A modified EACOP implementation for Real-Parameter Single Objective Optimization Problems

Andrea Tangherloni
Department of Computing Sciences
Bocconi Institute for Data Science and Analytics
Bocconi University, Milan, Italy
andrea.tangherloni@unibocconi.it

Vasco Coelho
Department of Informatics, Systems and Communication
University of Milano-Bicocca, Milan, Italy
v.coelho@campus.unimib.it

Francesca M. Buffa
Department of Computing Sciences
Bocconi University, Milan, Italy
Bocconi Institute for Data Science and Analytics
francesca.buffa@unibocconi.it

Paolo Cazzaniga
Department of Human and Social Sciences
University of Bergamo, Bergamo, Italy
paolo.cazzaniga@unibg.it

2024 IEEE Congress on Evolutionary Computation (CEC) | 979-8-3503-0836-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/CEC60901.2024.10611920

***Abstract*—Evolutionary algorithms are effective techniques for optimizing non-linear and complex high-dimensional problems. However, most of them require a precise fine-tuning of their functioning settings to achieve satisfactory results. In this work, we propose a modified version of an evolutionary approach called the Evolutionary Algorithm for COmplex-process oPtimization (EACOP), designed to have a limited number of hyper-parameters. The base version of EACOP (bEACOP) combines different strategies, including the scatter search methodology, local searches, and a novel combination method based on path relinking to balance the exploration and exploitation phases. Our improved version (iEACOP) intensifies the exploration phase to escape from suboptimal search space areas where, on the contrary, bEACOP gets stuck. Our results show that iEACOP outperforms bEACOP on 27 out of 29 CEC 2017 test suite benchmark functions, exhibiting comparable performance against the three best algorithms of the CEC 2017 competition on single-objective bound-constrained real-parameter numerical optimization. The source code of bEACOP and iEACOP will be made publicly available on GitHub upon acceptance.**

***Index Terms*—Global Optimization, Evolutionary Algorithms, Local Search, CEC 2017 competition, Real-parameter single objective optimization**

I. Introduction

Optimization is a fundamental task in Computer Science with applications ranging from building engineering design [1] to biochemical models calibration [2] and network development [3]. In many real-case scenarios, derivative-based algorithms face limitations due to the absence of analytical information or specific characteristics of the fitness landscape, such as noise, multi-modality, non-convexity, non-separability, and non-differentiability, which can lead to convergence to a local optimum. Various meta-heuristics within the families of derivative-free optimization algorithms have been introduced to address these challenges. These meta-heuristics draw inspiration from nature's strategies for solving problems that involve the minimization or maximization of a function. Two notable families include Swarm Intelligence (SI) [4], which emulates the collective behavior of social organisms, and Evolutionary Computation (EC) [5], inspired by Darwinian evolution theories.

In SI, individuals within the swarm are simple independent agents that engage in interactions governed by fundamental cooperation and/or competition rules. The resulting emergent behavior of the swarm offers an effective solution for navigating the search space of candidate solutions. Meta-heuristics such as Particle Swarm Optimization (PSO) [6], Artificial Bee Colony (ABC) [7], and Ant Colony Optimization [8] belong to the SI family. In the context of EC, the most promising individuals in a population, representing potential solutions to the optimization problem, undergo evolution through genetic operators that simulate natural selection. This iterative process facilitates the exploration of the search space over successive generations. Meta-heuristics such as Genetic Algorithms [9], Genetic Programming [10], and Evolution Strategies [11] belong to this family.

In this paper, we start with an evolutionary algorithm developed explicitly for the parameter estimation problem [12], and we present a modified implementation that allows for improving the overall optimization performance. This meta-heuristic, called Evolutionary Algorithm for COmplex-process oPtimization (EACOP) [13], [14], while incorporating some elements of scatter search, introduced a set of changes to the classic scatter search design to make the algorithm more robust and efficient. The goal is to obtain a better balance between diversification and intensification, which is crucial for global optimization algorithms, all while using fewer tuning parameters. The initial population is generated by sampling

solutions using the Latin hypercube, an empirically proven effective strategy, especially for large-scale optimization problems. Anyhow, as required by the competition rules, we used a uniform initialization strategy here. In what follows, we will refer to the base version of EACOP as bEACOP and to our improved version as iEACOP.

II. Related Works

EACOP was presented in [13] to address the optimization of complex-process models. The algorithm follows the standard scatter search design by generating an initial set of candidate solutions that is 10 times larger than the problem size. Specifically, the Latin hypercube uniform sampling is used to generate these solutions. All solutions are then evaluated, and the $Pop/2$ best ones in terms of fitness function (with Pop being the population size) are selected as members of the initial population. The remaining $Pop/2$ solutions are randomly selected from the other solutions. This simple strategy avoids the computational efforts otherwise required to calculate relative distances among solutions and select the most diverse.

The iterative process consists of applying a combination method to each pair of solutions. In particular, such a combination method defines hyper-rectangles around the solutions, which increases the number of search directions and enhances diversification, not only regarding search directions but also regarding search distance. Since the areas containing high-quality solutions should be explored more deeply than other areas, the relative quality of every pair of solutions (regarding their position in the sorted population) is used as a measure of bias to create hyper-rectangles. So, individuals with poor (good) fitness values will generate new solutions close to (far from) individuals with good (poor) fitness values with higher probability. Using a wide hyper-rectangle, it is unlikely to have combinations among individuals previously combined, avoiding the necessity of implementing memory structures. The fact that the size of the hyper-rectangles increases if the new solutions improve the old ones during consecutive iterations induces a diversification strategy, exploring regions where different minima can be found.

The population is updated throughout the generation following a $(1+1)$ strategy applied to every individual, similar to that used in other evolutionary algorithms [15]. To be more precise, each individual is combined with the rest of the individuals in the population, performing $Pop - 1$ combinations and creating $Pop - 1$ offspring. If the best individual outperforms its parent (i.e., the individual that was being combined), the former replaces the latter in the population. With this updating strategy, a solution can only enter the population by replacing its parent, enhancing the search intensification. Moreover, every solution follows a self-tuned annealing scheme, allowing significant steps at the beginning of the search while moving more locally towards the end due to the proximity of the individuals in the final stages.

bEACOP also uses a local search method to accelerate convergence speed. Specifically, dynamic hill climbing is implemented as it is characterized by a reduced computation time (with respect to other strategies), even if it does not ensure local convergence.

