On the Design of a New Stochastic Meta-Heuristic for Derivative-Free Optimization

N. C. Cruz[1](✉), Juana L. Redondo[2], E. M. Ortigosa[1], and P. M. Ortigosa[2]

[1] Department of Computer Architecture and Technology, University of Granada, Granada, Spain
{ncalavocruz,ortigosa}@ugr.es

[2] Department of Informatics, University of Almería, ceiA3 Excellence Agri-food Campus, Almería, Spain
{jlredondo,ortigosa}@ual.es

Abstract. Optimization problems are frequent in several fields, such as the different branches of Engineering. In some cases, the objective function exposes mathematically exploitable properties to find exact solutions. However, when it is not the case, heuristics are appreciated. This situation occurs when the objective function involves numerical simulations and sophisticated models of reality. Then, population-based meta-heuristics, such as genetic algorithms, are widely used because of being independent of the objective function. Unfortunately, they have multiple parameters and generally require numerous function evaluations to find competitive solutions stably. An attractive alternative is DIRECT, which handles the objective function as a black box like the previous meta-heuristics but is almost parameter-free and deterministic. Unfortunately, its rectangle division behavior is rigid, and it may require many function evaluations for degenerate cases. This work presents an optimizer that combines the lack of parameters and stochasticity for high exploration capabilities. This method, called Tangram, defines a self-adapted set of division rules for the search space yet relies on a stochastic hill-climber to perform local searches. This optimizer is expected to be effective for low-dimensional problems (less than 20 variables) and few function evaluations. According to the results achieved, Tangram outperforms Teaching-Learning-Based Optimization (TLBO), a widespread population-based method, and a plain multi-start configuration of the stochastic hill-climber used.

Keywords: Black-box optimization · Direct search · Stochastic meta-heuristic

1 Introduction

Optimization problems are ubiquitous. They usually arise in fields such as Architecture, Engineering, and Applied Sciences in general [3,7,19]. Broadly speaking, this sort of problem requires finding the extremes (maxima or minima, depending on the goal) of a function. The latter, called objective function in this context, involves different variables and models some aspect of interest. For instance, it can represent the cost of manufacturing a product depending on the providers selected and the quantities bought. In this situation, the points sought would be the minima, i.e., the values of the variables resulting in the minimum function value (cost). Alternatively, if the function modeled the strength of the resulting product, the points of interest would presumably be the maxima of the corresponding function. One of the applications in which optimization stands out is model tuning, where the parameters become variables, and the objective function is the comparison between the achieved and desired output [4,12]. It allows automating processes that used to rely on experts and might be biased by them.

Depending on the objective function, constraints, and variables (e.g., continuous or discrete), there exist different types of optimization problems and methods to address them. For example, linear objective functions and constraints with real bounded variables generally result in problems relatively easy to solve [2,5]. However, this is not always the case, especially when the functions involved do not exhibit a closed analytical form or do not have exploitable mathematical properties (such as linearity and convexity). Fortunately, there exist methods with fewer problem requirements that can even treat it as a black box. They usually rely on intuitive ideas to obtain acceptable results [10,15]. These strategies are known as heuristics when they are problem-specific approaches and meta-heuristics of general-purpose. Similarly, the use of randomness serves to enhance the exploration capabilities at the expense of uncertainty [4].

The formulation of the optimization problem that attracts the attention of this work is as follows:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & L_i \le x_i \le U_i, \; i = 1, \ldots, N. \end{aligned} \tag{1}$$

where f is a N-dimensional objective function, i.e., $f : \mathbb{R}^N \to \mathbb{R}$. The term x refers to any input in $\mathbb{R}^N$ belonging to the region $[L_1, U_1] \times \ldots \times [L_N, U_N]$, which is known as the search space. As can be seen, the problem only consists of the objective function and the bounds of each variable. There is no extra information about the mathematical properties of f (e.g., convexity, linearity and smoothness). It can only be evaluated in the N-dimensional domain defined as the search space. Thus, this problem can be classified as a black-box optimization one with box constraints [3,7].

