Noname manuscript No.
(will be inserted by the editor)

Default Policies for Global Optimisation of Noisy Functions with Severe Noise

Spyridon Samothrakis, Maria Fasli, Diego Perez, and Simon Lucas

Received: 1/12/2014 / Accepted: date

Abstract Global optimisation of unknown noisy functions is a daunting task that seems to appear in domains ranging from games to control problems to meta-parameter optimisation for machine learning. We show how to incorporate heuristics to Stochastic Simultaneous Optimistic Optimization (STOSOO), a global optimisation algorithm that has very weak requirements from the function. In our case, heuristics come in the form of Covariance Matrix Adaptation Evolution Strategy (CMA-ES). The new algorithm, termed Guided STOSOO (STOSOO-G), combines the ability of CMA-ES for fast local convergence (due to the algorithm following the "natural" gradient) and the global optimisation abilities of STOSOO. We compare all three algorithms in the "harder" parts of the COCO-BBOP benchmark suite, which provides a default set of functions for testing. We show that our approach keeps the best of both worlds, i.e. the almost optimal exploration/exploitation of STOSOO with the local optimisation strength of CMA-ES.

Keywords Evolutionary Computation, Tree Searches

1 Introduction

The problem of black-box noisy optimisation poses an interesting universality; it is so fundamental to most problem solving that, it is often not recognised as such. Most algorithms are concerned with identifying "local" minima, following some form of gradient. Gradient based solutions however are mostly limited by convergence to local minima. But algorithms tailored directly towards global optimisation tend to be slow, as they need to almost exhaust the search space before being able to give answers with any certainty. Nevertheless, it should be possible to combine global search with some stronger heuristic, some kind of optimizer that will approach local minima fast. In this paper, we combine an algorithm for global optimisation of arbitrary functions, termed Stochastic Simultaneous Optimistic Optimization (STOSOO) [10][1] with the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [5], and produce a third algorithm termed Guided STOSOO (STOSOO-G).

Spyridon Samothrakis, Maria Fasli, Diego Perez and Simon M. Lucas are with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, United Kingdom

[1] The original name of the algorithm is StoSOO, we refer it here as STOSOO, since we choose to capitalise all algorithm names for consistency.

Re-iterating the basic idea behind the paper, STOSOO does a very principled exploration of the search space, but can be slow in multi-dimensional settings, whereas CMA-ES, which follows closely the "natural" gradient, can be used as part of the optimisation process. The method is largely inspired by Monte Carlo Tree Search (MCTS), a technique that has helped improve performance immensely in domains such as Computer Go [1], which are mostly however in discrete domains. In MCTS, external or a-priori knowledge is embedded in the search in the form of patterns [4], which help improve the algorithm significantly. To the best of our knowledge, this is the first time that STOSOO is combined with an evolutionary inspired heuristic. Our hope is that the approach pioneered by MCTS, i.e., two different search methods, will prove equally strong in continuous domains.

Another way of seeing this paper, and hence the methods introduced, is to think of the role of the global optimiser as a very smart "restart" strategy, not that different from partial restarts [9]. Assuming infinite computation, continuous restarts of an algorithm from anywhere in the search space will help find a solution. In the case of a very noisy function however, it is hard to discern when such restarts should happen and what information should be retained. Coupling a global search method that is robust to heavy noise with a local search alleviates the problem to a certain extend. This is our basic hypothesis, which we try exploring here.

The rest of the paper is organised as follows. We start by providing a more formal presentation of the problem of noisy optimisation in Section 2. We present STOSOO in Section 3, followed by CMA-ES in Section 4. We describe our algorithm in Section 5. We then present the experimental framework on which we will benchmark all algorithms in Section 6, followed by a discussion of the experiments in Section 7. We conclude with a brief discussion on Section 8 on possible uses of our method within the wider Machine Learning Community.

2 Function optimisation

Function optimisation refers to the process of trying to find a local maxima or minima of a function, also known as "optima". The problem can be formalised as follows. Let us define a function $f : \boldsymbol{X} \rightarrow Y$, where $X \in \mathcal{R}^n$ and $Y \in \mathcal{R}$, with n being the length of vector $\boldsymbol{X}$. $\boldsymbol{X}$ is usually called the search space of the function. The goal of function minimisation is to find a $\boldsymbol{x}_0$ for which $f(\boldsymbol{x}) > f(\boldsymbol{x_0})$ for all x. Alternatively, one can look for $f(\boldsymbol{x}) < f(\boldsymbol{x_0})$ for all x, which is known as maximisation. In our case, we also assume that the function is perturbed in some way by a noise source and that the noise produced is independent of the function we are interested in optimising. If we assume that each noisy sample is termed r_t, then we can say $\mathbb{E}[r_t|x_t] = f(x_t)$. The difference between the best $f(\boldsymbol{x})$ and the one our algorithm found is termed *regret*, and the goal of optimisation is to minimise it. The problem of function optimisation is extremely generic and can be found under a host of settings. It is well known that no method for optimisation has any advantage over another [11] in the general case, however a number of functions that commonly occur in most engineering problems have the following qualities: they have some form of local smoothness, i.e. it is possible to find local derivatives/gradients. The function is not completely ill-posed, meaning that small changes in $\boldsymbol{x}$ result in small (or at least manageable) changes in $f(\boldsymbol{x})$. They are often non-convex, which makes it easy to get stuck in local minima for most search algorithms. The problem of noise polluting samples is also ever-prevalent, although the source of this noise can vary wildly from application to application. Noise sources range from sensor imperfections and artefacts to multiple agents interacting and, depending on their intensity,

can impact the algorithm severely. In this paper, we are only interested in functions where the noise level is significant proportion of the overall signal.