III. Methods

In this paper, following the work presented in [13], [16], [17], we propose a modified implementation of the bEACOP algorithm (named iEACOP) to improve its exploration capabilities. In particular, iEACOP limits the number of pointless local search calls, which often use large portions of the fitness evaluation budget. Our implementation is described in the following seven pseudocode listings. It is worth mentioning that we highlighted the modified parts with respect to the original pseudocode in blue.

The first listing (Algorithm 1) reports iEACOP's main procedure. The algorithm begins by initializing N individuals (line 2), where N can be either selected by the user or calculated using the heuristic shown in Algorithm 2 (line 1). In particular, $M = 10D$ individuals are *uniformly* generated within the search space (Algorithm 2, lines 15-16). Then, this initial population is sorted by quality (i.e., the lower the fitness value of an individual $\mathbf{x}$, the lower the position/index of $\mathbf{x}$ in the population, Algorithm 2, line 18). The first $N/2$ individuals of the M sorted individuals are selected, while the remaining $N/2$ individuals are randomly selected with replacement among the other $M - N/2$ sorted individuals (Algorithm 2, lines 19-25). Finally, the best individual of the population (i.e., the one with the lowest index) is assigned to $\mathbf{x}_{best}$, a placeholder for the best individual found so far during the optimization (Algorithm 2, line 27).

After that, iEACOP repeats the procedure reported in lines 4-30 of Algorithm 1 until a termination criterion is met (e.g., the maximum number of iterations or fitness evaluation is reached). First, it checks the individuals composing the current population to evaluate if there are individuals that are too similar to each other (Algorithm 1, line 5). In particular, for each pair of individuals $\mathbf{x}_i$ and $\mathbf{x}_j$, it calculates the relative difference between $\mathbf{x}_i$ and $\mathbf{x}_j$ for each dimension d, with $d = 1, \ldots, D$. If the maximum distance is lower than a user-defined threshold ϵ, an individual $\mathbf{x}$ is randomly generated within the search space and replaces $\mathbf{x}_j$ in the population (Algorithm 3, lines 6-9). Second, it creates an offspring population starting from the current population (Algorithm 1, line 6), as shown in Algorithm 4. For each pair of individuals $\mathbf{x}_i$ and $\mathbf{x}_j$, a new individual $\mathbf{x}$ is randomly generated within the hyper-rectangle defined by the two parents (Algorithm 4, lines 6-19). Third, iEACOP updates the current population by taking advantage of the generated offspring (Algorithm 1, line 7). To do so, it exploits possible promising directions using the *go-beyond* strategy (Algorithm 5, lines 1-22). For each individual $\mathbf{x}_i$ in the population, all its offspring are sorted by quality (Algorithm 5, line 26); if the best of them outperforms its parent $\mathbf{x}_i$ (i.e., it has a lower fitness value, line 29), a new non-convex solution in the direction defined by the child and its parent is randomly created (this is the core of *go-beyond* strategy, lines 7-17). The child becomes the new parent, and the newly generated

solution becomes the new child. If the improvement continues, we are probably in a promising area; thus, we apply this strategy again by doubling the area where we can randomly create new solutions. The *go-beyond* strategy ends when the fitness of the newly generated solution is greater than that of its parent. On the contrary, if an individual $\mathbf{x}_i$ in the population does not produce any new solution outperforming its fitness value for n_{change} consecutive iterations, we consider $\mathbf{x}_i$ stuck into a local optimum. Thus, $\mathbf{x}_i$ is replaced by another solution randomly generated within the search space (Algorithm 5, lines 33-43). Then, the updated population is sorted by quality (Algorithm 5, line 45). If $\mathbf{x}_1$ (i.e., the best individual of the population) is better than $\mathbf{x}_{best}$, we assign $\mathbf{x}_1$ to $\mathbf{x}_{best}$ (Algorithm 5, lines 46-47).

Finally, iEACOP applies the local search routine (Algorithm 1, lines 8-27). In bEACOP, the local search is performed around $\mathbf{x}_{best}$ if it has changed during the last generation. Otherwise, it applies another strategy where the first local search is still applied to $\mathbf{x}_{best}$, while the following ones to an offspring solution, which is selected based on a balancing strategy that takes into account their quality (i.e., fitness value) and diversity (i.e., the distance between the offspring and the old solutions found by the local search routines). All the details are reported in Algorithm 6 (lines 6-24). If the calculated local solution is better than $\mathbf{x}_{best}$, this solution is assigned to $\mathbf{x}_{best}$. We improved the original local search routine by introducing the following modifications. If $\mathbf{x}_{best}$ has changed during the last generation, iEACOP applies a local search around it, followed by a local search around an offspring solution (selected as described before). Otherwise, similar to the standard version, it performs the first local search to $\mathbf{x}_{best}$, while the following ones to selected offspring solutions. However, iEACOP does not execute local searches in all the generations to increase its exploration capabilities compared to bEACOP. Indeed, the local searches around the offspring solutions are performed only every a certain number of generations (i.e., n_2 = 10) when allowed, as our version deactivates the local search routine as follows. If a local search is applied and does not find a solution better than $\mathbf{x}_{best}$, the local search routine is deactivated for a certain number of iterations (between $n_{changes}$ and $2 \cdot n_{changes}$), based on the identification of novel $\mathbf{x}_{best}$ solutions through the generations (see Algorithm 1, lines 25-28; Algorithm 5, lines 46-55). In addition, since the selected local optimizer might strongly affect the results of the local search calls, iEACOP switches between "Powell" [18] and "L-BFGS-B" [19], [20]. Thanks to this switching approach, iEACOP exploits the peculiarities of both local optimization algorithms, allowing it to perform better local searches and deal with possible ill-conditioned problems. In particular, if applying the local search using "Powell" ("L-BFGS-B") to a selected solution (either $\mathbf{x}_{best}$ or an offspring) does not allow for finding a solution better than $\mathbf{x}_{best}$, we repeat the local search on the same initial solution using "L-BFGS-B" ("Powell") (Algorithm 7, lines 28-53). Finally, contrary to bEACOP, iEACOP replaces the worst individual of the population with $\mathbf{x}_{best}$ when it is identified by a local search routine (Algorithm 7, lines 16).