There exist numerous population-based meta-heuristics that can be applied to the problem defined above [1,15]. Traditional genetic algorithms, Differential Evolution [4,13], and the Universal Evolutionary Global Optimizer (UEGO) [6,11] are good examples. However, they have multiple parameters, so they require fine parameter tuning. Teaching-Learning-Based Optimization (TLBO) [4,14] avoids this problem as a population-based method that only needs the population size and the number of iterations. Regardless, this sort of method will generally need numerous function evaluations to converge due to its haphazard exploration strategy. Another option especially conceived for black-box optimization is DIRECT [7]. In contrast to the previous ones, it is a deterministic method. Furthermore, it only expects the number of function evaluations allowed and a tolerance factor that usually has little effect on its performance [8]. However, its rectangle division mechanism is rigid, and it may require an excessive number of function evaluations for degenerate cases.

This work presents an optimizer that balances the virtual lack of parameters, the use of randomness, and extensive exploration capabilities. To achieve this, it takes inspiration from previous optimization algorithms. The resulting method, called Tangram, defines an exploration procedure that moves from the center of the search space towards its corners. It keeps track of the best result achieved so far but uses that reference to divide the search space into smaller regions according to a deterministic division scheme. The new zones are ultimately explored by a local optimizer known as SASS [9], which is stochastic yet has a robust default configuration. Tangram aims to be effective for low-dimensional problems (less than 20 variables) and to be compatible with low budgets of function evaluations.

The rest of the paper is structured as follows: Sect. 2 describes the proposed method in detail. Section 3 explains the experimentation carried out. Finally, Sect. 4 shows the conclusions and states the future work.

2 Method Description

For a given optimization problem in the form of Eq. 1, the algorithm Tangram only expects as input the number of allowed function evaluations. Like DIRECT, the optimizer sees the search space as an N-dimensional unit hypercube, so it is first scaled accordingly from $\mathbb{R}^N$ to $[0,1]^N$. This strategy simplifies the implementation of the method and avoids issues concerning variables of significantly different scales at local search [16]. This idea is depicted in Fig. 1 for a hypothetical problem in a sub-domain in $\mathbb{R}^3$, $[10,20] \times [0,7] \times [1,3]$. The space of solutions remains unaltered.

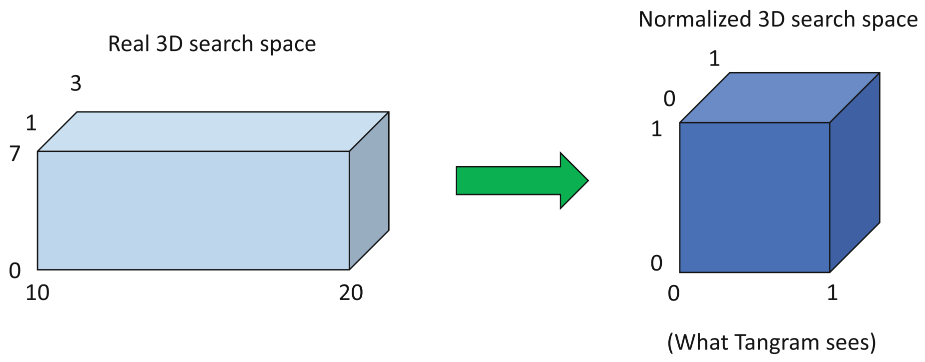


Fig. 1. Normalization of a hypothetical 3D search space.

Again like DIRECT, Tangram starts by evaluating the center of the search space, i.e., $(0.5, \ldots, 0.5) \in \mathbb{R}^N$, which becomes the current result. This process defines its initialization stage, which is shown in Fig. 2a. After that, it compares the budget of function evaluations to the dimensionality of the search space and enters into one of two modes, namely, the standard or the incisive one. Although both keep the same fundamentals, there is a subtle yet significant difference between them. The standard mode is covered first, as it was the only one at preliminary design stages. The local solver, which remains the same, is also explained in detail. Later, the decision criterion and the incisive mode are described in terms of the potential flaws of the standard one.