3 StoSOO

Stochastic Simultaneous Optimistic Optimization (STOSOO) [10] is a method of uncovering the maxima of a function (provided they exist) by iteratively breaking the search space down into different regions and sampling from them. For example, assuming function $\boldsymbol{X} \in [0, 1]$, STOSOO will initially sample from the whole range $[0, 1]$, and proceed into breaking this into K sampling points (in most cases, $K = 2$ or $K = 3$). Assuming $K = 2$, it will now break down the search space into two leaves $[0, 0.5]$, $(0.5, 1]$ and keep sampling from these leaves until some criterion is met (e.g., running out of a function evaluation budget). This way a "coverage" tree is created, with the root node being $[0, 1]$, a multitude of nodes in the middle, and a multitude of leaves at the bottom of the tree. Each node is evaluated k times. During each iteration, the algorithm decides which nodes in the tree to sample from (by calling the function "Sample"), based on a term called b-value $b_{i,j,max} = max\left\{b_{node[min_i,max_j]}\right\}$.

Algorithm 1 The STOSOO Algorithm.

function STOSOO(min, max,n)
  $k = n/log^3(n), h_{max} = \sqrt{n/k}, \delta = 1/\sqrt{(n)}$
  $\mathcal{T} \leftarrow \{node[min, max]\}$ ▷ Root Node
  $t \leftarrow 0$ Total Number of evaluations
  while $t < n$ **do**
    $b_{max} \leftarrow -\infty$
    for $h = 0$ to h_{max} **do**
      for each leaf $node[min_i, max_j]$ **do**
        $b \leftarrow \mathbb{E}(Node_{i,j}) + \sqrt{log(nk/\delta)/2T(Node_{i,j})}$ ▷ $T(Node_{i,j})$ is the number of times this node has been sampled
        $b_{node[min_i,max_j]} \leftarrow b$
      $b_{i,j,max} = max\left\{b_{node[min_i,max_j]}\right\}$
      $best_{i,j} = arg\max_{i,j}\left\{b_{node[min_i,max_j]}\right\}$
      if $b_{i,j,max} > b_{max}$ **then**
        if $T(best_{i,j}) < k$ **then**
          $S(Node_{i,j}) \leftarrow S(Node_{i,j})+$ **SAMPLE()** ▷ Draw a random sample
          $T(Node_{i,j}) \leftarrow T(Node_{i,j}) + 1$
          $\mathbb{E}(Node_{i,j}) = S(Node_{i,j})/T(Node_{i,j})$
        else
          Expand $Node_{i,j}$ and get its children nodes
          $b_{max} = b_{i,j,max}$
  return Node with highest $\mathbb{E}(Node_{i,j})$

Where to search next is mediated by constantly calculating $b - values$, which are a sum of exploration and exploitation terms, $b \leftarrow \mathbb{E}(Node_{i,j}) + \sqrt{log(nk/\delta)/2T(Node_{i,j})}$. The first part of the sum is the mean of the node, and refers to the *exploitation*. The second part is the *exploration* part of the equation. A totally greedy version of the algorithm would have the exploitation term only. The algorithm is presented in Algorithm 1. Note that the only assumption STOSOO makes is that the function to be optimised is Lipschitz continuity

around the optima, that is there is an upper limit to how fast the function can change. There is no requirement to know this limit, just that there exists one.

4 CMA-ES

The Covariance Matrix Adaptation - Evolutionary Strategy (CMA-ES) is an evolutionary algorithm designed for continuous domains, specially suited for non-linear and non-convex optimization problems [5]. In general, this algorithm is applied to those problems that are not constrained and are made of up to 100 dimensions.

The algorithm is based on using the multivariate normal distribution (MND). A multivariate vector $X = (x_0, x_1, \dots x_N), x_i \in \mathcal{R}$ is said to have an MND if it satisfies that any linear combination from its components, $w_0x_0 + w_1x_1 + \cdots + w_Nx_N$, is normally distributed. CMA-ES creates a population of individuals by sampling from an MND: $\mathcal{N}(m, C)$, which is uniquely defined by the distribution mean $m \in \mathcal{R}^n$ and its covariance matrix $C \in \mathcal{R}^{n\times n}$. The top of the density function, which corresponds to m, also determines the translation of the distribution. The covariance matrix C, positive definite and symmetric, determines the shape of the distribution and its graphical interpretation: it defines an iso-density ellipsoid $\{x_i \in \mathcal{R}|(x-m)^TC^{-1}(x-m) = 1\}$. The covariance matrix is isotropic if $C = \sigma^2 I$, I being the identity matrix. D represents a diagonal matrix, whereas C stands for a positive definite full covariance matrix. The contour lines define a potential objective function. The goal of CMA-ES is to adapt the shape of the distribution to the contour lines of the objective function to be minimized. The goal is obtained by iteratively updating the mean vector m, the covariance matrix C and a step-size σ. The mean vector m of the distribution is updated as $m = \sum_{i=1}^{\mu} w_i x_{i:\lambda} \qquad w_i > 0, \quad i = 1, 2, \dots, \mu$, where each x_i is sampled from the distribution: $x_i \sim m+\sigma\mathcal{N}(0, C)$, μ is the number of individuals taken for recombination, λ is the population size and w_i are the weighted coefficients whose positive recombination sums to 1.

