	500	510.6551	10.14	508.6602	10.07	4.48E-01	=
F6	600	600.7218	0.96	600.0000	0.00E+00	2.76E-04	+
F7	700	700.0007	0.00	700.0000	0.00E+00	1.63E-01	=
F8	800	800.0000	0.00E+00	800.0000	0.00E+00	-	=
F9	900	901.7578	3.38	900.1990	0.40	1.79E-02	+
F10	1000	1000.0104	0.02	1000.0000	0.00E+00	2.26E-02	+
F11	1100	1145.9096	63.41	1109.5188	10.84	4.14E-03	+
F12	1200	1200.1857	0.04	1200.0872	0.05	6.54E-12	+
F13	1300	1300.0997	0.02	1300.0504	0.02	3.22E-14	+
F14	1400	1400.0975	0.03	1400.0563	0.02	8.15E-07	+
F15	1500	1500.4698	0.11	1500.3978	0.10	1.12E-02	+
F16	1600	1601.1834	0.49	1600.3115	0.21	5.95E-11	+
F17	1700	1700.8065	0.59	1700.2255	0.15	8.88E-06	+
F18	1800	1800.0124	0.01	1800.0025	0.00	1.87E-09	+
F19	1900	1900.0013	0.01	1900.0013	0.00	9.98E-01	=
F20	2000	2000.0282	0.03	2000.0029	0.01	7.65E-05	+
F21	2100	2100.0219	0.08	2100.0006	6.38E-04	1.50E-01	=
F22	2200	2204.0216	21.98	2200.0015	0.00	3.25E-01	=
F23	2300	2500.0000	0.00E+00	2500.0000	0.00E+00	-	=
F24	2400	2513.9116	29.82	2500.9932	2.27	2.48E-02	+
F25	2500	2673.9115	38.94	2620.9641	30.54	2.72E-07	+
F26	2600	2700.0914	0.02	2700.0494	0.01	3.74E-12	+
F27	2700	2793.9056	100.94	2707.8128	36.30	9.19E-05	+
F28	2800	2993.3345	25.37	3000.0000	0.00E+00	1.61E-01	=

F29	2900	3089.3485	47.16	3075.1653	58.52	3.06E-01	=
F30	3000	3464.9788	7.04	3453.9795	47.97	2.24E-01	=

Table 8. Results on the CEC2017 suite (D = 10, 30 runs, $T_{max} = 10^6$)

F	f*	CPO Mean	CPO Std	ECPO Mean	ECPO Std	p (Welch)	Sig.
F1	100	100.0000	0.00E+00	100.0000	0.00E+00	-	=
F2	200	200.0000	0.00E+00	200.0000	0.00E+00	-	=
F3	300	300.0000	0.00E+00	300.0000	0.00E+00	-	=
F4	400	400.0000	0.00E+00	400.0000	0.00E+00	-	=
F5	500	500.4048	0.55	500.3317	0.75	6.71E-01	=
F6	600	600.0000	0.00E+00	600.0000	0.00E+00	-	=
F7	700	711.5086	0.87	710.8484	0.44	6.01E-04	+
F8	800	803.0512	4.27	800.1990	0.61	1.06E-03	+
F9	900	900.0000	0.00E+00	900.0000	0.00E+00	-	=
F10	1000	1028.6621	57.24	1013.5084	34.03	2.19E-01	=
F11	1100	1100.0000	0.00E+00	1100.0000	0.00E+00	-	=
F12	1200	1220.0136	45.03	1200.1405	0.16	2.21E-02	+
F13	1300	1300.1612	0.88	1300.0000	0.00E+00	3.26E-01	=
F14	1400	1400.0000	0.00E+00	1400.0000	0.00E+00	-	=
F15	1500	1500.0041	0.02	1500.0002	2.47E-04	2.84E-01	=
F16	1600	1600.2855	0.21	1600.0020	0.00	3.32E-08	+
F17	1700	1700.6276	3.11	1700.0000	0.00E+00	2.78E-01	=
F18	1800	1800.1898	0.18	1800.0226	0.07	2.71E-05	+
F19	1900	1900.0003	0.00	1900.0000	0.00E+00	3.26E-01	=
F20	2000	2000.0000	0.00E+00	2000.0000	0.00E+00	-	=
F21	2100	2213.7458	35.15	2203.4836	19.08	1.67E-01	=
F22	2200	2290.0000	30.51	2290.0000	30.51	1.00E+00	=
F23	2300	2600.3549	1.13	2600.1027	0.56	2.78E-01	=
F24	2400	2648.4926	93.67	2573.5068	105.91	5.22E-03	+
F25	2500	2915.0846	21.90	2903.9038	15.74	2.73E-02	+
F26	2600	2900.0000	0.00E+00	2900.0000	0.00E+00	-	=
F27	2700	3094.2503	10.78	3091.3787	2.42	1.64E-01	=
F28	2800	3215.6311	154.89	3100.0000	0.00E+00	3.14E-04	+
F29	2900	3134.7706	3.98	3129.1514	1.94	1.69E-08	+

3.4.5. Composite Descriptive Indices

The signed-rank test establishes whether an improvement exists but says nothing about its size, and the raw errors of Equation 15 cannot be averaged across functions because they differ by orders of magnitude. A scale-free paired index is needed. We use the paired relative improvement

$$\mathrm{PRI}(\theta) = (\theta_{\mathrm{CPO}} - \theta_{\mathrm{ECPO}})/(|\theta_{\mathrm{CPO}}| + |\theta_{\mathrm{ECPO}}|) \text{ with } \mathrm{PRI}(\theta) = 0 \text{ when } \theta_{\mathrm{CPO}} = \theta_{\mathrm{ECPO}} = 0 \quad (22)$$

where θ is any non-negative performance quantity. PRI is bounded in $[-1, +1]$, with positive values favoring ECPO. It is invariant to a common rescaling of θ, is defined when one of the two quantities is zero, and is defined by the stated convention when both are zero — the case that arises on the 21 functions on which both algorithms return the same value in every run, and that a ratio-based index would leave undefined. Averaging PRI over functions is, therefore, meaningful in a way that averaging raw errors is not. We report it for three quantities: the floor-corrected error $\tilde{E}$, the standard deviation σ over runs, and the mean wall-clock time $\bar{t}$ per run, giving an accuracy, a reliability, and an efficiency component.

For a single summary figure, we combine the three components into a memory gain index:

$$\mathrm{MGI} = 0.5 \cdot \mathrm{PRI}(\tilde{E}) + 0.3 \cdot \mathrm{PRI}(\sigma) + 0.2 \cdot \mathrm{PRI}(\bar{t}) \quad (23)$$

The weights represent a judgment rather than a measurement: accuracy is of course the most important, as it is the very objective of the optimization process; reliability is next, as a practitioner who can afford to run one experiment is just as concerned about the spread as they are about the mean, and runtime is the least important as it is the most implementation-dependent of the three. We state the weights explicitly rather than implicitly in the code, and we make the corresponding

caveat. MGI is a descriptive index, not a test statistic: it has no null distribution, no p-value can be assigned to it, and a different weighting would yield a different MGI. Its purpose here is simply to provide a summary of the direction and approximate magnitude of the change that is already established in Equation 19. No conclusions in this paper are dependent on it. It is a supplementary summary: the results are all interpreted on the mean error, the standard deviation over runs and the runtime with each of these reported separately in Tables 2, 3 and 5 to 11. Since all three components are positive on all four suites, any positive weighting will lead to a positive MGI on all suites; the ordering of the suites however does depend on the weights (CEC2020 and CEC2022 differ by 0.0001 in Table 6). This is yet another reason to treat MGI as a summary rather than a ranking.