2.1 Standard Execution Mode

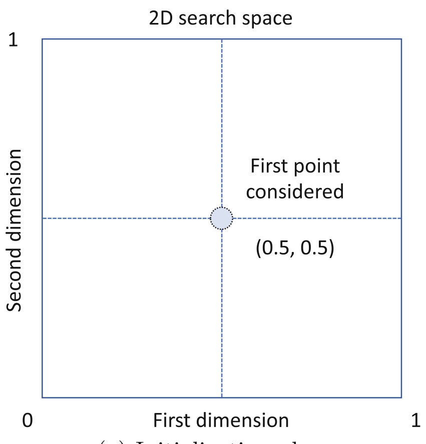
Algorithm 1 outlines Tangram (omitting the selection between modes). As can be seen, the initial steps described above (without the mode selection) are in lines 1 and 2.

Algorithm 1. Tangram optimizer (Standard Mode only)

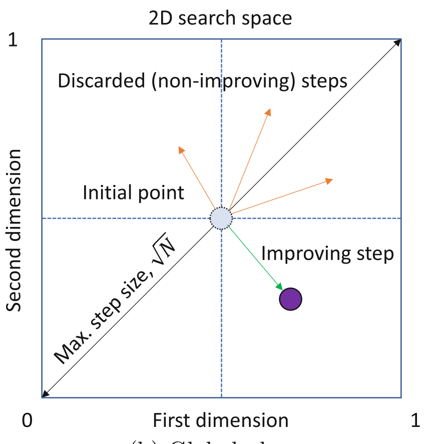
```
   Input: Function: $f : \mathbb{R}^N \to \mathbb{R}$; Int: $evals$
1  Point corners[$2^N$] = get_Corners_Of_Hypercube($N$);
2  Point $result$ = $(0.5, \ldots, 0.5) \in \mathbb{R}^N$;
3  while $evals > 0$ do
4      $result$, = Global_Phase($result$);            // Change if improved only!
5      Point midpoints[$2^N$];
6      for $corner \in corners$ do
7          midpoints[$corner$] = $(result + corner)/2$;
8      end
9      midpoints = sort(midpoints, order=ascending $f$);
10     for $point \in midpoints$ do
11         $midpoints[point]$ = Local_Phase($point$, radius=$|corner - point|$);
12     end
13     $result$ = best_Of($result \cup midpoints$);
14 end
15 return $result$;
```

The optimization loop defining the standard mode is between lines 3 and 14. It lasts while there are function evaluations remaining and consists of the following stages:

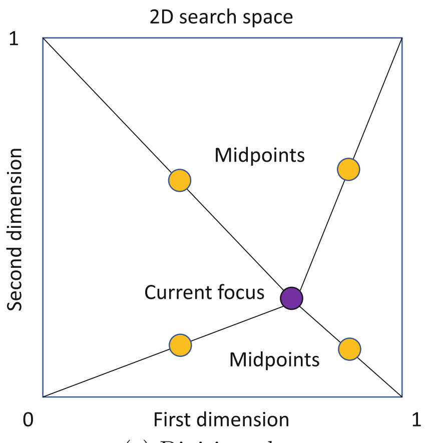
Global Phase (line 4): The local search component, SASS, is launched from the current result to try to improve it. This is the local solver generally used with the aforementioned memetic algorithm, UEGO, and has been chosen because of its effectiveness for different problems. The optimizer is configured so that the maximum step size is equal to the diameter of the search space, i.e., $\sqrt{N}$.