IV. Results

Our novel implementation of EACOP (i.e., iEACOP) was tested and evaluated using the benchmark functions from the CEC 2017 competition on single-objective bound-constrained real-parameter numerical optimization. The function suite, listed in Table I, encompasses a variety of optimization challenges, including shifted, rotated, not-separable, highly ill-conditioned, and complex problems [21].

TABLE I
CEC 2017 Benchmark Functions for the competition on single-objective bound-constrained real-parameter numerical optimization.

Typology	No.	Function name	Optimum
Unimodal functions	1	Shifted and rotated Bent Cigar	100
	2	Shifted and rotated Zakharov	300
Simple multimodal functions	3	Shifted and rotated Rosenbrock	400
	4	Shifted and rotated Rastrigin	500
	5	Shifted and rotated Expanded Schaffer F6	600
	6	Shifted and rotated Lunacek Bi-Rastrigin	700
	7	Shifted and rotated Non-Continuous Rastrigin	800
	8	Shifted and rotated Levy	900
	9	Shifted and rotated Schwefel	1000
Hybrid functions	10	Zakharov; Rosenbrock; Rastrigin	1100
	11	High-conditioned Elliptic; Modified Schwefel; Bent Cigar	1200
	12	Bent Cigar; Rosenbrock; Lunacek bi-Rastrigin	1300
	13	High-conditioned Elliptic; Ackley; Schaffer F7; Rastrigin	1400
	14	Bent Cigar; HGBat; Rastrigin; Rosenbrock	1500
	15	Expanded Schaffer F6; HGBat; Rosenbrock; Modified Schwefel	1600
	16	Katsuura; Ackley; Expanded Griewank plus Rosenbrock; Schwefel; Rastrigin	1700
	17	High-conditioned Elliptic; Ackley; Rastrigin; HGBat; Discus	1800
	18	Bent Cigar; Rastrigin; Griewank plus Rosenbrock; Weierstrass; Expanded Schaffer F6	1900
	19	HappyCat; Katsuura; Ackley; Rastrigin; Modified Schwefel; Schaffer F7	2000
Composition functions	20	Rosenbrock; High-conditioned Elliptic; Rastrigin	2100
	21	Rastrigin; Griewank; Modified Schwefel	2200
	22	Rosenbrock; Ackley; Modified Schwefel; Rastrigin	2300
	23	Ackley; High-conditioned Elliptic; Griewank; Rastrigin	2400
	24	Rastrigin; HappyCat; Ackley; Discus; Rosenbrock	2500
	25	Expanded Schaffer F6; Modified Schwefel; Griewank; Rosenbrock; Rastrigin	2600
	26	HGBat; Rastrigin; Modified Schwefel; Bent Cigar; High-conditioned Elliptic; Expanded Schaffer F6	2700
	27	Ackley; Griewank; Discus; Rosenbrock; HappyCat; Expanded Schaffer F6	2800
	28	F15; F16; F17	2900
	29	F15; F18; F19	3000

We ran all tests with the following configuration, as explicitly requested for the competition:

- problem dimensions $D = 30$;
- 25 independent runs for each benchmark function;
- search space boundaries $[-100, 100]^D$;
- uniform in $[-100, 100]^D$;
- uniform random initialization within the search space;
- maximum number of fitness evaluations $Max_{FEs} = 10000D$.

As a first set of tests, we compared the standard implementation of bEACOP with our improved version iEACOP described in Section III. Table II reports the mean and standard deviation

of the best individual obtained after the last fitness evaluation over 25 runs with our implementation of iEACOP (left column) and bEACOP (right column). We applied a Wilcoxon signed-rank test to evaluate whether iEACOP's results are statistically significantly better than bEACOP's results. It is worth specifying that both algorithms are initialized with the same population to properly run the statistical test. In particular, we applied the two-tailed test with a significance level equal to 0.05; if we reject the null hypothesis (i.e., two related paired samples come from the same distribution), then we use the left-tailed test. If we can reject this null hypothesis, then the distribution underlying iEACOP's results is stochastically less than the distribution underlying bEACOP's results. Otherwise, we use the right-tailed test to check if it is true that the distribution underlying iEACOP's results is stochastically greater than the distribution underlying bEACOP's results. The $+$ $(-)$ sign in Table II denotes a statistically significant difference between the two algorithms, i.e., iEACOP is better (worse) than bEACOP. In contrast, the $=$ sign indicates there is no statistical difference between them. iEACOP achieves better results in 27 out of 29 benchmark functions, while their performance is comparable in the other 2 cases (i.e., no statistical difference is observed).

TABLE II
STATISTICAL COMPARISON OF THE RESULTS OBTAINED BY IEACOP AND BEACOP. THE $+$ $(-)$ SIGN DENOTES A STATISTICALLY SIGNIFICANT DIFFERENCE BETWEEN THEM, WHILE THE $=$ SIGN INDICATES THERE IS NO STATISTICAL DIFFERENCE BETWEEN THEM.