Table 9. Outcome by function class, overall across the four suites. W / T / L count the number of wins, ties, and losses of ECPO; "σ decreased" counts the number of functions for which the standard deviation over 30 runs decreased; "σ = 0" counts the number of functions solved with zero dispersion by CPO and by ECPO, respectively

Class	Functions	W / T / L	Significant	σ decreased	σ = 0 (CPO → ECPO)	mean PRI(Ẽ)
Unimodal	7	0 / 7 / 0	0	0	7 → 7	0.000
Multimodal	27	20 / 7 / 0	14	18	7 → 10	0.418
Hybrid	23	16 / 4 / 3	9	18	4 → 7	0.573
Composition	24	18 / 4 / 2	10	16	3 → 6	0.380
All functions	81	54 / 22 / 5	33	52	21 → 30	0.415

Table 10. Outcome by the run-to-run dispersion of the baseline. Functions are partitioned using the baseline's own statistics only, before any ECPO result is consulted. The partition locates the difference between the two programs; it does not identify its cause (Section 3.3.2)

Group	Functions	ECPO improves	σ decreased	mean PRI(Ẽ)
High-dispersion ($\sigma_{CPO} > 0.5\ E_{CPO}$)	34	31 / 34	31 / 34	0.678
All remaining functions	47	23 / 47	21 / 47	0.224

Table 11. Multi-problem Wilcoxon signed-rank test of ECPO against CPO on paired per-function mean errors. W / T / L counts wins, ties, and losses of ECPO. One-sided p-values against the alternative that ECPO has the smaller error

Test suite	Functions	W / T / L	R+	R−	p-value	Significant at 5%
CEC2014	30	23/5/2	306	19	5.64E-05	Yes
CEC2017	29	18/11/0	171	0	9.82E-05	Yes
CEC2020	10	7/1/2	38	7	3.71E-02	Yes
CEC2022	12	6/5/1	27	1	1.56E-02	Yes
ALL	81	54/22/5	1671	99	1.49E-09	Yes

4. Results

CPO and ECPO were run on all 81 functions in accordance with the protocol of Section 3.3.2. Tables 2, 3, 7 and 8 give for each function the optimum f* as defined by the suite, the mean and standard deviation of the final objective value for both algorithms, the p-value of the Welch comparison of Equation 20 and a symbol summarising the outcome: "+" if ECPO is better at the 5% level, "−" if ECPO is worse at the 5% level and "=" otherwise. Over the four suites, ECPO has a lower mean on 54 functions, the same mean on 22, and a higher mean on 5; of these, 33 are significant in favor of ECPO at the 5% level (22 after Holm's correction, as per Section 3.4.4), and none are significant against it. Figure 2 is a scatter plot of the mean error, and Figure 3 is a scatter plot of the standard deviation of the error, with the bulk of the effect concentrated there. These figures are not repeated in the subsections that follow. In Section 3.2.4, the paper made a specific prediction about where the improvement would be found; Sections 4.1 to 4.4 examine each suite in light of that prediction, and Section 4.5 tests that prediction directly. Two caveats in relation to everything in this section. The first concerns what the comparison actually measures. The baseline was run with three different control parameters and a different population schedule to ECPO, and it selects its survivors by fitness in the same way as Equation 14 does (Section 3.3.2). The numbers that follow are therefore an accurate account of how these two programs performed, and nothing more. Where differences are seen, they accurately reflect the performance of the two programs. The memory of Equations 12 and 13 is one of several possible explanations for the difference, and, on the evidence available, is not the most likely. We do describe the differences and the patterns in those differences, and we have deleted from this section the statements in the submitted version that ascribed a result to one or the other of the modifications. The second caveat is significant. A number of the functions are referenced in the per-suite discussion as examples, even though their individual tests are not significant (again, typically due to one or two very outlying runs that inflate the standard deviation of CPO). We report the p-value in each case. These functions are again just descriptive examples of a pattern which is established at the suite level (Table 11); they are not independent evidence for the pattern.

4.1. CEC2014

The biggest suite is CEC2014, with eight composition and six hybrid functions, offering the most scope for a diversity-preserving modification to have an effect. ECPO improved the mean on 23 of the 30 functions, matched it on 5 and made it worse on 2, with 17 of the improvements significant at the 5% level. The five ties are of interest. They occur on F1, F2, F3, F8 and F23. For the first four of these, the two algorithms reach the optimum in all 30 runs; on F23, they both return the same value in all 30 runs without reaching the optimum. The unimodal functions (F1 to F3) have a single basin, so a single attractor is the correct design, and the attractor multiplicity in Equation 13 is irrelevant. That the ECPO incurs no gain or loss in this regime is exactly what the design of Equation 13 would predict, and it confirms that neither algorithm is at a disadvantage in the regime where a single attractor is correct. The real results are in the multi-modal functions. On F27 (a composition function), the mean error drops from 93.91 to 7.81, with a Hedges' g of 1.12 (Equation 21; p = 9.19E-05), the largest single improvement in the study; on F25, it drops from 173.91 to 120.96, with g = 1.49 (p = 2.72E-07). What these figures tell us is about the spread, not the mean. On F22 (a multi-modal function), the CPO mean is 2204.02 with a standard deviation of 21.98; the ECPO mean is 2200.00 with a standard deviation of 1.60E-03: CPO gets to the correct basin in most of the runs and gets it wrong in a few; ECPO gets it in all of the runs. The difference is not significant (p = 0.325): with so few outlying runs, the Welch test can't distinguish it from chance, and F22 is included here to illustrate the pattern rather than as evidence for it. F11 also repeats this pattern, but now with a lower error level (45.91 to 9.52) and a smaller dispersion (63.41 to 10.84). In both cases, the mean improves because the tail of failed runs is removed, not because the successful runs improve — the mode is unchanged. This is the pattern identified in Section 3.2.4 as being associated with survivor selection. It is not evidence for Equation 14 in this case, as the baseline also selects its survivors based on fitness (Section 3.3.2). The difference between the two programs on these functions is simply in how many individuals remain active and how often the attack operator is applied. F30 is a different case. The two algorithms are very similar in the mean, but the best run of ECPO reaches 3200.00, compared to 3462.69 for the best run of CPO (a full composition basin). The means aren't significantly different (p=0.224), and a best-run statistic is of no value here. The suite means for this function are given in Table 7. Both programs do, in fact, get to this basin in some runs and not in others, and it's a matter of which one does so more often that a best-run statistic can't answer. We note the difference without attempting to explain it. The two losses are worth a mention. The difference on F19 is 3e-06 and is just numerical noise. On F28, the difference is not: the mean of CPO is 2993.33 (SD 25.37), and the mean of ECPO is 3000.00 across all 30 runs. In this function, the baseline does, in fact, reach the better basin (2900.02) in a minority of runs, and ECPO never does so. ECPO is therefore the more consistent of the two and is consistently worse. Section 5.2 goes on to discuss what might be responsible for this and why the explanation offered in the submitted version fails when the two programs are compared in Section 3.3.2.