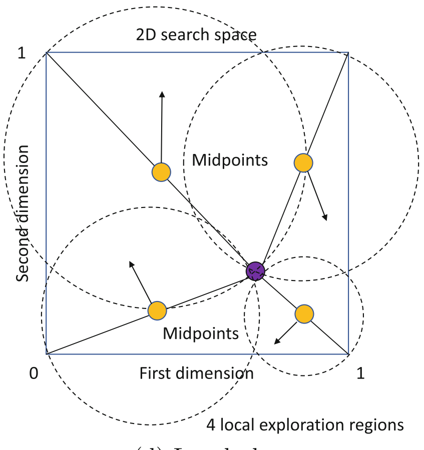
(a) Initialization phase



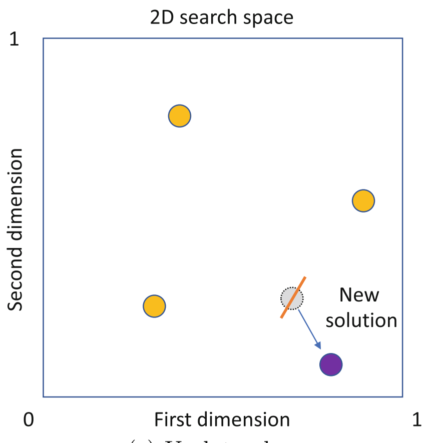
(b) Global phase



(c) Division phase



(d) Local phase



(e) Update phase

Fig. 2. Main concepts of Tangram.

This strategy allows reaching any solution and also comes from the way in which UEGO handles its initial or first-level species. Figure 2b depicts this process starting from the initial point, i.e., the center of the search space, for a hypothetical 2D problem. Notice how SASS attempts several movements, but the current result only changes when a better point is found.

Division (lines 5 to 9): Tangram computes and evaluates the midpoint between each corner of the search space and the current result. This part emphasizes the exploration of the search space. It allows the method to escape from local optima and identify new promising regions. It is vaguely inspired in how DIRECT keeps a representing point of every region of the search space. However, the division does not keep the regions strictly isolated. Figure 2c represents this stage graphically extending the previous context. It also serves to explain the name chosen for the proposed optimizer. Namely, if the resulting regions were colored in different colors, they would look like a Tangram, i.e., the widespread Chinese dissection puzzle [18].

Local Phase (lines 10 to 12): Tangram launches its local optimizer, SASS, from each of the previous midpoints to improve them. Those having a better value for the objective function are chosen first. By proceeding this way, in case the number of function evaluations allowed is low, the method ensures having explored the most promising regions at least. This aspect is relevant because the optimizer has been conceived for situations in which calling the objective function is computationally demanding. At this stage, in contrast to the global phase, SASS is configured not to take steps bigger than the distance between each starting midpoint and the corner used to define it (line 11). The further the midpoint is from the current global result, the wider initial region it has. Thus, exploration is automatically enhanced while the whole search space always remains virtually covered. Figure 2d summarizes these ideas graphically. Finally, notice that since the local solver updates its current point every time that it finds a better one, the definition of regions is dynamic like in UEGO.

Update (line 13): If any of the points found after the division and local searches outperforms the current solution, that point replaces it. Then, the algorithm returns to the global phase and starts to repeat the previous process if there are function evaluations available. This stage is show in Fig. 2e.

After consuming all the function evaluations allowed, Tangram returns the best solution achieved so far (line 15). However, despite being omitted from Algorithm 1, notice that the method registers every evaluation of the objective function. Hence, this situation can be detected at any stage. If that happens, Tangram assigns an infinite value to any new point and tries to finish as soon as possible.

2.2 Local Search Component (SASS)

Regarding the local search component, as introduced, it is SASS, which was initially proposed in [17]. This name is an acronym for Single-Agent Stochastic

Search. SASS does not have any special requirement for the objective function apart from being fully defined in the search space. For this reason, it is especially suitable for black-box optimization. It was one of the main reasons why it was chosen for UEGO, and the same criterion has been followed for Tangram.

SASS is a stochastic hill-climber of adaptive step size. It starts at any given point, which is treated as its current local solution, and randomly decides a direction to move. The jump size cannot be greater than a given threshold. As explained, it will be $\sqrt{N}$ for the global phase and the distance between the midpoint and the corner used to compute it for any local one. However, the jump size is further scaled depending on the number of improving (accepted) and non-improving (discarded) movements.