	iEACOP	bEACOP	
F1	2.8524e-06 (8.1708e-07)	1.1275e+03 (1.3218e+03)	+
F2	1.1997e-06 (1.6844e-06)	1.3184e+00 (1.2415e+00)	+
F3	1.9136e+00 (1.9917e+00)	7.6395e+00 (1.8049e+01)	+
F4	2.9889e+01 (7.3086e+00)	4.7882e+01 (1.1837e+01)	+
F5	1.3205e-01 (2.0505e-01)	1.6761e-01 (3.2448e-01)	=
F6	5.3905e+01 (5.7495e+00)	7.1958e+01 (1.3957e+01)	+
F7	2.7522e+01 (6.326e+00)	4.3235e+01 (1.0430e+01)	+
F8	2.3809e+00 (9.732e+00)	3.2766e+01 (2.3683e+01)	+
F9	2.6376e+03 (4.6672e+02)	2.9548e+03 (3.0156e+02)	+
F10	3.1643e+01 (1.3364e+01)	4.9513e+01 (2.0493e+01)	+
F11	1.4991e+03 (4.7642e+02)	2.7108e+04 (4.9077e+04)	+
F12	1.5835e+02 (1.9149e+02)	5.2353e+03 (7.2041e+03)	+
F13	1.0156e+02 (2.7465e+01)	3.5200e+04 (4.1366e+04)	+
F14	7.8638e+01 (8.3057e+01)	1.6779e+03 (1.615e+03)	+
F15	3.7520e+02 (1.4856e+02)	4.7685e+02 (1.7156e+02)	+
F16	1.0751e+02 (6.7040e+01)	1.2436e+02 (7.4085e+01)	+
F17	2.2804e+02 (5.6898e+01)	2.2813e+05 (2.8967e+05)	+
F18	7.7244e+01 (7.2363e+01)	2.8494e+03 (4.2253e+03)	+
F19	1.8376e+02 (3.1189e+01)	1.9536e+02 (4.7942e+01)	=
F20	2.2399e+02 (6.8492e+00)	2.4511e+02 (1.2264e+01)	+
F21	1.0000e+02 (1.5605e-08)	1.0000e+02 (8.2336e-06)	+
F22	3.7734e+02 (1.1644e+01)	4.0908e+02 (1.5156e+01)	+
F23	4.4162e+02 (3.8874e+00)	4.6812e+02 (1.4085e+01)	+
F24	3.8563e+02 (1.6451e+00)	3.8616e+02 (1.9188e+00)	+
F25	1.0564e+03 (4.7261e+02)	1.2981e+03 (6.7282e+02)	+
F26	5.1217e+02 (6.7655e+00)	5.1652e+02 (7.5282e+00)	+
F27	3.1855e+02 (4.3402e+01)	3.1649e+02 (3.7788e+01)	+
F28	5.1931e+02 (4.8598e+01)	5.4802e+02 (5.8227e+01)	+
F29	3.2233e+03 (5.4382e+02)	3.7161e+03 (1.3628e+03)	+

Table III reports a complete summary of iEACOP's results, including the best and worst results, along with the mean, median, and standard deviation calculated over the 25 runs.

To assess the potential and limitations of iEACOP, we compared its performance against the state-of-the-art competitors that won the competition on the benchmark suite used here:

TABLE III
SUMMARY OF THE STATISTICS OF IEACOP'S RESULTS, OBTAINED OVER 25 INDEPENDENT RUNS, ON THE BENCHMARK PROBLEMS DEFINED IN $D = 30$ DIMENSIONS.

	Best	Worst	Median	Mean	Std
F1	3.5671e-07	3.7804e-06	3.0678e-06	2.8524e-06	8.1708e-07
F2	8.7519e-09	8.5058e-06	9.1044e-07	1.1997e-06	1.6844e-06
F3	4.1703e-08	3.9866e+00	2.1710e-07	1.9136e+00	1.9917e+00
F4	1.5919e+01	4.5768e+01	2.8854e+01	2.9889e+01	7.3086e+00
F5	2.7321e-03	7.2217e-01	2.2872e-02	1.3205e-01	2.0505e-01
F6	4.1738e+01	6.4838e+01	5.3609e+01	5.3905e+01	5.7495e+00
F7	1.4924e+01	3.9798e+01	2.5869e+01	2.7522e+01	6.326e+00
F8	3.1196e-10	4.9905e+01	8.9528e-02	2.3809e+00	9.732e+00
F9	1.2776e+03	3.4473e+03	2.7013e+03	2.6376e+03	4.6672e+02
F10	1.3936e+01	7.3633e+01	2.9856e+01	3.1643e+01	1.3364e+01
F11	6.9446e+02	2.8175e+03	1.3759e+03	1.4991e+03	4.7642e+02
F12	3.0854e+01	9.1755e+02	1.0473e+02	1.5835e+02	1.9149e+02
F13	5.2905e+01	1.5484e+02	1.0086e+02	1.0156e+02	2.7465e+01
F14	9.0117e+00	3.8715e+02	4.6242e+01	7.8638e+01	8.3057e+01
F15	1.2646e+02	6.9753e+02	3.7599e+02	3.7520e+02	1.4856e+02
F16	4.0037e+01	2.9626e+02	7.6147e+01	1.0751e+02	6.7040e+01
F17	8.3824e+01	3.1293e+02	2.462e+02	2.2804e+02	5.6898e+01
F18	1.7277e+01	3.4921e+02	4.5065e+01	7.7244e+01	7.2363e+01
F19	1.015e+02	2.4277e+02	1.8228e+02	1.8376e+02	3.1189e+01
F20	2.1294e+02	2.4509e+02	2.2295e+02	2.2399e+02	6.8492e+00
F21	1.0000e+02	1.0000e+02	1.0000e+02	1.0000e+02	1.5605e-08
F22	3.6149e+02	4.0850e+02	3.7664e+02	3.7734e+02	1.1644e+01
F23	4.3547e+02	4.4923e+02	4.4197e+02	4.4162e+02	3.8874e+00
F24	3.8340e+02	3.8699e+02	3.8680e+02	3.8563e+02	1.6451e+00
F25	2.0000e+02	1.5058e+03	1.2802e+03	1.0564e+03	4.7261e+02
F26	4.9697e+02	5.2433e+02	5.1159e+02	5.1217e+02	6.7655e+00
F27	3.0000e+02	4.5393e+02	3.0000e+02	3.1855e+02	4.3402e+01
F28	4.5833e+02	6.6293e+02	5.1433e+02	5.1931e+02	4.8598e+01
F29	2.4085e+03	4.7911e+03	3.2130e+03	3.2233e+03	5.4382e+02