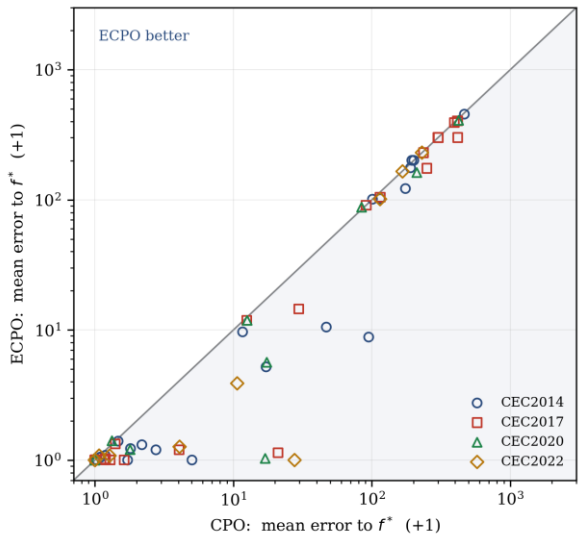


Figure 2. Mean error with respect to the benchmark optimum, ECPO against CPO, for all 81 functions. Both axes are shifted by one unit so that exactly solved functions can be shown on logarithmic scales. Points below the diagonal favor ECPO

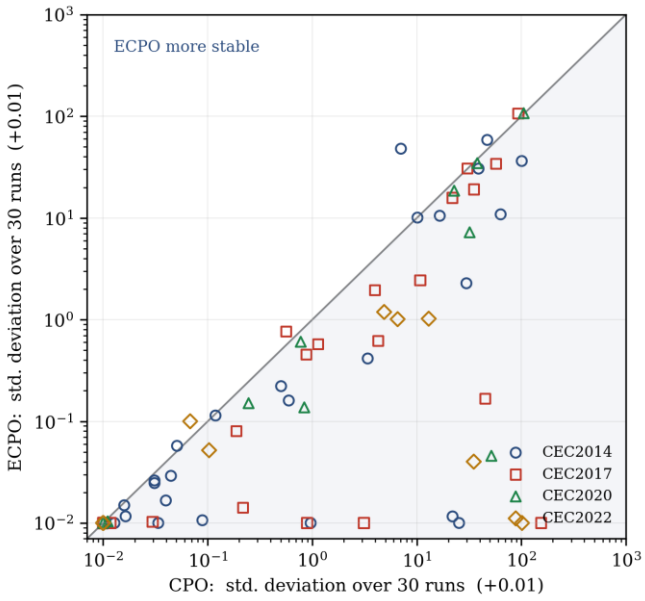


Figure 3. Standard deviation of the final objective value over 30 runs, ECPO against CPO, for all 81 functions, shifted by 0.01. The concentration of points along the lower edge corresponds to the 30 functions on which ECPO returns the same value in every run

4.2. CEC2017

CEC 2017 produces the best result among the four suites: ECPO improves 18 of the 29 functions, degrades none, with 9 of the improvements significant at the 5% level and a signed rank sum of R+ = 171 (vs R− = 0). The most interesting single result is F 28. The ECPO mean error is 300.00. 00 (std dev 0) as opposed to 415. 63 and 154.89. 89 for CPO. 300. 00 is in fact the best value that either algorithm achieves on this function at D = 10 (i. e., it is the reference value in Equation 16), so ECPO doesn' t in any way outperform CPO' s best run; it simply replicates that best run in all of its own. What has changed is the dispersion, not the achievable accuracy. F 12 is the same story: error drops from 20. 01 to 0. 14, dispersion from 45. 03 to 0. 16. On F13 and F17 ECPO achieves the optimum in all 30 runs, which CPO does not (though neither of those differences is significant on its own (p = 0. 326 and p = 0. 278)). Across the suite, the standard deviation is reduced for 16 functions and is zero for four of them (F13, F17, F19, and F28). A second type of improvement should be noted. On F 24 the error falls from 248. 49 to 173. 51 (p = 5. 22 E- 03) and on F 10 from 28.28.66 to 13.51 (not significant, p = 0. 219) in both cases with the distribution shifting rather than narrowing. These are improvements in the quality of the average run and are of the type expected from any difference in attractor diversity (i.e., a colony that keeps sampling regions not yet claimed by the incumbent ends up, on average, in a better basin). Both types of difference (narrowing and shifting) are present in the data. Which of the differences between the two programs is responsible for them is not established by these experiments (Sections 3.3.2 and 5.3). Of the eleven ties, nine are functions that both algorithms solve exactly in every run. On F 22 and F 26 both return the same mean and standard deviation without reaching the optimum. The fact that no function in this suite is degraded is the clearest evidence that the difference between the two programs has no systematic cost.

4.3. CEC2020

CEC2020 is a very small suite of ten rather difficult functions, and it gives the worst results: ECPO is better on 7, tied on 1, and worse on 2, with 4 significant improvements. The signed-rank test gives R+ = 38, R− = 7, and p = 3.71E-02, which just barely meets the conventional threshold. The test has very limited power with only 10 paired observations and this p-value should be interpreted as being consistent with the pooled result rather than as an independent confirmation of it. F6 is the clearest example of the reliability pattern observed in this study (although the Welch test is not significant, p = 0.102): the mean error decreases from 15.97 to 0.03, and the standard deviation decreases from 51.69 to 0.04. A standard deviation three times the mean can only occur if a few runs go very badly and the rest go very well and it is precisely these bad runs that ECPO eliminates. F9 is the other mechanism at work: ECPO achieves the exact optimum of 2400.00 in its best run (CPO never does) and the mean improves from 2610.30 to 2560.92 (p = 0.077). To find a basin that the baseline doesn't find in 30 runs is precisely what you'd expect of a search that preserves more attractor diversity; which of the two algorithms is responsible for this is not established here (Sections 3.3.2 and 5.3). The differences are both very small. On F4 the difference of 0.079 is trivial. On F8, the 3.33 difference between 2286.67 and 2283.33 is just a reflection of how often each algorithm ends up in the 2200 rather than the 2300 basin, and that's only a difference of one run in thirty. We record it but do not claim it is anything other than a very minor difference.