In practical terms, every movement consists in generating a new candidate solution, x', according to Eq. (2), where x is the current solution and ξ is a normally-distributed random vector (perturbation). The standard deviation, σ, is globally defined between 1e−5 and 1, starting at the upper bound. Every component has a specific mean or bias factor that is initially set to 0. They form the bias vector, $b = (b_1, \ldots, b_N)$. If the movement amplitude (the module of the perturbation), is greater than allowed, ξ is rescaled by the maximum step size.

$$x' = x + \xi \tag{2}$$

SASS then computes the objective function at x'. If it is better than the current solution, x' replaces it, the iteration is considered successful, the bias vector is updated as $b = 0.2b + 0.4\xi$, and a new iteration starts. Otherwise, the opposite direction is explored by generating and evaluating a new candidate solution, $x'' = x - \xi$. If it outperforms the current one, x'' replaces it, and the iteration is also considered successful. In this situation, the bias vector is updated as $b = b - 0.4\xi$. However, if neither x' nor x'' are better than the current solution, the iteration is tagged as failed, and the bias vector is set to $b = 0.5b$. Figure 3 shows the key aspects of an iteration of SASS in Tangram, which forces any perturbation vector to be within a delimited (yet moving) region.

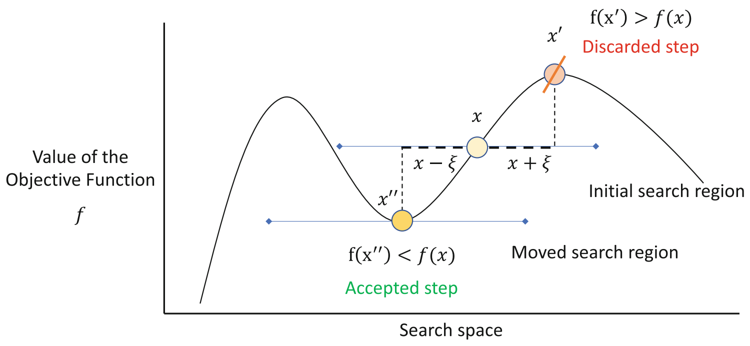


Fig. 3. Depiction of an iteration of SASS.

The global standard deviation, σ, is doubled after five consecutive successful movements (expansion) or halved after three consecutive discarded ones (contraction), which represents the adaptive nature of SASS. This is the recommended configuration, and it is known to perform well. Besides, within Tangram, SASS will terminate after 32 iterations. This arbitrary threshold is assumed enough to converge to the nearby optima according to previous knowledge on the method when used with UEGO. Nevertheless, varying this local budget would just result in a second parameter to tune.

2.3 Incisive Execution Mode

According to the previous explanation, the first function evaluation is always for the center of the search space. Then, the loop of the standard execution mode starts. Every global phase takes 32 evaluations through SASS. After that, the division requires 2^N function evaluations, one for each new midpoint. Let us think of a problem with $N = 20$, which is the highest dimensionality for a problem expected to be addressed with Tangram. It would be approximately 1×10^6 evaluations, which is the recommended budget for a robust execution of UEGO. Later, the local phase would end the current iteration after consuming $32{*}2^N$ evaluations, i.e., 2^{N+5}.

In this context, if the function evaluation budget is low (less than $33 + 2^N$), Tangram will not even be able to reach the local stage. This means that the local search will have been launched only once. Thus, the candidate solutions competing with the current result will be a subset of midpoints not sharpened by SASS. Accordingly, the probability of having identified a competitive point in the search space is low. The incisive mode of Tangram deals with this problem.

The incisive mode is selected over the standard one when the number of function evaluations allowed is less than $33 + 2^N$. This mode maintains the structure of the standard one but has a relevant modification in the division and local phases, which are merged. Namely, instead of generating and evaluating all the midpoints before moving to the local exploration phase, Tangram launches SASS from every new midpoint iteratively. By proceeding this way, the proposed method avoids the lack of local optimization that results in working with non-sharpened points at the expense of not prioritizing the most promising regions.

3 Experimentation and Results

The goals of Tangram are to be effective for low-dimensionality problems, without tuning requirements, and with a low budget of function evaluations. These aims are well aligned with the experimentation described in [3]. Thus, the 20 continuous box-constrained problems proposed in that work, which feature between 1 and 10 variables, have been addressed with Tangram. As the authors say, they are challenging for black-box methods not exploiting any analytical information. The limit of function evaluations has been computed as in the referred paper, i.e., $30(N+1)$, which is low and makes the test more challenging. It is also compatible with a context where the cost function is computationally demanding (e.g., simulation-based model tuning).