Effective Butterfly Optimizer with Covariance Matrix Adapted Retreat Phase (EBOwithCMAR), which is a dual population-based bio-inspired optimization technique that uses a covariance matrix to generate new solutions possibly improving the local search capability of the optimizer [22]; jSO, which is a variant of the iL-SHADE algorithm that uses a weighted version of the mutation strategy to improve the quality of the solutions [23]; a variant of the L-SHADE algorithm called LSHADE-cnEpSin, which introduces a performance adaptation scheme based on an ensemble of sinusoidal approaches, and covariance matrix learning for the crossover operator [24]. In what follows, we graphically present the obtained results by showing the convergence plot, which considers the median obtained in the 25 runs, and the boxplot, which encompasses the information of the best solution achieved after the last fitness evaluation of each run. For clarity, we use the same color scheme in all figures; the boxplots follow the exact ordering of the labels in the legend.

Analyzing the performance of iEACOP with respect to that of its competitors, we observed that in some cases (i.e., functions 3, 24, and 27), iEACOP achieves better results, as can be clearly observed in Figure 1 that plots the results obtained with function F3. In other cases, the performance of the four meta-heuristics are comparable (i.e., functions 5, 23, and 25) as reported in Figure 2, which reports the results obtained with function F27.

A different situation is observed in several cases (i.e., functions 2, 8, 11, 12, 13, 14, 17, 18, and 29) where iEACOP performs worse than the other algorithms, as shown in Figure 3 where the convergence toward the optimal solution is slower

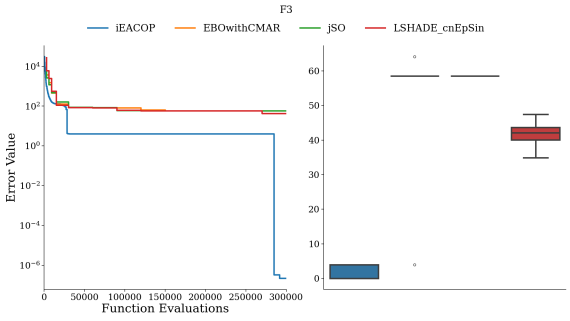

Fig. 1. Comparison between iEACOP, EBOwithCMAR, jSO, and LSHADE-cnEpSin for the optimization of the F3 benchmark functions. The convergence plot (left) is obtained by considering the median of the best individuals' error values found during the 25 runs. The boxplots (right) represent the distribution of the error values of the best individuals found by each algorithm over 25 runs.

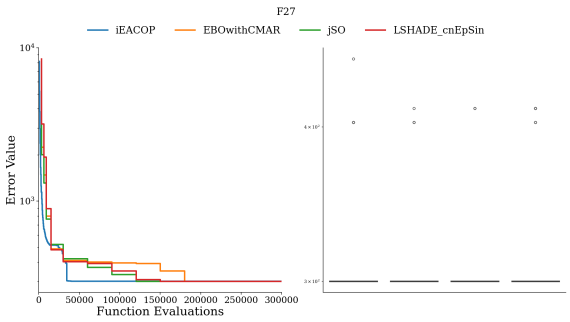

Fig. 2. Comparison between iEACOP, EBOwithCMAR, jSO, and LSHADE-cnEpSin for the optimization of the F27 benchmark functions. The convergence plot (left) is obtained by considering the median of the best individuals' error values found during the 25 runs. The boxplots (right) represent the distribution of the error values of the best individuals found by each algorithm over 25 runs.

and, in general, the final best results are significantly worse.

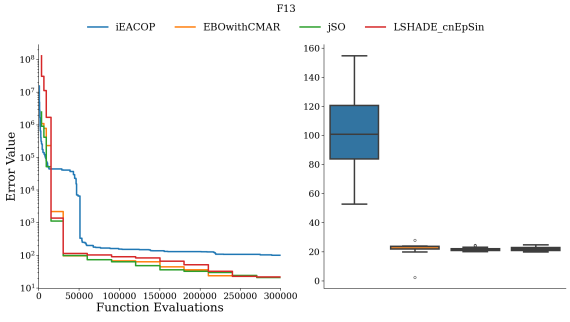

Fig. 3. Comparison between iEACOP, EBOwithCMAR, jSO, and LSHADE-cnEpSin for the optimization of the F13 benchmark functions. The convergence plot (left) is obtained by considering the median of the best individuals' error values found during the 25 runs. The boxplots (right) represent the distribution of the error values of the best individuals found by each algorithm over 25 runs.

In all other functions (the majority of the cases), the results present an interesting pattern in which, despite iEACOP's struggling to compete with the other algorithms, its convergence speed is way faster in the first part of the optimization process. This behavior can be observed, for instance, in Figure 4 (left), where iEACOP converges faster at the beginning of the optimization, achieving median error values clearly lower than its competitors, and gets stuck in a local minimum in the second part of the runs. The final results are eventually worse, as summarized in the boxplot of Figure 4 (right).

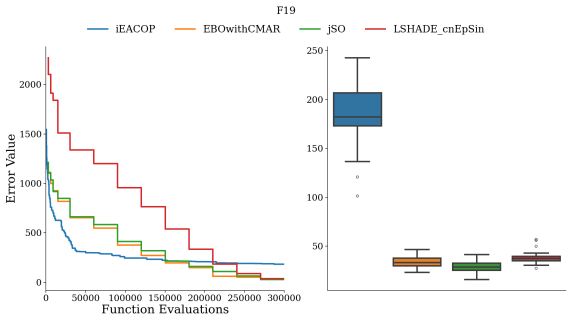

Fig. 4. Comparison between iEACOP, EBOwithCMAR, jSO, and LSHADE-cnEpSin for the optimization of the F19 benchmark functions. The convergence plot (left) is obtained by considering the median of the best individuals' error values found during the 25 runs. The boxplots (right) represent the distribution of the error values of the best individuals found by each algorithm over 25 runs.