4.4. CEC2022

On CEC2022 ECPO improves on 6 of the 12 functions, ties on 5 and loses on 1 (R+ = 27, R− = 1, p = 1.56E-02). The sheer number of ties in this suite is characteristic of the suite itself: on F1, F3, F5, and F7, both algorithms reach the optimum in every run, and on F9, they return the same value in every run. So, in nearly half the cases there is no headroom at all. Where there is headroom the effect is large and again dominated by dispersion. On F11, the CPO mean error is 26.67, with a dispersion

of 101.48 (four times the mean), i.e., it solves the function in most runs and fails badly in the rest. The difference is not significant ($p = 0.161$) for the same reason as on CEC2014 F22. F10 is the same story, the dispersion drops from 35.06 to 0.03. The mean error drops to 0.27 on F2 and to 2.89 on F4 with substantially reduced standard deviations in both cases. The difference is significant on F4 ($p = 1.84E-08$) but not on F2 ($p = 0.245$). The single loss is 0.010 and that's not meaningful. This suite shows the largest floor corrected gain of the four for arithmetic rather than algorithmic reasons: the floor corrected errors are so small here that removing most of what's left is a large percentage. The absolute and corrected forms of the measure in Equation 17 should therefore be read together rather than one in place of the other.

4.5. Where the Improvement Occurs, and whether it Occurs where Predicted

The per-suite discussion points to two regularities: gains are obtained on multi-basin rather than unimodal landscapes and gains are predominantly attributable to the elimination of failed runs. Both of these can be tested directly. Table 6 breaks down the 81 functions by function class. The unimodal row is straightforward: 7 functions, 7 ties, no wins, no losses and a mean PRI of precisely zero (as both algorithms solve each one exactly in every run). Both algorithms solve all of them exactly so again there is nothing for any of the differences in Section 3.3.2 to act upon; this is consistent with Section 3.2.3 but is not a test of it. The other classes all show some benefit. The highest mean PRI is achieved in the hybrid class (0.573), with its sub-functions in disjoint coordinate blocks. A search that can maintain several separate good positions would be expected to do well on such a class. Still, the experiments don't demonstrate that this is the reason for the difference between the two algorithms. The composition class has the lowest mean PRI of the three (0.380), but it does have 18 wins out of 24 functions. On these functions, the floor-corrected errors of both algorithms are still large relative to the difference between them, so the wins are fairly modest in terms of relative improvement.

The second regularity is where the two programs differ most. Section 3.2.4 anticipates that a difference in survivor selection would be manifest in the functions whose outputs vary most from run to run, and the same holds for differences in population size or in the attack operator's rate. The partition below therefore separates the functions on which any of these could have an effect from those on which none of them would have much scope.

Of the 34 functions where the CPO outcome distribution is spread out, ECPO does better on 31 and reduces the spread on 31 of them, with a mean PRI of 0.678. On the other 47, it does better on 23, with a mean PRI of 0.224 (a third of the value in the high-dispersion group). The gap between the two groups is very large and very consistent: it's the very thing that causes the outcome of a run to vary that is being separated, and it's very little else. Overall, the standard deviation is reduced for 52 of the 81 functions, and the number of functions that return zero dispersion in every run has increased from 21 to 30. Figure 3 shows the same thing: the points are almost entirely below the diagonal with a very dense band along the lower edge where the dispersion of ECPO has gone to zero.

One caveat here. The partition in Table 7 is post hoc, as it was calculated after the experiments had been run (even though the criterion is based solely on baseline data and the prediction was made before any examination of the results). The partition tells you where the two programs differ, it doesn't tell you why. It is consistent with the mechanism proposed in Section 3.2.4 and with the differences in population schedule and attack rate described in Section 3.3.2. Which of those is the right explanation requires the matched re-execution and ablation described in Section 5.3.

4.6. Suite-level statistical analysis

Table 11 reports the multi-problem Wilcoxon signed-rank test of Equation 19 on the paired per-function mean errors, for each suite and for the pooled set. By Equation 18, these statistics are invariant to the reference point, so identical values would follow from raw fitness or from floor-corrected error.

The pooled test yields $R+ = 1671$ and $R- = 99$ across 59 non-tied functions, with $p = 1.49E-09$. Each suite also rejects the null hypothesis at the 5% level, though with very different margins: decisively for CEC2014 and CEC2017, and marginally for CEC2020 and CEC2022. The margin tracks the number of functions rather than the size of the effect — CEC2022 has the largest floor-corrected improvement of the four and the second-weakest p-value, which is what one expects of a rank test with 10 to 12 observations. We report the suites separately and in a pooled analysis because consistency of direction across four independently designed benchmark sets is a stronger claim than a single pooled p-value. That claim would be lost in the pooling. The tests use one paired value per function and make no distributional assumptions so are not subject to the limitations of the per function Welch tests discussed in Section 3.4.4.

4.7. Aggregate indices

Table 6 collects the descriptive indices of Section 3.4.5, and Figure 4 displays the three PRI components and the combined index.

The two error columns differ by a factor of three and that's the point of Section 3.4.2. Against the nominal optima, ECPO reduces the mean error across all 81 functions from 67.885 to 60.085 (an 11.5% reduction). Against the best value observed in these experiments the same runs give 19.505 to 11.705 (a 40.0% reduction). The first of these is the key result: it's against a fixed, published optimum and is therefore directly comparable to other work. The second is secondary and merely descriptive. It's a measure of how much of the gap to the best observed value was closed, and since that value is derived from our own 60 runs per function, it would differ for a different set of algorithms, budgets, or runs. It's not evidence that the remaining error is unattainable. We report it because, in the functions where neither algorithm came close to f*, the nominal error is dominated by a common offset, which masks the difference between the two. The PRI decomposition shows us what actually improved. $PRI(\tilde{E}) = 0.415$ pooled is a significant accuracy gain, $PRI(\sigma) = 0.360$ is a reliability gain of almost the same size and $PRI(\bar{t}) = 0.042$ is a small runtime gain which Section 4.8 links to the stopping rule rather than to cheaper

evaluations. That the accuracy and reliability components are of comparable magnitude is the quantitative expression of the observation that runs through Sections 4.1 to 4.5: what separates these two programs is as much in terms of consistency as it is in terms of precision. The pooled MGI is 0.324 and across the suites it ranges from 0.283 (on CEC2020 and CEC2022) to 0.353 (on CEC2014). As noted in Section 3.4.5, MGI is purely descriptive. It is a summary of a direction already established in Table 8 and carries no inferential value of its own. It is a supplementary summary and nothing in the Abstract or Conclusion is based on it.

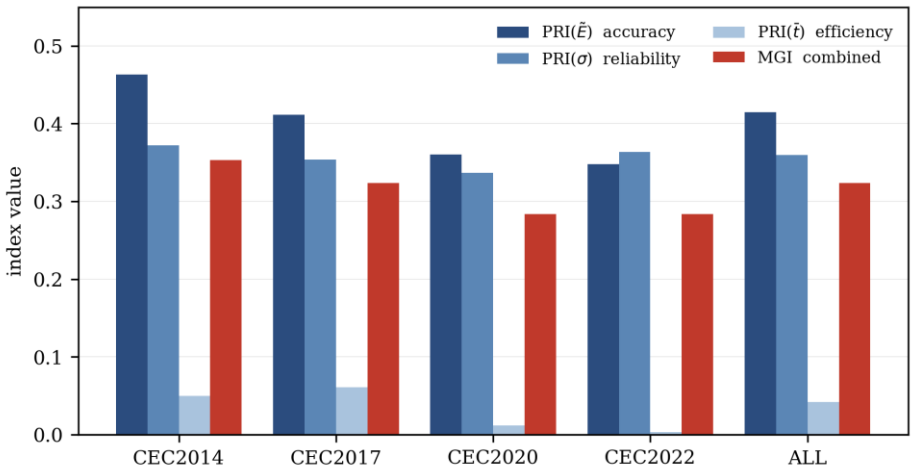


Figure 4. Paired relative improvement of ECPO over CPO, decomposed into accuracy, reliability, and efficiency components, together with the combined memory gain index, for each suite and for the pooled set

4.8. Computational Cost

The memory apparatus adds arithmetic to every trial, so ECPO should be marginally slower per evaluation. Table 5 shows the measured effect running the other way.