To have an adequate reference, Tangram has been compared to a pure random search (PRS), a plain multi-start configuration of SASS (MSASS), and the population-based optimizer TLBO. The pure random search is expected to be a baseline reference, since any optimizer should be more effective than simply generating and evaluating points. The multi-start SASS is a especially descriptive comparison: Since Tangram can be seen as a rule-based multi-start component linked to SASS, it should serve as a better guide for this local solver rather than simply generating random starts. Finally, TLBO has been included in the comparison due to its fame of being simple to tune and effective [4]. It is also relevant to highlight that both TLBO and MSASS achieved very good results in the model tuning application described in [4], which supports their selection.

All of the optimizers considered have been configured with the same function evaluation limit. This task is more difficult for TLBO, as it depends on a population size and a number of cycles. For this reason, TLBO takes that limit at least but slightly exceeds it in some cases. The development environment used is MATLAB 2020a in Mac OSX (MacBook Pro, Intel i5 2.9 GHz, 8 GB of RAM). Each method has been executed 200 times for each case to handle stochasticity.

Table 1 contains the results of experimentation. More specifically, the first column lists the 20 problems addressed including their name with their dimensionality and optimum value below. The other columns show the results obtained with each optimizer in the form of the average value of the objective function $\pm$ the standard deviation below. The cells in bold highlight the best result of the row. As can be seen in the results, the average differs from the theoretical optimum in most cases. Namely, only problems 3, 4, and 5 have a method that virtually offers the best possible value. This situation confirms the challenging nature of the testbeds for this kind of method and the evaluation budget allowed.

In this context, Tangram stands out as the best performing solver, as it offers the best average result for all the instances. In general, the number of function evaluations allows it to run in the standard mode. However, for problems 11, 14, and 15, the incisive mode is selected according to the defined criterion (e.g., for problem 8, the budget is 270, while the threshold is 289, so the incisive mode is activated). Its role is decisive, as the results of the standard mode for instances 11, 14, and 15 would have been $8003.54550 \pm 2889.46980$, 332.05440 ± 61.81830, and 246.38150 ± 63.87150, respectively. Hence, the predominant position of the proposed optimizer would be lost in those degenerate cases. There is another revealing fact to highlight: MSASS performs worse that Tangram even though they share the same local solver. Thus, the design of Tangram seems effective.

Regarding the other methods, as expected, PRS is the worst option due to its complete lack of search orientation. MSASS and TLBO approximately share the second position, with 12 relative victories for the former and 8 for the latter. Regardless, the differences are small in general, and both methods exhibit particularly bad averages in some problems. See for example problems 7 for MSASS and 12 for TLBO, where they are almost equivalent to PRS.

Table 1. Results of the compared optimizers for the 20 test problems.