It is worth mentioning that both bEACOP and iEACOP are written in `Python 3` programming language and only rely on NumPy [25] and SciPy [26] libraries. To evaluate the benchmark functions, we created bindings of the official C++ code using the Python `pybind11` library. The source code of bEACOP and iEACOP is available on GitHub at the following address: https://github.com/andrea-tango/iEACOP. We also calculated an empirical computational complexity of bEACOP and iEACOP, as described in the instructions of the CEC 2024 competition. Table IV reports 3 algorithm complexity indicators of bEACOP and iEACOP. T_1 indicates the computing time of 10000 evaluations, averaged over each problem. T_2 is the complete computing time of the algorithm with 10000 evaluations, averaged over each problem. To accommodate variations in computing time due to the adaptive nature of bEACOP and iEACOP, we considered the mean $\hat{T}_2$ over 5 independent calculations of T_2. The algorithm overhead is reflected by $T_3 = (\hat{T}_2 - T_1)/T_1$. This table indicates that iEACOP is slightly slower than bEACOP, probably due to the higher number of generations performed by iEACOP.

TABLE IV
ALGORITHM COMPLEXITY.

	T_1	$\hat{T}_2$	T_3
bECAOP	0.138	0.673	3.876
iECAOP	0.138	0.816	4.841

All tests were executed on a workstation equipped with two Intel® Xeon® Gold 6152 (22 cores each and clock 2.10 GHz) and 192 GB of RAM, running Ubuntu 22.04 LTS.

V. CONCLUSIONS

In this paper, we proposed an improved version of bEACOP [12]–[14], an evolutionary algorithm designed for the opti-

mization of complex-process models. Our version (iEACOP) improves bEACOP's exploration capabilities by limiting the number of pointless local search calls and avoiding wasting large portions of the fitness evaluation budget in non-promising areas of the search space. iEACOP showed better optimization capabilities than bEACOP in 27 out of 29 benchmark functions, highlighting the effectiveness of our modifications.

When compared to the state-of-the-art algorithms that won the competition on the CEC 2017 test suite benchmark functions, iEACOP showed comparable performance, sometimes obtaining better results (e.g., functions 3, 24, and 27). In other functions, despite iEACOP struggles to compete with the other algorithms at the end of the optimization, it exhibits a better convergence speed in the first part of the optimization process, achieving median error values lower than its competitors, showing that the strategies behind iEACOP are promising. We will further investigate this behavior to develop a new version of iEACOP that can escape local minima. We plan to apply the local searches based on the fitness evaluation budget, deactivating them when the algorithm struggles and is stuck on local optima. This will force it to explore other, possibly more promising areas of the search space. In addition, we will implement alternative strategies for generating new individuals encompassing hyper-spheres instead of hyper-rectangles.

These results clearly indicate that although bEACOP was originally designed to solve optimization problems in Systems Biology, our modifications make iEACOP a suitable algorithm for any real-valued problem. As a possible practical application of iEACOP, we will apply it to solve the Parameter Estimation of complex biochemical networks, which are often characterized by oscillatory behaviors that make it difficult to find good model parameters [27].

ACKNOWLEDGMENT

We acknowledge the CINECA award under the ISCRA initiative, for the availability of high-performance computing resources and support.

REFERENCES

[1] K. Deb, *Optimization for engineering design: Algorithms and examples*. PHI Learning Pvt. Ltd., 2012.

[2] M. S. Nobile, A. Tangherloni, L. Rundo, S. Spolaor, D. Besozzi, G. Mauri, and P. Cazzaniga, "Computational intelligence for parameter estimation of biochemical systems," in *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2018, pp. 1–8.

[3] M. Elbes, S. Alzubi, T. Kanan, A. Al-Fuqaha, and B. Hawashin, "A survey on particle swarm optimization with emphasis on engineering and network applications," *Evolutionary Intelligence*, vol. 12, no. 2, pp. 113–129, 2019.

[4] J. Kennedy and R. C. Eberhart, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[5] D. Simon, *Evolutionary Optimization Algorithms*. Wiley, 2013.

[6] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[7] D. Karaboga, "An idea based on honey bee swarm for numerical optimization," Technical report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department, Tech. Rep., 2005.

[8] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York: Addison-Wesley, 1989.

[10] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[11] H. Beyer, *The Theory of Evolution Strategies*. Springer, Berlin, Heidelberg, New York, 2001.

[12] D. R. Penas, P. González, J. A. Egea, R. Doallo, and J. R. Banga, "Parameter estimation in large-scale systems biology models: a parallel and self-adaptive cooperative strategy," *BMC bioinformatics*, vol. 18, pp. 1–24, 2017.

[13] J. A. Egea, R. Martí, and J. R. Banga, "An evolutionary method for complex-process optimization," *Computers & Operations Research*, vol. 37, no. 2, pp. 315–324, 2010.

[14] J. A. Egea, E. Balsa-Canto, M.-S. G. García, and J. R. Banga, "Dynamic optimization of nonlinear processes with an enhanced scatter search method," *Industrial & Engineering Chemistry Research*, vol. 48, no. 9, pp. 4388–4401, 2009.

[15] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, pp. 341–359, 1997.

[16] A. F. Villaverde, J. A. Egea, and J. R. Banga, "A cooperative strategy for parameter estimation in large scale systems biology models," *BMC systems biology*, vol. 6, no. 1, pp. 1–17, 2012.

[17] J. A. Egea, D. Henriques, T. Cokelaer, A. F. Villaverde, A. MacNamara, D.-P. Danciu, J. R. Banga, and J. Saez-Rodriguez, "Meigo: an open-source software suite based on metaheuristics for global optimization in systems biology and bioinformatics," *BMC bioinformatics*, vol. 15, no. 1, pp. 1–9, 2014.

[18] M. J. D. Powell, "An efficient method for finding the minimum of a function of several variables without calculating derivatives," *The Computer Journal*, vol. 7, no. 2, p. 155, 1964.

[19] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[20] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization," *ACM Trans. Math. Softw.*, vol. 23, p. 550–560, 1997.