Pooled over the 81 functions, ECPO requires 9. 867 s per run compared to 10.410 for CPO (5.2% reduction) and is faster on 44 of them. The explanation is in the stopping rule of Section 3.3.2: a run terminates when the incumbent reaches the known optimum. An algorithm that reaches that point more often and more quickly will of course use fewer function evaluations. The bookkeeping cost of Equations 10 to 14 is real but is less than the difference observed here whatever the cause. The pattern across the suites is consistent with this: the reduction is largest on CEC 2017 (−6. 6.9%) and CEC 2014 (−6. 6.3%) the suites with the most exactly solvable functions and is negligible on CEC 2020 (−1.3%) and CEC 2022 (−1.1%). The reason for the split is given in Section 3.3.2: the stopping test can fire on every function in CEC 2014 and CEC 2017, and on CEC 2020 F1, but on none of the other 21 functions, where every run of both algorithms consumes the full budget. For those 21 functions, the two timings effectively measure the cost of the same number of evaluations (to within the 131 of Section 3.2.5), and they differ by about 1%. The correct conclusion is that the modification imposes no practical overhead, not that ECPO is intrinsically faster. On a problem with no known optimum where the budget is always exhausted the small additional arithmetic would be paid in full. Two further caveats are required. The number of evaluations consumed per run was not logged in these experiments so the time per evaluation cannot be reported directly and the 5.2% figure should not be interpreted as a saving per evaluation. The two programs also did not have the same cost per evaluation in any case, as for most of the run the baseline had 80 active individuals compared to 127 for ECPO (Section 3.3.2), which reduces its per-sweep bookkeeping. The comparison in Table 10 is therefore between two programs that differ in more than just memory. What can be said with certainty is the arithmetic that the memory itself adds: one comparison per evaluation (Equation 10), one uniform draw on about one trial in ten (Equation 12) and three sorts of at most 130 values per run (Equation 14) all of which are negligible in comparison to the cost of evaluating a CEC function.

4.9. Reproducibility Check Against the Published Baseline

Section 3.4.3 argued that the statistical baseline must be a matched re-execution rather than the values printed in the source publication. This subsection tests that argument on our own data, using CEC2017, for which per-function means of CPO under the same 30-run protocol are available in Abdel-Basset et al. study [4].

Our re-implementation of CPO yields a mean error of 92.04, compared with the published 82.20, and is worse on 18 of the 28 functions (signed-rank p = 8.38E-04). The difference is systematic rather than random as the Spearman rank correlation between the two sets of per-function errors is 0.94, so we are clearly looking at the same algorithm on the same landscapes, just one that is uniformly worse. Most of the difference is accounted for by the baseline settings, which we established only as we were preparing this revision and the report (Section 3.3.2). It was run with $T_f = 0.2$ so it was applying the physical attack operator on 40% of its trials rather than the 10% implied by the $T_f = 0.8$ of Table 1 and with a schedule that reduces its active population to 80. Both of these settings cause a single-leader algorithm to converge more quickly and explore less, which is not the behavior the published CPO is reported to exhibit. Both are departures from the reference implementation rather than characteristics of CPO. The other possible contributing factors are also listed: the uniform resampling of out of bound coordinates as per Equation 8 as in the reference code; the use of MATLAB's built in random number generator in both

programs; a budget of 10^6 evaluations with termination at the value 100q; and an evaluation counter that is advanced once per evaluation rather than once per generation as in the reference code. The random number stream itself and the floating point accumulation could also be responsible. Because the published results are means, a run-level comparison is not possible, and independent verification of the reference implementation under the current protocol is listed as a limitation (Section 5.3).

We report the gap because a reader who re-implements CPO from its description will see it too. The consequence for the assessment of ECPO is the central point. Against the published CPO values, ECPO achieves 82.74 versus 82.20, with a signed-rank p of 0.865 (statistically indistinguishable), and a naive reader would conclude that the modification has achieved nothing. Against our in-house baseline, the same ECPO runs yield p = 9.82E-05, with 18 wins and 0 losses. The two conclusions are in fact opposite and are derived from the same ECPO numbers. The paper claims that ECPO outperformed this particular baseline under this protocol. It does not claim that ECPO outperforms the CPO of Abdel-Basset et al. [4] as published, and, for the reasons given in Section 3.3.2, it does not claim that individual memory is the cause of the difference. Nor, in the light of Section 3.3.2, does it support the reading offered in the submitted version that the modification recovers the performance of the published baseline from an implementation that is below it. The plain reading is that ECPO was run with the settings of the reference implementation and the baseline was not and that is essentially what Table 11 records. We report the table because it is the clearest evidence we have that the comparison in Section 4 is not yet controlled, and the two p-values in the columns do make the point of Section 3.4.3 regardless of the cause of the gap.

5. Discussion and Conclusion

5.1. What the Measured Difference Looks Like

The paper's finding is very narrow in scope and narrower than the claim made in the submitted version. ECPO beat the CPO baseline on all 81 functions and across all four of the independently designed suites at D=10. The difference is highly significant in the pooled analysis (p = 1.49E-09), significant in each suite individually, never reversed on any individual function and 22 of the per-function improvements survive Holm's correction. What the experiments do not show is the cause. The baseline differs from ECPO in three control parameters and in its population schedule and it already selects its survivors by fitness (Section 3.3.2), so the memory of Equations 12 and 13 is one of four possible explanations and the two modifications cannot be credited with the result. The following paragraphs describe the nature of the difference which is in itself informative and the question of the cause is left open. The nature of the difference is more interesting than its size. Roughly half of the difference is a reduction in dispersion and half is a shift in the mean. A difference in survivor selection would account for the first of these as selecting on quality removes a source of randomness that selecting by array position leaves in; so would a larger active population for the same reason that larger samples vary less. The submitted version ascribed the dispersion effect to Equation 14 which the comparison does not support as the baseline itself selects on quality. On functions where the baseline outcome distribution is dispersed ECPO improves 31 of 34 and reduces the dispersion on 31 of 34, against 23 of 47 and 21 of 47 on the remainder and the mean PRI differs by a factor of three between the two groups (Table 7). Individual cases make the same point more clearly. On CEC2017 F28 ECPO doesn't find a better solution than CPO's best at all; it finds CPO's best in every run and its standard deviation drops from 154.89 to zero. The same pattern is visible descriptively on CEC2022 F11 and CEC2020 F6, although in neither case is the difference significant on its own (p = 0.161 and p = 0.102). In each of these cases the baseline solves the function in the majority of runs and fails badly in a small minority of runs, the signature of an outcome determined by something other than the quality of the search and the effect of whatever is responsible for the difference is simply that the failures stop. A difference in attractor diversity (whatever the source) would manifest itself differently: not as fewer failures but as better basins, either as a shift in the whole distribution or as an improvement in the best run. The best run of ECPO on CEC2014 F30 (3200.00), as opposed to the best run of CPO (3462.69), is of this type, as is the shift in mean on CEC2017 F24 (and descriptively on F10). The breakdown by function class in Table 6 is consistent with this description: for the unimodal functions a single attractor is correct and the colony should converge on it whereas for the hybrid functions the block structure means that the individuals have different things to remember. Both types of difference are present in the data, and neither can be attributed to a mechanism without the matched comparison and the ablation in Section 5.3. One observation does stand out and is worth making. Whatever is responsible for the difference, it is a very small one: the memory attractor touches 2% of all trials, the active populations differ by about a third, and the attack rate differs by a factor of four in about a tenth to a half of the trials. That differences of such a small magnitude can separate two runs of the same algorithm at p = 1.49E-09 over 81 functions is itself worth recording, and it suggests that the decisions involved have a much greater impact than the fraction of the algorithm they occupy.