An *Evolution Path* is defined as a sequence of consecutive steps over several generations, and they contain relevant information about the correlation between them. If two consecutive steps are taken in the same direction, the evolution path becomes longer. Evolution paths are used to guide the updates of both C and σ, in order to provide an evolution mechanism that does not converge prematurely but leads to an optimum. *Cumulation* (or *Exponential Smoothing*) is used to build the evolution path at generation g for C (denoted $p_c^{(g)}$), as described by the next Equation, where c_c is a defined constant ≤ 1 and $N(x) = \sqrt{x(2-x)\mu_{ef}}$ is a normalization factor for $p_c^{(g)}$, $p_c^{(g)} = (1 - c_c)p_c^{(g-1)} + \frac{(m-m^{(g-1)})}{\sigma^{(g-1)}}N(c_c)$. Similarly, an evolution path is updated for the step-size control. For a more detailed description of these equations, the interested reader is referred to [5], but the update equation is: $p_\sigma^{(g)} = (1 - c_\sigma)p_\sigma^{(g-1)} + \frac{(m-m^{(g-1)})}{\sigma^{(g-1)}}N(c_\sigma){C^{(g)}}^{(-\frac{1}{2})}$. Once the cumulations have been calculated, both C and σ can be updated. The value of C for the next generation is obtained by applying the *Rank-σ update*. The *Rank-σ update* C_μ extends the rank-one update ($(p_c^{(g)}p_c^{(g)})^T$) for large population sizes. The update is $C^{(g+1)} = (1-c_1-c_\mu)C^{(g)} + c_1(p_c^{(g)}p_c^{(g)})^T + c_\mu C_\mu$, where $C_\mu = \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T, \qquad y_i \sim \mathcal{N}(0, C)$. Finally, the step-size σ is updated using the cumulative path length control method (CSA, or cumulative step-size adaptation). The reasoning is as follows: if the evolution path $p_\sigma^{(g)}$ is long, the steps take similar directions, ergo the same distance can be covered with longer but

fewer iterations. Hence, the step size must be increased. However, if $p_\sigma^{(g)}$ is short, the steps take opposite directions (they cancel each other) and their size should be decreased. Then, given $p_\sigma^{(g)}$, σ is updated as follows: $\sigma^{(g+1)} = \sigma^{(g)} \quad \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|p_\sigma^{(g)}\|}{E\|\mathcal{N}(0,I)\|} - 1\right)\right)$.

An initial mean m, step-size σ and population size λ must be provided at the beginning of the algorithm. In our experiments, σ is set to $1/3$ of the range of each variable and $\lambda \leftarrow 4.0 + 3.0 * \log(N)$, where N is the number of dimensions of the problem. An initial mean value m is used that is half the range of each variable.

It was recently shown [7] that CMA-ES is a special case of natural gradient optimisation algorithms, when using a Gaussian distribution as a model for the search parameters. It is obvious that this implies a strong heuristic on the nature/shape of the function CMA-ES can model (and thus optimise for) successfully, and one can easily design cases to get the algorithm stuck on purpose. CMA-ES has been used as derivative-free global optimisation [8], thought this is not its niche. Unfortunately the results of [8] focus on the noise-free case, which differs significantly from our own.

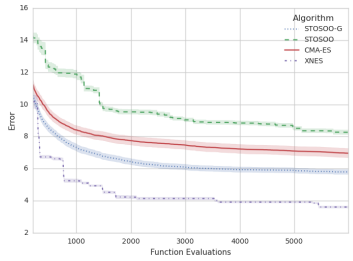
5 STOSOO-G

Our algorithm is a combination of STOSOO and CMA-ES, inspired by the use of patterns [4] in standard Monte Carlo Tree Search. The idea is that in the sampling function (what is termed *default policy*) we embed any knowledge we might have, which however should not impact the performance of the whole algorithm by diverging from any possible optima. Thus, in our case, we use CMA-ES as a kind of a heuristic in the sampling process, modifying STOSOO. Instead of sampling randomly between bounds, we sample based on what CMA-ES proposes. Thus, our algorithm is exactly the same as STOSOO, but uses a modified (bounded) version of CMA-ES as the sampling method. The modification simply involves ensuring that bounds are respected. Now we sample from $v = m + \mathcal{N}(m, C)$ as normal, but if any of the samples goes outside the limits $[min, max]$ that we have set as bounds, we truncate elements for vector v to uniform random values within those bounds. This new CMA-ES now becomes our new sampling method. Whenever STOSOO requires a sample, CMA is used for $l = 4$ iterations (l can be set to any "sensible" default value). If CMA-ES fails to progress for l iterations, the best value CMA discovers is given to STOSOO. In higher dimension functions, the interplay between STOSOO and CMA-ES provides an effective restart mechanism, while in lower dimensions STOSOO can help attack functions that due to their curvature CMA-ES cannot. In this sense, CMA-ES is being used as our default policy. As long as it can improve on the underlying function, we trust it to optimise it, otherwise we follow STOSOO.

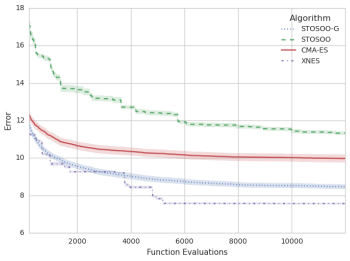
The above ideas have been explored before in the context of global optimisation, especially by algorithms that branch and partition the search space, a popular example of which is MCS [6]. In our case, we combine an algorithm whose only requirement is the function being Lipschitz continuous close to the optima with an algorithm that follows the "natural" Gradient.