	Problem	PRS	MSASS	Tangram	TLBO
1	branin (2D: 0.3979)	0.96322 ±0.56865	0.88190 ±0.56618	**0.59423 ±0.28007**	0.87950 ±0.62720
2	camel (2D: –1.0316)	–0.81055 ±0.17976	–0.90562 ±0.16800	**–0.94933 ±0.14442**	–0.90070 ±0.16960
3	ex4_1_1 (1D: –7.4873)	–6.69670 ±1.22100	–7.11600 ±1.01150	**–7.42420 ±0.17009**	–7.28160 ±0.66860
4	ex4_1_2 (1D: –663.4994)	–656.42850 ±13.03620	–661.33590 ±6.22790	**–662.86990 ±1.51650**	–660.71710 ±10.67700
5	ex8_1_1 (2D: –2.0218)	–1.86310 ±0.10054	–2.02000 ±0.00483	**–2.02140 ±0.00115**	–1.99390 ±0.08030
6	ex8_1_4 (2D: 0)	1.01380 ±0.77884	0.81421 ±0.89664	**0.45123 ±0.59453**	0.68060 ±0.79930
7	goldsteinprice (2D: 3)	24.15690 ±20.62910	21.46320 ±25.74360	**12.00310 ±14.97650**	16.31050 ±17.93680
8	hartman3 (3D: –3.8626)	–3.63850 ±0.14490	–3.71340 ±0.16565	**–3.78000 ±0.09150**	–3.67310 ±0.18480
9	hartman6 (6D: –3.3224)	–2.30300 ±0.36590	–2.72140 ±0.30703	**–2.83540 ±0.24749**	–2.59240 ±0.30490
10	least (3D: 14085.1398)	89426.35870 ±48162.68080	85657.53800 ±90868.74070	**54380.77930 ±23588.04900**	66667.12100 ±46139.03410
11	perm0_8 (8D: 1000)	7594.46260 ±2814.81180	4386.96390 ±2552.2268	**4193.35220 ±2640.35760**	4781.10300 ±2837.30440
12	perm_6 (6D: 1000)	2410159.72000 ±1880505.3559	1348865.22460 ±1705484.8604	**1040601.32000 ±1771209.76700**	2010600.02910 ±2541900.01131
13	rbrock (2D: 0)	17.69760 ±27.8399	13.49500 ±60.25080	**4.22410 ±5.3311**	7.91002 ±13.0823
14	schoen_10_1 (10D: –1000)	311.78160 ±54.87700	194.78420 ±79.80000	**180.87800 ±92.37480**	223.21380 ±106.12960
15	schoen_10_2 (10D: –1000)	292.26700 ±43.38010	117.05820 ±96.74010	**87.80030 ±104.40370**	179.37180 ±110.95960
16	schoen_6_1 (6D: –1000)	162.39550 ±115.7606	–45.82550 ±207.03060	**–300.34410 ±228.3106**	108.37580 ±124.07530
17	schoen_6_2 (6D: –1000)	126.52030 ±83.2584	–90.22060 ±186.51570	**–325.30290 ±192.2352**	10.06960 ±156.32220
18	shekel10 (4D: –10.5363)	–1.25000 ±0.51079	–1.86710 ±1.03470	**–3.20280 ±1.34020**	–1.87330 ±1.09240
19	shekel5 (4D: –10.1532)	–0.88923 ±0.47061	–1.60540 ±0.99205	**–3.50500 ±1.39050**	–1.45500 ±0.77001
20	shekel7 (4D: –10.4028)	–1.03920 ±0.41355	–1.65020 ±0.90632	**–3.26410 ±1.45330**	1.73670 ±0.78700

4 Conclusions and Future Work

In this work, a new algorithm for black-box optimization has been described. The method is virtually parameter-free, as it only expects the number of allowed function evaluations, and aims to be effective for low-dimensionality problems (up to 20 variables). It is expected to be valid for model tuning through numerical optimization. Its name is Tangram, and it combines several ideas from the literature. More specifically, it works in a unitary hypercube and tries to apply a deterministic strategy to define regions, like DIRECT. Besides, its behavior can be self-adapted to adapt the search to best-effort explorations in which the algorithm detects that it will not be able to execute all its steps appropriately. The proposal also uses a local optimizer from the literature, SASS, which is the one usually used within the memetic algorithm UEGO. From this population-based method, Tangram replicates the scheme of making SASS focus on different regions by limiting the steps yet keeping one covering the whole search space. Nevertheless, Tangram avoids the sophisticated management of points as a population in UEGO.

Tangram has been used to address 20 benchmark problems from the literature in a context of few function evaluations allowed. Its performance has been compared to a pure random search, a random multi-start configuration of SASS, and TLBO, a widespread population-based method. Tangram outperforms all of them. Among the appreciated aspects, one of the most descriptive ones is the fact that Tangram performs better than SASS in the multi-start configuration, which supports the design of the proposal. Another interesting idea to highlight from the experimentation is how the adaptive nature of Tangram is effectively able to anticipate the potential flaws of its standard mode and opts for the incisive one instead.