5.2. Where it Fails

CEC2014 F28 is well worth looking at rather than just dismissing. CPO escapes to a better basin in a minority of runs and achieves a mean of 2993.33, ECPO escapes in none of its 30 runs. The submitted paper explained that the baseline loses its anchor during a reduction and drifts into a better basin, which merit-based ECPO selection would prevent. That explanation is undermined by Section 3.3.2, as the baseline also selects on merit. Two of the other differences could account for the same outcome. The baseline applies the attack operator in 40% of its trials, as opposed to 10% for ECPO, so its colony is pulled onto the incumbent much more often. On a landscape where the incumbent is in a mediocre basin, a strong pull towards it is also a strong push through it. Its population drops to 80 whereas that of ECPO remains at 127 so it is sampling the same budget from fewer, more mobile points. Neither of those explanations is actually tested here and we merely offer them as possibilities. The case is one worth making as a general property rather than as a single, isolated result. It is a general property of any mechanism that reduces the variation between runs that it will tend to narrow the distribution from both ends: it will remove the most catastrophic runs (which is good) and the most fortunate ones (which is not). On 31 of the 34 high-dispersion functions, the trade is in favor of ECPO, as the number of bad accidents exceeds the number of good ones; on F28, the

landscape is such that it isn't. A basin that is worth keeping and a basin that is worth abandoning are not in any way distinguishable by any signal internal to the algorithm without some further information. The natural solution is to apply selection pressure conditionally, only when there is evidence that the search has in fact stalled. ECPO computes no diversity or progress signal so that remains for future work.

The case does in fact illustrate a more general property of the selection pressure that Equation 14 represents relative to the CPO as published and it's worth noting this as it will be relevant when we come to the matched comparison in Section 5.3 Truncation is applied at every reduction event and always at the same end of the ranking. An individual that has strayed into a bad region on its way to a good basin is, at that point, one of the worst, and it is removed by truncation before it even gets there. For most functions, such individuals are just bad; on a landscape like F28, they may be the only way out of a mediocre basin. Three possible solutions to this follow, none of which we have actually tried: a stochastic survivor rule (e.g. a binary tournament) which still favours the good individuals but gives a weak one a chance to survive with some probability; a short period of protection for individuals that have just improved (so that a trajectory isn't cut off in the middle); and the stagnation conditional rule described above which would only apply truncation once the colony has stopped improving.

5.3. Limitations

So the experiments measure the combined effect of the four changes and can't separate the memory from the other three. The experiment that would allow that is a re-execution of the baseline with the parameter values and the schedule of Table 1, nothing else. It requires no new code beyond the parameter values and the parenthesisation of Equation 2 and costs the same as the runs already reported. It wasn't possible within the revision period and until it is done no causal claim can be made about the individual memory. Every statement in Sections 4 and 5.1 has been written with that in mind. A second limitation of the same kind is the absence of an ablation. The two modifications were introduced together and tested together. Sections 4.5 and 5.1 describe both differences in the data, but even in a matched comparison, we can't say what the relative contribution of Equations 13 and 14 is. The missing experiment is trivial to specify and we state it so that it can be run: four variants under the protocol of Section 3.3.2, namely CPO; CPO with the memory attractor only ($\pi = 0.2$, survivors chosen by array position as in CPO); CPO with memory-informed survivor selection only ($\pi = 0$, Equation 14 applied); and ECPO with both. Setting π to zero exactly recovers the attractor behaviour of CPO and reverting Equation 14 exactly recovers the reduction behaviour of CPO as published (which as Section 3.3.2 notes is not what the baseline used here does) so that arm has to be written before the ablation can be run. The ECPO runs reported here provide one of the four arms; the baseline runs reported here provide none of them for the reason already given. The comparison would tell us whether one modification dominates, whether the two interact or whether each one contributes on its own. Until both that experiment and the matched re-execution are done no part of the difference reported in Section 4 can be attributed to either modification. All experiments have $D = 10$. The two modifications are not equally sensitive to dimension. Equation 14 is dimension-free and should behave the same way. In contrast, the value of an individual memory as an attractor depends on how well a single remembered point represents a basin, which degrades as the volume increases. This study provides no evidence regarding behavior at $D = 30$, 50, or 100, where the effect may be weaker; all conclusions in this paper are restricted to $D = 10$. As π was fixed a priori at 0.2 and didn't vary then this study provides no information on its sensitivity. An interior optimum in π is certainly plausible at the value that best balances information transfer between the colony best and the individual memory but its location and its dependence on the problem have not been explored; a sensitivity study over $\pi \in \{0, 0.1, 0.2, 0.4, 0.6, 0.8, 1\}$ on the same suites is the obvious next step. Two further properties of the implementations restrict the results. The schedule realised by Equation 2 in ECPO keeps between 127 and 130 individuals active so Equation 14 has been run with three individuals removed over the course of a run and its behaviour under a schedule that reaches the nominal floor of 120 (or the 80 of the baseline) is untested; the same schedule also makes ECPO's colony monotonic rather than cyclic which is a departure from CPO that we didn't intend and which we haven't evaluated. And as u_2 is the zero vector in the reference code the perturbation terms in Equations 5 to 7 and 13 were inactive in every run; whether the conclusions hold for an implementation with a genuine random mask has not been examined. As the individual run values were not retained the per-function significance is based on the Welch approximation of Equation 20 rather than on a rank test of the raw runs (a rank sum test was in fact computed at run time in Section 3.4.4 but neither the p-values nor the runs behind them were saved and the normality assumption underpinning that approximation is not satisfied by the final fitness distributions). The conclusions of the paper do not depend on it and would be unchanged if the per-function symbols were removed (though they should be read as indicative only). Holm's correction and the confidence intervals in Supplementary Table S1 are based on the same summary statistics, and the raw values from the 30 runs cannot be released for the same reason; future experiments will retain them. The number of evaluations consumed per run was not logged either so runtime differences cannot be split into earlier termination and cost per evaluation (Section 4.8). Our baseline also falls short of the published CPO results on CEC2017 (Section 4.9); the settings account for much of that gap but an independent verification of the reference implementation under the protocol used here remains to be done. Finally, the stopping test compares the incumbent with 100q, which is the optimum on 60 of the 81 functions and below it on the other 21 (Section 3.3.2), so no run of either program terminated early on CEC2020 F2 to F10 or on CEC2022; that affects the runtimes in Table 10 and nothing else. The benchmark covers only shifted and rotated synthetic functions; no constrained or engineering design problems were included so no claims are made outside of that setting. Lastly, ECPO was compared only to its own baseline. That is the appropriate comparison for the question we are asking, namely, what the modification adds. It tells us nothing about absolute competitiveness. To place ECPO in relation to the leading entries in the CEC competitions (in particular L-SHADE [10] and LSHADE-SPACMA [31]) it is necessary to run them under the protocol used here as the published competition results were obtained under different budgets (e.g. 10^5 evaluations at D=10 on CEC2014 and CEC2017) and cannot be set beside ours. Until that is done the claims of this paper are limited to the comparison with the CPO baseline reported here and imply nothing about the standing of ECPO among other optimisers.