6 The BBOP Benchmarking platform

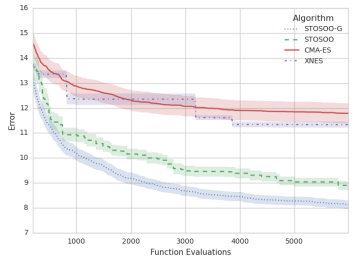
Our benchmarks are conducted on the aforementioned algorithms on a standardised benchmark suite, called BBOP-2012 [3]. We will provide a description of the benchmarks in this



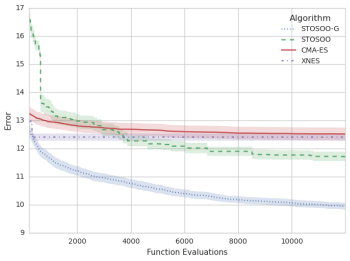
(a) Average error for f_{122} in dimension 20.



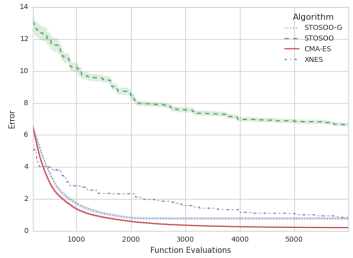
(b) Average error for f_{122} in dimension 40.



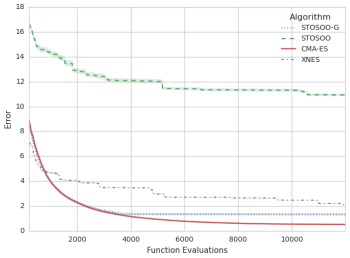
(c) Average error for f_{123} in dimension 20.



(d) Average error for f_{123} in dimension 40.



(e) Average error for f_{124} in dimension 20.



(f) Average error for f_{124} in dimension 40.

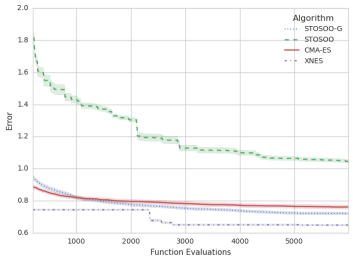
Fig. 1: Figures plots for functions f_{122},f_{123},f_{124}, level of significance $\alpha = 0.05$

section, but for a complete description see: [3]. All functions have a global minimum in $[-5, 5]$. We can begin by defining a set of helper functions:

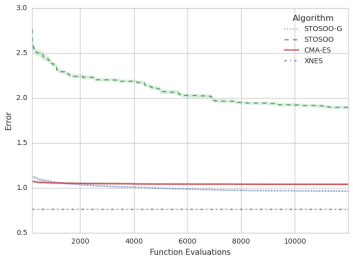
1. D, the dimension of the problem. This can be one of $2, 3, 5, 10, 20, 40$.
2. $f_{opt} = f(x^{opt})$, a function that performs a random transformation on the optimal values x^{opt}, for more detail see the BBOP's manual [3].
3. Λ^a, a diagonal matrix with elements $\lambda = \alpha^{\frac{i-1}{2(D-1)}}$
4. R, Q are rotation matrices, again see BBOP's manual [3] for more detail.

The following noise models are used for all functions

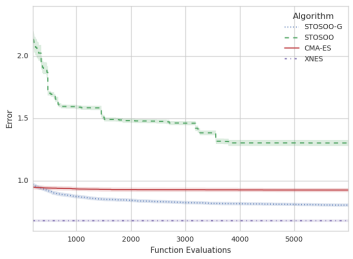
1. **Gaussian Noise:** $f_{GN}(f, \beta) = f \times exp(\beta \mathcal{N}(0, 1))$, where $\beta = 1$.
2. **Uniform Noise:** $f_{UN}(f, \alpha, \beta) = f \times \mathcal{U}(0, 1)^{\beta} max\left(1, \left(\frac{10^9}{g+\epsilon}\right)^{\alpha \mathcal{U}(0,1)}\right)$, with $\beta = 0.01$, $\alpha = (0.49 + 1/D)$ and $\epsilon = 10^{-99}$.



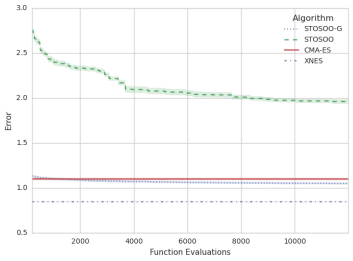
(a) Average error for f_{125} in dimension 20.



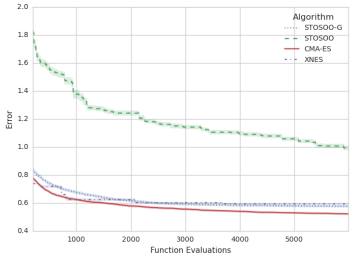
(b) Average error for f_{125} in dimension 40.



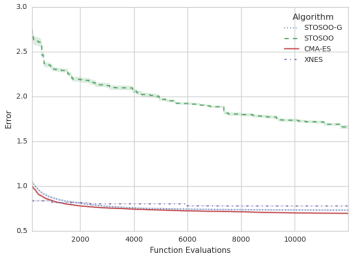
(c) Average error for f_{126} in dimension 20.