For future work, we will study how to automatically detect the local convergence of SASS to save function evaluations that can be reassigned to other parts of the method. We also intend to increase the number of benchmark problems and the optimizers compared, possibly starting with DIRECT and UEGO.

Acknowledgements. This work has been supported by the Spanish Ministry of Science through the projects RTI2018-095993-B-I00, financed by MCIN/AEI/10.13039/501100011033/ and FEDER ("A way to make Europe"), as well as INTSENSO (MICINN-FEDER-PID2019-109991GB-I00), by Junta de Andalucía through the projects UAL18-TIC-A020-B, CEREBIO (P18-FR-2378) and P18-RT-1193 by the European Regional Development Fund (ERDF). N.C. Cruz is supported by the Ministry of Economic Transformation, Industry, Knowledge and Universities from the Andalusian government.

References

1. Boussaïd, I., Lepagnot, J., Siarry, P.: A survey on optimization metaheuristics. Inf. Sci. **237**, 82–117 (2013)
2. Boyd, S., Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge University Press (2004)

3. Costa, A., Nannicini, G.: RBFOpt: an open-source library for black-box optimization with costly function evaluations. Math. Program. Comput. **10**(4), 597–629 (2018). https://doi.org/10.1007/s12532-018-0144-7
4. Cruz, N.C., Marín, M., Redondo, J.L., Ortigosa, E.M., Ortigosa, P.M.: A comparative study of stochastic optimizers for fitting neuron models. application to the cerebellar granule cell. Informatica **32**, 477–498 (2021)
5. Griva, I., Nash, S.G., Sofer, A.: Linear and nonlinear optimization, vol. 108. Siam (2009)
6. Jelasity, M., Ortigosa, P.M., García, I.: Uego, an abstract clustering technique for multimodal global optimization. J. Heuristics **7**(3), 215–233 (2001)
7. Jones, D.R., Martins, J.R.R.A.: The DIRECT algorithm: 25 years later. J. Global Optim. **79**(3), 521–566 (2021)
8. Jones, D.R., Perttunen, C.D., Stuckman, B.E.: Lipschitzian optimization without the lipschitz constant. J. Optim. Theory Appl. **79**(1), 157–181 (1993)
9. Lančinskas, A., Ortigosa, P.M., Žilinskas, J.: Multi-objective single agent stochastic search in non-dominated sorting genetic algorithm. Nonlinear Anal. Model. Control **18**(3), 293–313 (2013)
10. Lindfield, G., Penny, J.: Introduction to nature-inspired optimization. Academic Press (2017)
11. Marín, M., Cruz, N.C., Ortigosa, E.M., Sáez-Lara, M.J., Garrido, J.A., Carrillo, R.R.: On the use of a multimodal optimizer for fitting neuron models. application to the cerebellar granule cell. Frontiers Neuroinformatics **15**, 663797 (2021)
12. Monterreal, R., Cruz, N.C., Redondo, J.L., Fernández-Reche, J., Enrique, R., Ortigosa, P.M.: On the optical characterization of heliostats through computational optimization. In: Proceedings of SolarPACES 2020, pp. 1–8 (2020)
13. Price, K., Storn, R.M., Lampinen, J.A.: Differential evolution: a practical approach to global optimization. Springer Science & Business Media (2006)
14. Rao, R.V., Savsani, V.J., Vakharia, D.P.: Teaching-learning-based optimization: an optimization method for continuous non-linear large scale problems. Inf. Sci. **183**(1), 1–15 (2012)
15. Salhi, S.: Heuristic Search. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-49355-8
16. Snyman, J.A., Wilke, D.N.: Practical Mathematical Optimization. SOIA, vol. 133. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77586-9
17. Solis, F.J., Wets, R.J.B.: Minimization by random search techniques. Math. Oper. Res. **6**(1), 19–30 (1981)
18. Wang, F.T., Hsiung, C.C.: A theorem on the Tangram. Am. Math. Mon. **49**(9), 596–599 (1942)
19. Zou, F., Wang, L., Hei, X., Chen, D.: Teaching-learning-based optimization with learning experience of other learners and its application. Appl. Soft Comput. **37**, 725–736 (2015)