[21] R. Cheng, M. Li, Y. Tian, X. Zhang, S. Yang, Y. Jin, and X. Yao, "Benchmark functions for cec'2017 competition on evolutionary many-objective optimization," in *Proc. IEEE Congr. Evol. Comput*, 2017, pp. 1–20.

[22] A. Kumar, R. K. Misra, and D. Singh, "Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase," in *2017 IEEE congress on evolutionary computation (CEC)*. IEEE, 2017, pp. 1835–1842.

[23] J. Brest, M. S. Maučec, and B. Bošković, "Single objective real-parameter optimization: Algorithm jSO," in *2017 IEEE congress on evolutionary computation (CEC)*. IEEE, 2017, pp. 1311–1318.

[24] N. H. Awad, M. Z. Ali, and P. N. Suganthan, "Ensemble sinusoidal differential covariance matrix adaptation with euclidean neighborhood for solving cec2017 benchmark problems," in *2017 IEEE congress on evolutionary computation (CEC)*. IEEE, 2017, pp. 372–379.

[25] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[26] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[27] A. Tangherloni, S. Spolaor, P. Cazzaniga, D. Besozzi, L. Rundo, G. Mauri, and M. S. Nobile, "Biochemical parameter estimation vs. benchmark functions: A comparative study of optimization performance and representation design," *Applied Soft Computing*, vol. 81, p. 105494, 2019.

Algorithm 1 Main procedure

$solutions$: data structure containing the candidate solutions at each generation.
$offspring$: data structure containing the offspring solutions at each generation.
$solutions_{local}$: data structure containing the solutions generated by the local search procedures over the generations.
$\mathbf{x}_{best}$: the best solution found so far.
$n_{change} = 20$, $\epsilon = 10^{-3}$, $n_1 = 1$, $n_2 = 10$, $balance = 0.5$, $opt = Powell$

procedure EACOP
  INITIALISE(N, D)
  $it \leftarrow 0$
  repeat
    CHECK_DIVERSITY()
    COMBINATION_METHOD()
    UPDATE_POPULATION()
    if $apply_{local}$ **then**
      if $last_{B_{local}} = 0$ **then**
        LOCAL_SEARCH$_1$()
        if $|solutions_{local}| > 1$ **then**
          LOCAL_SEARCH$_2$()
        end if
      else
        if $|solutions_{local}| = 0$ **then**
          if $n_{L_{eval}} \geq n_1$ **then**
            LOCAL_SEARCH$_1$()
          end if
        else if $n_{L_{eval}} \geq n_2$ & $it \bmod n_2 = 0$ **then**
          LOCAL_SEARCH$_2$()
        end if
      end if
      $n_{L_{eval}} \leftarrow 0$
    end if
    if $last_{R_{local}} \geq n_{change}$ **then**
      $apply_{local} \leftarrow True$
      $last_{R_{local}} \leftarrow 0$
    end if
    $it \leftarrow it + 1$
  until stopping criterion is met.
end procedure

Algorithm 2 Initialisation routine

function HEURISTIC(D)
  $N \leftarrow 1 + \sqrt{1 + 4D}$
  $N \leftarrow \lceil \frac{N}{2} \rceil$
  if $N \bmod 2 = 0$ **then**
    return N
  else
    return $N + 1$
  end if
end function

function INITIALISE(N, D)
  if $N = 0$ **then**
    $N \leftarrow$ HEURISTIC(D)
  end if
  $M \leftarrow 10D$
  $solutions_{initial} \leftarrow$ RANDOM_SAMPLING(M)
  EVALUATE($solutions_{initial}$)
  SORT($solutions_{initial}$)
  ▷ $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M\}$ so that $f(\mathbf{x}_i) \leq f(\mathbf{x}_j)$
  ▷ where $i, j \in 1, 2, \ldots, M$ with $i < j$
  for $i \leftarrow 1, \frac{N}{2}$ **do**
    $solutions[i] \leftarrow solutions_{initial}[i]$
  end for
  for $i \leftarrow \frac{N}{2} + 1, N$ **do**
    $rnd \leftarrow$ RANDOM_INTEGER($\frac{N}{2} + 1, M$)
    $solutions[i] \leftarrow solutions_{initial}[rnd]$
  end for
  SORT($solutions$)
  $\mathbf{x}_{best} \leftarrow solutions[1]$
  $n_{eval} \leftarrow n_{eval} + M$
  $n_{L_{eval}} \leftarrow n_{L_{eval}} + M$
end function

Algorithm 3 Identification of similar individuals

function CHECK_DIVERSITY()
  for $i \leftarrow 1, N - 1$ **do**
    $\mathbf{x}_i \leftarrow solutions[i]$
    for $j \leftarrow i + 1, N$ **do**
      $\mathbf{x}_j \leftarrow solutions[j]$
      if MAX($|\frac{\mathbf{x}_i - \mathbf{x}_j}{\mathbf{x}_j}|$) $\leq \epsilon$ **then**
        $\mathbf{x} \leftarrow$RANDOM_SAMPLING(1)
        EVALUATE($\mathbf{x}$)
        $solutions[j] \leftarrow \mathbf{x}$
        $n_{eval} \leftarrow n_{eval} + 1$
        $n_{L_{eval}} \leftarrow n_{L_{eval}} + 1$
      end if
    end for
  end for
end function