(d) Average error for f_{126} in dimension 40.



(e) Average error for f_{127} in dimension 20.



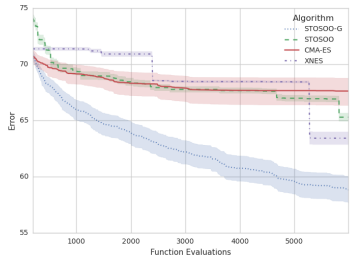
(f) Average error for f_{127} in dimension 40.

Fig. 2: Figures plots for functions f_{125},f_{126},f_{127}, level of significance $\alpha = 0.05$

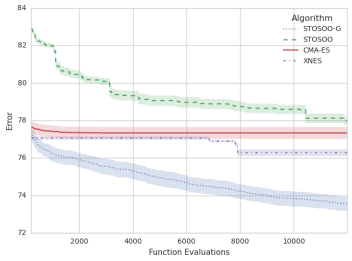
3. **Cauchy Noise:** $f_{CN}(f, \alpha, p) = f + \alpha max(0, 1000 + \mathbb{1}_{\{\mathcal{U}(0,1)<p\}} \frac{\mathcal{N}(0,1)}{|\mathcal{N}(0,1)|+\epsilon}$, with $\alpha = 1, p = 0.2$ and $\epsilon = 10^{-199}$.

Alongside the above functions, the following test functions are defined:

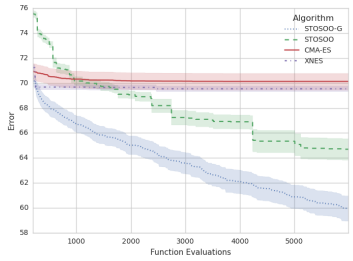
1. **Shaffer's F7:**
$f_{schaffer}(\boldsymbol{x}) = \left(\frac{1}{D-1} \sum_{i=1}^{D-1} \sqrt{s_i} + \sqrt{s_i} sin^2 \left(50 s_i^{1/5} \right) \right), \boldsymbol{z} = \Lambda^{10} \boldsymbol{Q} T_{asy}^{0.5} \left(\boldsymbol{R} \left(\boldsymbol{x} - \boldsymbol{x}^{opt} \right) \right),$
$s_i = \sqrt{z_i^2 + z_{i+1}^2}$
2. **Composite Griewank-Rosenbrock:**
$f_{f8f2}(\boldsymbol{x}) = \left(\frac{1}{D-1} \right) \sum_{i=1}^{D-1} \left(\frac{s_i}{4000} - cos(s_i) \right) + 1,$
$\boldsymbol{z} = max \left(1, \frac{\sqrt{D}}{8} \right) \boldsymbol{R}\boldsymbol{x} + 0.5, s_i = 100 \left(z_i^2 - z_{i+1} \right)^2 + (z_i - 1)^2, \boldsymbol{z}^{opt} = \mathbf{1}$



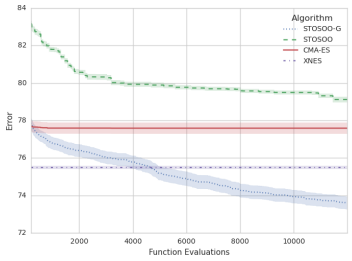
(a) Average error for f_{128} in dimension 20.



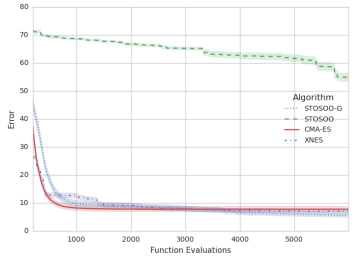
(b) Average error for f_{128} in dimension 40.



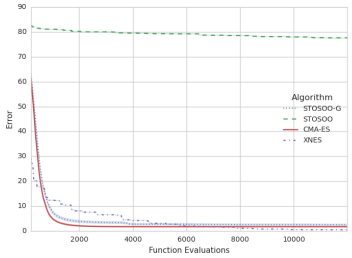
(c) Average error for f_{129} in dimension 20.



(d) Average error for f_{129} in dimension 40.



(e) Average error for f_{130} in dimension 20.



(f) Average error for f_{130} in dimension 40.

Fig. 3: Figures plots for functions f_{128},f_{129},f_{130}, level of significance $\alpha = 0.05$

3. **Gallagher's Gaussian Peaks, globally rotated:**
$f_{gallagher}(\boldsymbol{x}) = T_{osz}(10 - \max_{i=1}^{101} w_i exp\left(-1/2D(\boldsymbol{x} - \boldsymbol{y}_i)^T \boldsymbol{R}^T C_i R(\boldsymbol{x} - \boldsymbol{y}_i)\right))^2, w_i = \begin{cases} 1.1 + 8((i-2)/99) & \text{if } i > 1 \\ 10 & \text{if } i = 1 \end{cases}, \boldsymbol{C}_i = \Lambda^a / a_i^{1/4}$

BBOP also uses a boundary function that heavily penalises values outside the range $[-5, 5]$, $f_{pen} = \sum_{i}^{D} 100max(0, |x_i| - 5)^2$. f_{opt} is a fixed optima.

The benchmarking functions are now defined as a combination of the above functions.