5.4. Conclusion

This paper didn't start with a metaphor; it started with an observation: for each individual, the algorithm keeps the best position that individual has ever been to and then uses only that information when it's time to roll back. Two decisions that are obviously dependent on the individual's history (the step taken by the exploitative attack operator and which individuals survive a population reduction) are made without any reference to that history. ECPO takes that quantity and makes it explicit memory, allowing it to act on precisely those two decisions while leaving all the other decisions untouched. The cost is one parameter, O(ND) storage and no function evaluations. What this paper reports is a measurement not a demonstration. The baseline against which ECPO was run differs from it in three control parameters and in its population schedule and selects survivors by quality as ECPO does (Section 3.3.2). The figures that follow are therefore a measure of the difference between two algorithms and do not demonstrate that the design was responsible for it. Under a budget of 10^6 evaluations and 30 independent runs, over the 81 functions of CEC2014, CEC2017, CEC2020 and CEC2022 at D=10, ECPO improved the mean on 54 functions, matched it on 22 and degraded it on 5. 33 of the improvements were significant at the 5% level (22 after Holm's correction), and there were no significant degradations, with a pooled signed-rank p of 1.49E-09. The mean error relative to the nominal optima was reduced by 11.5% (40.0% relative to the best value either algorithm ever reached, a secondary, experiment-dependent measure). The standard deviation across runs was reduced for 52 functions; the number of functions returning with zero dispersion increased from 21 to 30, and the mean runtime was reduced by 5.2% as more runs terminated early at the optimum. The difference between the two algorithms is concentrated in the areas where ECPO's design is expected to have an effect. However, again, for the reason stated above, that is not evidence of causality. The difference is absent for the unimodal functions, is largest for the hybrid functions, and is dominated by the disappearance of failed runs rather than by the refinement of successful ones. That last point is the practical one. An optimizer that returns a good answer in 29 out of 30 runs and a poor one in the one remaining run is not as useful as its average would suggest. A practitioner whose objective is an expensive simulation can afford to run just one of those 30. They have no way of knowing which of the two outcomes they have actually obtained. Reducing the number of functions on which such a gamble is made (i.e. the number of functions with non-zero dispersion) from 60 to 51 is more useful in practice than a proportional gain in average accuracy. Beyond the algorithm itself the paper makes a more general point about measurement which we believe is applicable in a much wider context. Absolute accuracy should be reported relative to the benchmark optima, with the error floor explicitly stated. Otherwise the only available reference is the implementation itself and that is not an independent reference. Significance should be assessed against a matched re-execution of the baseline because, in a paired test, the reference cancels out, but the baseline does not. Published values serve as a check on the implementation, not as the basis for a hypothesis test. Section 4.9 shows what happens when the two are conflated: the same ECPO results are Significant in one case and not in the other. Our own baseline wasn't matched either in the parameters or in the implementation (same error, just in a less obvious form), which is why we report it in full (Section 3.3.2). Against the published CPO results on CEC2017, ECPO is statistically indistinguishable (p=0.865), and we make no further claim. The limitations point very clearly to the work that must now be done. The first is a re-execution of the baseline with the parameters and schedule in Table 1, which would turn the measurement reported here into a proper controlled comparison; the second is the ablation of Equations 13 and 14, which would then identify the dominant mechanism. Both are trivial and cheap to do, and neither requires any new code. Fixing the failure on CEC2014 F28 would require a stagnation-conditional selection pressure that relaxes truncation selection when the colony stops improving, at the cost of a diversity signal the algorithm currently lacks. Applying the same two-point intervention to other recent swarm optimizers would then tell us whether the pattern we've seen here is a property of CPO or of the design pattern to which it belongs.

6. Statements & Declarations

6.1. Author Contributions

Conceptualization, J.G.; methodology, Z.A., H.A.K., and J.G.; software, Z.A. and H.A.K.; validation, Z.A., H.A.K., S.A.Z., and J.G.; formal analysis, Z.A. and H.A.K.; investigation, Z.A., H.A.K., and S.A.Z.; data curation, Z.A.; writing—original draft preparation, Z.A. and H.A.K.; writing—review and editing, S.A.Z. and J.G.; visualization, Z.A. and H.A.K.; supervision, J.G.; project administration, J.G. All authors have read and agreed to the published version of the manuscript.

6.2. Data Availability Statement

The MATLAB implementation of ECPO (the baseline program as it was run), the driver script, the settings as specified in Table 1, and the per-function results as reported in Tables 2 to 11 and Supplementary Table S1 will be made available in a public repository upon acceptance; in the interim, they are available from the corresponding author on request. We are depositing the baseline as it was run (all differences and all) so that the comparison made in Section 3.3.2 can be checked against it. The raw values from the individual runs are not being retained (Section 5.3). The CEC2014, CEC2017, CEC2020, and CEC2022 benchmark implementations are available in their respective technical reports, and the CPO reference implementation is available from the source cited by Abdel-Basset et al. [4].

6.3. Funding

This research received no external funding.

6.4. Acknowledgments

The authors would like to thank the organizers of the CEC special sessions for making the benchmark suites and their reference implementations freely available, and Abdel-Basset et al. for making the source code of the Crested Porcupine Optimizer (CPO) available, which was essential to constructing the baseline used in this work. We would like to thank the

Artificial Intelligence Research Laboratory (AIRL) at the University of Mazandaran for their support of this work. We are also very grateful to the anonymous reviewers and the journal editorial board for their very careful, thorough, and constructive reviews of the manuscript. The reviewers' comments and observations have been extremely helpful in improving the quality, clarity, and scientific rigor of the work. The authors would also like to acknowledge the use of AI-based tools (DeepSeek and Grammarly as per the Declaration of Generative AI below).

6.5. Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this manuscript, the author(s) used AI-assisted tools for language editing and for assistance with the statistical post-processing of the experimental results. After using these tools, the author(s) reviewed and edited the content as needed and took full responsibility for the content of the published article.

6.6. Ethics Approval

Not applicable.

6.7. Informed Consent

Not applicable.

6.8. Consent for Publication

The author(s) consent to the publication of this manuscript.