Algorithm 4 Generation of the offspring population

function COMBINATION_METHOD()
  for $i \leftarrow 1, N$ **do**
    $\mathbf{x}_i \leftarrow solutions[i]$
    for $j \leftarrow 1, N$ **do**
      $\mathbf{x}_j \leftarrow solutions[j]$
      if $i \neq j$ **then**
        $\alpha \leftarrow -1$
        if $i < j$ **then**
          $\alpha \leftarrow 1$
        end if
        $\beta \leftarrow \frac{|j-i|-1}{N-2}$; $\boldsymbol{\delta} \leftarrow \frac{\mathbf{x}_j - \mathbf{x}_i}{2}$
        $\mathbf{c}_1 \leftarrow \mathbf{x}_i - \boldsymbol{\delta}(1 + \alpha\beta)$; $\mathbf{c}_2 \leftarrow \mathbf{x}_i + \boldsymbol{\delta}(1 - \alpha\beta)$
        $\mathbf{r} \leftarrow$ RANDOM(D); $\mathbf{x} \leftarrow \mathbf{c_1} + (\mathbf{c}_2 - \mathbf{c}_1) \bullet \mathbf{r}$
        ▷ where $\bullet$ represents the Hadamard Product
        EVALUATE($\mathbf{x}$)
        $offspring[i][j] \leftarrow \mathbf{x}$
      end if
    end for
    $n_{eval} \leftarrow n_{eval} + N - 1$
    $n_{L_{eval}} \leftarrow n_{L_{eval}} + N - 1$
  end for
end function

Algorithm 5 Update of the current population

function GO_BEYOND(i)
  $\mathbf{x}_{pr} \leftarrow solutions[i]$
  $\mathbf{x}_{ch} \leftarrow offspring[i][1]$
  $improvement \leftarrow 1$
  $\lambda \leftarrow 1$
  while $f(\mathbf{x}_{ch}) < f(\mathbf{x}_{pr})$ **do**
    $\mathbf{c}_1 \leftarrow \mathbf{x}_{ch} - \frac{(\mathbf{x}_{pr} - \mathbf{x}_{ch})}{\lambda}$
    $\mathbf{c}_2 \leftarrow \mathbf{x}_{ch}$
    $\mathbf{x} \leftarrow$ RANDOM_UNIFORM($\mathbf{c}_1, \mathbf{c}_2$)
    EVALUATE($\mathbf{x}$)
    $\mathbf{x}_{pr} \leftarrow \mathbf{x}_{ch}$; $\mathbf{x}_{ch} \leftarrow \mathbf{x}$
    $improvement \leftarrow improvement + 1$
    if $improvement = 2$ **then**
      $\lambda \leftarrow \frac{\lambda}{2}$
      $improvement \leftarrow 0$
    end if
    $n_{eval} \leftarrow n_{eval} + 1$
    $n_{L_{eval}} \leftarrow n_{L_{eval}} + 1$
  end while
  return $\mathbf{x}_{pr}$
end function

```
function UPDATE_POPULATION()
    for i ← 1, N do
        SORT(offspring[i])
        x_i ← solutions[i]
        x_o ← offspring[i][1]
        if f(x_o) < f(x_i) then
            x ← GO_BEYOND(i)
            solutions[i] ← x
            n_stuck[i] ← 0
        else
            n_stuck[i] ← n_stuck[i] + 1
            if n_stuck[i] > n_change then
                x ←RANDOM_SAMPLING(1)
                EVALUATE(x)
                solutions[i] ← x
                n_eval ← n_eval + 1
                n_{L_eval} ← n_{L_eval} + 1
                n_stuck[i] ← 0
            end if
        end if
    end for
    SORT(solutions)
    if f(solutions[1]) < f(x_best) then
        x_best ← solutions[1]
        last_{B_local} ← 0
        last_{R_local} ← 0
        if it ≥ 2 · n_change then
            apply_local ← True
        end if
    else
        last_{R_local} ← last_{R_local} + 1
    end if
end function
```

Algorithm 6 Local search main routines

```
function LOCAL_SEARCH_1()
    z ← LOCAL_SEARCH(x_best, opt)
    LOCAL_EVALUATE(z, x_best)
end function

function LOCAL_SEARCH_2()
    y ← FLATTEN(offspring)
    y_q ← SORT(y)
    ▷ y_q = {y_{q,1}, y_{q,2}, ..., y_{q,M}} so that
    ▷ f(y_{q,i}) ≤ f(y_{q,j}) if i < j
    y_d ← ∅
    for i ← 1, LENGTH(y) do
        d ← ∅
        for j ← 1, LENGTH(solutions_local) do
            d = d∪EUCL(y[i], solutions_local[j])
        end for
        y_d ← y_d∪ MIN(d)
    end for
    y_d ← SORT(y_d)
    ▷ y_d = {y_{d,1}, y_{d,2}, ..., y_{d,M}} so that
    ▷ d_i ≥ d_j if i < j
    for k ← 1, LENGTH(y) do
        score[k] = (1 − balance) · i + balance · j
        ▷ where i is the index of y_k in y_q and
        ▷ j is the index of y_k in y_d
    end for
    y_min = MIN(score)
    z ← LOCAL_SEARCH(y_min, opt)
    LOCAL_EVALUATE(z, y_min)
end function
```

Algorithm 7 Local search subroutines

```
function LOCAL_SEARCH(z, opt)
    if opt = "Powell" then
        z_local ← POWELL(z)
    else
        z_local ← L-BFGS-B(z)
    end if
    n_eval ← n_eval + n_local
    return z_local
end function

function LOCAL_CHECK(z)
    if f(z) < f(x_best) then
        last_{R_local} ← 0
        last_{B_local} ← 0
        apply_local ← True
        solutions[−1] ← z
        SORT(solutions)
        x_best ← solutions[1]
        return True
    else
        last_{R_local} ← last_{R_local} + 1
        last_{B_local} ← last_{B_local} + 1
        apply_local ← False
        return False
    end if
end function

function LOCAL_EVALUATE(z, z_1)
    if LOCAL_CHECK(z) = False then
        opt_old ← opt
        if opt = "Powell" then
            opt ← "L-BFGS-B"
        else
            opt ← "Powell"
        end if
        z_new ← LOCAL_SEARCH(z_1, opt)
        LOCAL_CHECK(z_new)
        if f(z_new) < f(z) then
            if z_new ∉ solutions_local then
                solutions_local ← solutions_local ∪ z_new
            end if
        else
            opt ← opt_old
            if z ∉ solutions_local then
                solutions_local ← solutions_local ∪ z
            end if
        end if
    else
        if z ∉ solutions_local then
            solutions_local ← solutions_local ∪ z
        end if
    end if
end function
```