- $f_{122}(\boldsymbol{x}) = f_{GN}(f_{schaffer}(\boldsymbol{x})), 1) + f_{pen}(\boldsymbol{x}) + f_{opt}$
- $f_{123(\boldsymbol{x})} = f_{UN}\left(f_{schaffer}(\boldsymbol{x}), 0.01\left(0.49 + 1/D\right), 1\right) + f_{pen}(\boldsymbol{x}) + f_{opt}$
- $f_{124}(\boldsymbol{x}) = f_{CN}\left(f_{schaffer}(\boldsymbol{x}), 1, 0.2\right) + f_{pen}(\boldsymbol{x}) + f_{opt}$
- $f_{125}(\boldsymbol{x}) = f_{GN}(f_{f8f2}(\boldsymbol{x})), 1) + f_{pen}(\boldsymbol{x}) + f_{opt}$

- $f_{126}(\boldsymbol{x}) = f_{UN}\left(f_{f8f2}(\boldsymbol{x}), 0.01\left(0.49 + 1/D\right), 1\right) + f_{pen}(\boldsymbol{x}) + f_{opt}$
- $f_{127}(\boldsymbol{x}) = f_{CN}\left(f_{f8f2}(\boldsymbol{x}), 1, 0.2\right) + f_{pen}(\boldsymbol{x}) + f_{opt}$
- $f_{128}(\boldsymbol{x}) = f_{GN}(f_{gallagher}(\boldsymbol{x})), 1) + f_{pen}(\boldsymbol{x}) + f_{opt}$
- $f_{129}(\boldsymbol{x}) = f_{UN}\left(f_{gallagher}(\boldsymbol{x}), 0.01\left(0.49 + 1/D\right), 1\right) + f_{pen}(\boldsymbol{x}) + f_{opt}$
- $f_{130}(\boldsymbol{x}) = f_{CN}\left(f_{gallagher}(\boldsymbol{x}), 1, 0.2\right) + f_{pen}(\boldsymbol{x}) + f_{opt}$

7 Experiments

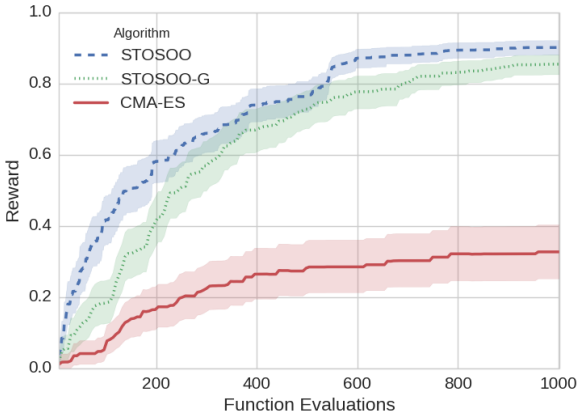


Fig. 4: f_{sp} performance for all three different algorithms, with level of significance $\alpha = 0.05$

Finally, all tests function are run for 2,3,5,10,20 and 40 dimensions. Results in the form of regret (i.e., the optimal result minus what our algorithm found) for statistical significance $\alpha = 0.05$ are printed in Table 1. Bold letters indicate the best result with significance, while italic indicates the worst significant result for the specific function/dimension combination. We also print all the provided graphical representation for all BBOP functions in 20 and 40 dimensions. We can see the results for Gallaghers Gaussian Peaks in Figure 3, for Shaffers F7 in Figure 1 and for Composite Griewank-Rosenbrock in Figure 2. We did 200 runs of each algorithm, in order to get statistical significance in our results. Notice how STOSOO-G performs better in all scenarios in high dimensions, with the exception of f_{130} in 40 dimensions. In all cases, results are better than just using one of the algorithms by itself, with the notable exception of f_{124}, which is a variant of Shaffers F7. STOSOO-G is significantly better in 13/18 functions in 40 dimensions. Notice also that in cases like functions 128/129 where STOSOO-G performs comparatively well in higher dimensions, STOSOO-G performs much better than both algorithms. This is mostly in settings with Gaussian or Uniform noise, while Cauchy noise seems to be more of a problem. This is to be expected, as STOSOO-G and STOSOO calculate means. CMA-ES is a ranked based algorithm, so the median would be a more robust estimator under Cauchy noise. In lower dimensions think are slightly more confusing, with no clear winner, but a clear loser in CMA-ES. Obviously, the performance of STOSOO-G is impacted, but not enough as to give significantly weaker

performance in the majority of functions. Across dimensions we can see various function performances, ranging from STOSOO-G being better in every instance (as in f_{129}), to no clear winner.

Overall, STOSOO-G maintains the advantages of STOSOO. It is safe to claim that on average, STOSOO-G achieves performance close to one of its best components, either its heuristic/default policy, CMA-ES, or the main search function, STOSOO. Even in the case where the algorithm does not perform that well, its performance is closer to its best constituent. In higher dimensions, where the strength of the default policy (i.e. CMA-ES) plays a more significant role, the strength of the combined method shines, whereas in lower dimension CMA-ES does not misguide the search.