6.9. Conflicts of Interest

The author(s) declare no conflict of interest.

7. References

[1] S. Mirjalili and J. Song Dong, "Introduction to nature-inspired algorithms," in Nature-Inspired Optimizers: Theories, Literature Reviews and Applications, vol. 811, Studies in Computational Intelligence. Springe, Cham, Switzerland, 2020, doi:10.1007/978-3-030-12127-3.

[2] M. Crepinsek, S. H. Liu, and M. Mernik, “Exploration and exploitation in evolutionary algorithms: A survey,” ACM Comput. Surv., vol. 45, no. 3, pp. 1–33, 2013, doi:10.1145/2480741.2480752.

[3] D. H. Wolpert and W. G. Macready, “No free lunch theorems for optimization,” IEEE Trans. Evol. Comput., vol. 1, no. 1, pp. 67–82, 1997, doi:10.1109/4235.585893.

[4] M. Abdel-Basset, R. Mohamed, and M. Abouhawwash, “Crested Porcupine Optimizer: A new nature-inspired metaheuristic,” Knowledge-Based Syst., vol. 284, p. 111257, 2024, doi:10.1016/j.knosys.2023.111257.

[5] F. Glover, “Tabu Search—Part I,” ORSA Journal on Computing, vol. 1, no. 3, pp. 190–206, Aug. 1989, doi:10.1287/ijoc.1.3.190.

[6] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: Optimization by a colony of cooperating agents,” IEEE Trans. Syst. Man, Cybern. Part B Cybern., vol. 26, no. 1, pp. 29–41, 1996, doi:10.1109/3477.484436.

[7] J. Kennedy and R. Eberhart, “Particle swarm optimization,” Proceedings of ICNN'95 - International Conference on Neural Networks, vol. 4, pp. 1942–1948, doi:10.1109/icnn.1995.488968.

[8] R. Storn and K. Price, “Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces,” Journal of Global Optimization, vol. 11, no. 4, pp. 341–359, Dec. 1997, doi:10.1023/a:1008202821328.

[9] R. Tanabe and A. Fukunaga, “Success-history based parameter adaptation for Differential Evolution,” 2013 IEEE Congress on Evolutionary Computation, pp. 71–78, Jun. 2013, doi:10.1109/cec.2013.6557555.

[10] R. Tanabe and A. S. Fukunaga, “Improving the search performance of SHADE using linear population size reduction,” 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 1658–1665, Jul. 2014, doi:10.1109/cec.2014.6900380.

[11] Á. Fialho, L. da Costa, M. Schoenauer, and M. Sebag, “Analyzing bandit-based adaptive operator selection mechanisms,” Ann. Math. Artif. Intell., vol. 60, no. 1, pp. 25–64, 2010, doi:10.1007/s10472-010-9213-y.

[12] S. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey Wolf Optimizer,” Advances in Engineering Software, vol. 69, pp. 46–61, Mar. 2014, doi:10.1016/j.advengsoft.2013.12.007.

[13] S. Mirjalili and A. Lewis, “The Whale Optimization Algorithm,” Adv. Eng. Softw., vol. 95, pp. 51–67, 2016, doi:10.1016/j.advengsoft.2016.01.008.

[14] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, and S. M. Mirjalili, “Salp Swarm Algorithm: A bio-inspired optimizer for engineering design problems,” Adv. Eng. Softw., vol. 114, pp. 163–191, 2017, doi:10.1016/j.advengsoft.2017.07.002.

[15] Faramarzi, M. Heidarinejad, B. Stephens, and S. Mirjalili, “Equilibrium optimizer: A novel optimization algorithm,” Knowledge-Based Syst., vol. 191, p. 105190, 2020, doi:10.1016/j.knosys.2019.105190.

[16] I. Ahmadianfar, O. Bozorg-Haddad, and X. Chu, “Gradient-based optimizer: A new metaheuristic optimization algorithm,” Information Sciences, vol. 540, pp. 131–159, Nov. 2020, doi:10.1016/j.ins.2020.06.037.

[17] S. Li, H. Chen, M. Wang, A. A. Heidari, and S. Mirjalili, “Slime mould algorithm: A new method for stochastic optimization,” Futur. Gener. Comput. Syst., vol. 111, pp. 300–323, 2020, doi:10.1016/j.future.2020.03.055.

[18] Abdollahzadeh, F. S. Gharehchopogh, and S. Mirjalili, “African vultures optimization algorithm: A new nature-inspired metaheuristic algorithm for global optimization problems,” Comput. Ind. Eng., vol. 158, p. 107408, 2021, doi:10.1016/j.cie.2021.107408.

[19] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies.,” Evol. Comput., vol. 9, no. 2, pp. 159–195, 2001, doi:10.1162/106365601750190398.

[20] J. J. Liang, B. Y. Qu, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization," in "Technical Report," Nanyang Technological University, Singapore, 2013.

[21] N. H. Awad, M. Z. Ali, J. J. Liang, B. Y. Qu, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective real-parameter numerical optimization," in "Technical Report," Nanyang Technological University, Singapore, 2016.

[22] C. T. Yue, K. V. Price, P. N. Suganthan, J. J. Liang, M. Z. Ali, B. Y. Qu, N. H. Awad, and P. P. Biswas, “Problem definitions and evaluation criteria for the CEC 2020 special session and competition on single objective bound constrained numerical optimization,”

Tech. Rep. 201911, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China, and Nanyang Technological University, Singapore, Nov. 2019.
[23] A. Kumar, K. V. Price, A. W. Mohamed, A. A. Hadi, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2022 special session and competition on single objective bound constrained numerical optimization," in "Technical Report," Nanyang Technological University, Singapore, 2021.
[24] V. Beiranvand, W. Hare, and Y. Lucet, "Best practices for comparing optimization algorithms," Optim. Eng., vol. 18, no. 4, pp. 815–848, 2017, doi:10.1007/s11081-017-9366-1.
[25] F. Wilcoxon, "Individual Comparisons by Ranking Methods," Biometrics Bull., vol. 1, no. 6, p. 80, 1945, doi:10.2307/3001968.
[26] Demšar, "Statistical comparisons of classifiers over multiple data sets," J. Mach. Learn. Res., vol. 7, pp. 1–30, 2006.
[27] Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," Swarm Evol. Comput., vol. 1, no. 1, pp. 3–18, 2011, doi:10.1016/j.swevo.2011.02.002.
[28] L. WELCH, "The generalisation of student's problems when several different population variances are involved.," Biometrika, vol. 34, no. 1–2, pp. 28–35, 1947, doi:10.1093/biomet/34.1-2.28.
[29] V. Hedges, "Distribution Theory for Glass's Estimator of Effect size and Related Estimators," J. Educ. Stat., vol. 6, no. 2, pp. 107–128, 1981, doi:10.3102/10769986006002107.
[30] J. Cohen, Statistical Power Analysis for the Behavioral Sciences, 2nd ed. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, 1988.
[31] W. Mohamed, A. A. Hadi, A. M. Fattouh, and K. M. Jambi, "LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems," in 2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings, 2017, pp. 145–152. doi:10.1109/CEC.2017.7969307.