In order to showcase the behaviour in lower dimensions, a final experiment worth discussing in greater was performed. We tried to maximise for function [10]

$$f_{sp}(x_0, x_1) = \begin{cases} 0.5(sin(13x_1)sin(27x_1)) + 0.5 + \mathcal{U}[-0.16, 0.16] & \text{if } |x_0 - 1| \leq 0.01 \\ 0 & \text{if } |x_0 - 1| > 0.01 \end{cases}$$

, which has maximum at $f_{sp}(0.99, 0.867526) \approx 0.975599$. The function is constant for quite a large part of x_0, returning 0 in all of them and it has bounds around $[0, 1]$. From Figure 4 we can see that, though CMA-ES struggles to perform adequately, it merely slows down the quality of solutions found without impacting the quality itself, as both algorithms overlap after a while. This slowing down is to be expected, as CMA-ES is providing a suboptimal heuristic, confusing rather than helping the overall search process. W

8 Conclusion

We have shown that one can adapt the heuristic methods used in Monte Carlo Tree Search to problems in continuous domains. To the best of our knowledge, this is the first time something like this has been explored. Our algorithm outperforms its constituent parts in most cases under heavy noise. Overall we would like to emphasise the beneficial combination of two search strategies within the same search algorithm. A top level, accurate one, trying to weed out the "hard" part of search, while another, simpler one guiding search using some kind of heuristic method. The intuition here is that a complex search problem can be decomposed into simpler, smaller, "tidier" search problems and these can be combined back to a global search algorithm.

Next steps involve testing the algorithm in real domains, which could range from open loop planning to Reinforcement Learning control problems. An interesting easily explorable possible next step is to push simpler algorithms as the default policy (e.g., Linear Regression, problem specific heuristics) and use them instead of CMA-ES as the local search model. If possible, one might even substitute problem dimensions with ad-hoc heuristics tailored to the specific problem. In that respect, what we are proposing is mostly a framework, rather than an algorithm itself.

In real world problems, the sampling method does not have to be as sophisticated as CMA-ES. It can be any method that incorporates domain knowledge. Another promising research avenue is to combine STOSOO with off-policy algorithms (e.g., see [2]), which in our case would signify one policy exploring and one policy building a different model of the environment.

	Dimension	2	3	5	10	20	40
Function	**Algorithm**						
f_{122}	STOSOO-G	0.196 ± 0.032	0.504 ± 0.052	1.455 ± 0.083	2.691 ± 0.119	$\mathbf{5.798 \pm 0.151}$	$\mathbf{8.466 \pm 0.119}$
	STOSOO	0.202 ± 0.028	$\mathbf{0.278 \pm 0.041}$	1.328 ± 0.068	$\mathit{3.826 \pm 0.123}$	$\mathit{8.266 \pm 0.143}$	$\mathit{11.325 \pm 0.087}$
	CMA-ES	$\mathit{1.165 \pm 0.299}$	$\mathit{1.108 \pm 0.232}$	$\mathit{2.024 \pm 0.217}$	2.920 ± 0.224	6.961 ± 0.287	9.973 ± 0.208
f_{123}	STOSOO-G	0.620 ± 0.068	$\mathbf{1.306 \pm 0.084}$	2.440 ± 0.113	$\mathbf{4.809 \pm 0.158}$	$\mathbf{8.121 \pm 0.166}$	$\mathbf{9.953 \pm 0.115}$
	STOSOO	$\mathbf{0.457 \pm 0.042}$	1.513 ± 0.091	2.423 ± 0.103	5.952 ± 0.129	8.786 ± 0.142	11.715 ± 0.158
	CMA-ES	$\mathit{4.372 \pm 0.929}$	$\mathit{3.352 \pm 0.325}$	$\mathit{3.931 \pm 0.225}$	$\mathit{7.580 \pm 0.461}$	$\mathit{11.787 \pm 0.392}$	$\mathit{12.518 \pm 0.216}$
f_{124}	STOSOO-G	0.098 ± 0.018	0.199 ± 0.028	0.367 ± 0.042	0.491 ± 0.044	0.786 ± 0.064	1.338 ± 0.075
	STOSOO	0.097 ± 0.010	$\mathit{0.446 \pm 0.044}$	$\mathit{1.041 \pm 0.039}$	$\mathit{3.099 \pm 0.081}$	$\mathit{6.648 \pm 0.103}$	$\mathit{10.939 \pm 0.092}$
	CMA-ES	$\mathit{0.476 \pm 0.181}$	0.130 ± 0.049	$\mathbf{0.193 \pm 0.053}$	$\mathbf{0.090 \pm 0.012}$	$\mathbf{0.212 \pm 0.027}$	$\mathbf{0.529 \pm 0.043}$
f_{125}	STOSOO-G	0.004 ± 0.001	0.046 ± 0.004	$\mathit{0.184 \pm 0.009}$	0.455 ± 0.010	$\mathbf{0.721 \pm 0.009}$	$\mathbf{0.964 \pm 0.008}$
	STOSOO	0.007 ± 0.001	$\mathit{0.082 \pm 0.008}$	0.150 ± 0.006	$\mathit{0.562 \pm 0.016}$	$\mathit{1.045 \pm 0.011}$	$\mathit{1.896 \pm 0.015}$
	CMA-ES	0.006 ± 0.002	0.040 ± 0.005	0.144 ± 0.011	$\mathbf{0.422 \pm 0.013}$	0.761 ± 0.013	1.041 ± 0.012
f_{126}	STOSOO-G	$\mathbf{0.006 \pm 0.001}$	0.061 ± 0.005	$\mathbf{0.235 \pm 0.011}$	$\mathbf{0.553 \pm 0.013}$	$\mathbf{0.807 \pm 0.011}$	$\mathbf{1.049 \pm 0.009}$
	STOSOO	0.009 ± 0.001	0.052 ± 0.006	$\mathit{0.303 \pm 0.010}$	$\mathit{0.825 \pm 0.017}$	$\mathit{1.304 \pm 0.023}$	$\mathit{1.961 \pm 0.022}$
	CMA-ES	$\mathit{0.016 \pm 0.004}$	$\mathit{0.076 \pm 0.010}$	0.271 ± 0.019	0.608 ± 0.020	0.927 ± 0.015	1.099 ± 0.012
f_{127}	STOSOO-G	0.004 ± 0.001	0.042 ± 0.003	0.161 ± 0.008	0.378 ± 0.010	0.577 ± 0.008	0.732 ± 0.006
	STOSOO	0.004 ± 0.001	0.036 ± 0.005	0.146 ± 0.008	$\mathit{0.510 \pm 0.007}$	$\mathit{0.993 \pm 0.015}$	$\mathit{1.662 \pm 0.018}$
	CMA-ES	0.007 ± 0.004	0.029 ± 0.003	$\mathbf{0.108 \pm 0.007}$	$\mathbf{0.305 \pm 0.009}$	$\mathbf{0.522 \pm 0.008}$	$\mathbf{0.697 \pm 0.006}$
f_{128}	STOSOO-G	0.036 ± 0.017	0.528 ± 0.075	2.073 ± 0.198	20.566 ± 1.099	$\mathbf{58.771 \pm 1.167}$	$\mathbf{73.550 \pm 0.381}$
	STOSOO	0.035 ± 0.007	0.431 ± 0.088	1.859 ± 0.149	20.477 ± 0.998	65.294 ± 0.354	$\mathit{77.992 \pm 0.230}$
	CMA-ES	$\mathit{0.741 \pm 0.105}$	$\mathit{1.227 \pm 0.121}$	$\mathit{2.778 \pm 0.456}$	$\mathit{30.176 \pm 2.249}$	$\mathit{67.620 \pm 1.237}$	77.337 ± 0.302
f_{129}	STOSOO-G	$\mathbf{0.061 \pm 0.019}$	$\mathbf{0.632 \pm 0.082}$	$\mathbf{3.271 \pm 0.260}$	$\mathbf{22.426 \pm 0.929}$	$\mathbf{59.959 \pm 1.008}$	$\mathbf{73.604 \pm 0.331}$
	STOSOO	0.168 ± 0.023	1.000 ± 0.103	4.344 ± 0.370	26.995 ± 0.932	64.700 ± 0.865	$\mathit{79.120 \pm 0.139}$
	CMA-ES	$\mathit{0.732 \pm 0.095}$	$\mathit{1.622 \pm 0.144}$	$\mathit{7.288 \pm 0.668}$	$\mathit{39.606 \pm 1.807}$	$\mathit{70.126 \pm 0.672}$	77.590 ± 0.295
f_{130}	STOSOO-G	0.089 ± 0.027	0.503 ± 0.077	1.293 ± 0.088	$\mathbf{3.703 \pm 0.490}$	$\mathbf{5.657 \pm 0.772}$	2.528 ± 0.359
	STOSOO	$\mathbf{0.014 \pm 0.003}$	0.405 ± 0.059	1.398 ± 0.128	$\mathit{9.925 \pm 0.871}$	$\mathit{54.906 \pm 1.495}$	$\mathit{77.601 \pm 0.209}$
	CMA-ES	$\mathit{0.736 \pm 0.093}$	$\mathit{1.095 \pm 0.120}$	1.484 ± 0.216	5.982 ± 0.890	7.785 ± 1.028	$\mathbf{1.734 \pm 0.295}$

Table 1: Average regret for BBOP Functions with severe noise, with level of significance $\alpha = 0.05$

References

1. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1), 1–43 (2012)
2. Degris, T., White, M., Sutton, R.S.: Linear off-policy actor-critic. In: In International Conference on Machine Learning. Citeseer (2012)
3. Finck, S., Hansen, N., Ros, R., Auger, A.: Real-Parameter Black-Box Optimization Benchmarking 2010: Noisy Functions Definitions (2012). URL http://coco.lri.fr/downloads/download11.06/bbobdocnoisyfunctions.pdf
4. Gelly, S., Wang, Y., Munos, R., Teytaud, O., et al.: Modification of uct with patterns in monte-carlo go. Tech. rep., INRIA (2006)
5. Hansen, N.: The CMA evolution strategy: a comparing review. In: J. Lozano, P. Larranaga, I. Inza, Bengoetxea (eds.) Towards a new evolutionary computation. Advances on estimation of distribution algorithms, pp. 75–102. Springer (2006)
6. Huyer, W., Neumaier, A.: Global optimization by multilevel coordinate search. Journal of Global Optimization **14**(4), 331–355 (1999)
7. Ollivier, Y., Arnold, L., Auger, A., Hansen, N.: Information-geometric optimization algorithms: A unifying picture via invariance principles. arXiv preprint arXiv:1106.3708 (2011)
8. Rios, L., Sahinidis, N.: Derivative-free optimization: a review of algorithms and comparison of software implementations. Journal of Global Optimization **56**(3), 1247–1293 (2013). DOI 10.1007/s10898-012-9951-y. URL http://dx.doi.org/10.1007/s10898-012-9951-y
9. la Tendresse, I., Gottlieb, J., Kao, O.: The effects of partial restarts in evolutionary search. In: Artificial Evolution, pp. 117–127. Springer (2002)
10. Valko, M., Carpentier, A., Munos, R.: Stochastic Simultaneous Optimistic Optimization. In: 30th International Conference on Machine Learning. Atlanta, États-Unis (2013). URL http://hal.inria.fr/hal-00789606
11. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. Evolutionary Computation, IEEE Transactions on **1**(1), 67–82 (1997)