A SIMPLE ALGORITHM SELECTOR FOR CONTINUOUS OPTIMISATION PROBLEMS

Tarek A. El-Mihoub[1], Christoph Tholen[1,2] , Lars Nolle[1,2],
[1]Department of Engineering Science,
Jade University of Applied Sciences,
26389 Wilhelmshaven, Germany
[2]German Research Center for Artificial Intelligence (DFKI),
Marine Perception,
26129 Oldenburg, Germany
{tarek.el-mihoub | lars.nolle | }@jade-hs.de
christoph.tholen@dfki.de

KEYWORDS

Algorithm Selection Problem, Evolution Strategy, Covariance Matrix Adaptation, Linear Search, STEP, Continuous Optimisation, Nelder-Mead Algorithm, One-Fifth Success Rule, (1+1)-CMA-ES, COCO Framework, BBOB-2009 Testbed.

ABSTRACT

A large number of algorithms has been proposed for solving continuous optimisation problems. However, there is limited theoretical understanding of the strengths and weaknesses of most algorithms and their individual applicability. Furthermore, the performance of these algorithms is highly dependent on their control parameters, which need to be configured to achieve a peak performance. Automating the processes of selecting the most suitable algorithm and the right control parameters can help in solving continuous optimisation problems effectively and efficiently. In this paper, a simple online algorithm selector is proposed. It decides on selecting the right algorithm based on the current state of the search process to solve a given problem. Each algorithm in the portfolio of the algorithm selector competes with others and utilises the results of other algorithms to locate the global optimum. The proposed algorithm selector and the algorithms of the portfolio as stand-alone algorithms were benchmarked on the noise-free BBOB-2009 testbed. The results show that the performance of the simple algorithm selector is better than the performances of the individual algorithms in general. It was also able to solve eleven out of twenty-four functions of the test suite to the ultimate accuracy of 10^{-8}.

INTRODUCTION

A large number of search algorithms for solving optimisation problem has been developed (Cuevas, et al., 2020; Whitley, 2019). In most cases, the performance of these algorithms can be further improved by introducing small modifications through adjusting their control (hyper) parameters (Vermetten, et al., 2020; El-Mihoub, et al., 2014). However, the no free-lunch theorem (Wolpert & Macready, 1997) states that no single algorithm can outperform all other algorithms on all optimisation problems. Based on this theorem, solving any optimisation problem requires selecting a suitable algorithm with a suitable configuration (Huang, et al., 2019). Practitioners usually face the Algorithm Selection (AS) problem and the Algorithm Configuration (AC) problem when applying optimisation algorithms (Kerschke, et al., 2018). The AS and the AC problems have gained increasing attention in the last decades (Huang, et al., 2019; Kerschke, et al., 2018).

Different methodologies have been followed to automate deciding on the right algorithm and the right configuration. Some approaches adopt Rice's framework (Rice, 1976) for solving the AS problem to learn from the algorithms experience in solving different problems (Kerschke, et al., 2018). Most of these approaches use Rice's features-based model to solve the AS (Cruz-Reyes, et al., 2012) and the AC (Belkhir, et al., 2017) problems. Most of these approaches depict the AS and the AC problems as two separated problems in spite of the strong relations between them (Janković & Doerr, 2019). In most cases, the AS and AC problems are addressed sequentially. This separation can affect the algorithm efficiency or narrow the range of problems, on which an approach can be applied. Few researches have dealt with the problem as a Combined Algorithm Selection and Hyper-parameter optimisation (CASH) problem (Vermetten, et al., 2020).

Hyper-heuristics methodology (Drake, et al., 2019) solves the AS and the AC problems using different approaches. The generative hyper-heuristics explore the space of the algorithmic components of algorithms to design a customised search algorithm. This methodology can be used to generate tailor-made and well-configured optimisation algorithms to solve the CASH problem. These approaches are also suitable for solving dynamic problems. However, they can be extremely costly (Miranda, et al., 2017).

On the other hand, the selective hyper-heuristics approach deals with the AS problem as an optimisation problem. A hyper-heuristic is employed to discriminate between different search algorithms based on their

Communications of the ECMS, Volume 36, Issue 1,
Proceedings, ©ECMS Ibrahim A. Hameed, Agus Hasan,
Saleh Abdel-Afou Alaliyat (Editors) 2022
ISBN: 978-3-937436-77-7/978-3-937436-76-0(CD) ISSN 2522-2414

performance on a given problem. Following this approach, a suitable algorithm is selected for each search iteration. Off-line selective hyper-heuristics approaches require featuring different states of different optimisation problems. Whereas, online selective hyper-heuristics approaches incur an additional high cost on the optimisation process.

A simple online algorithm selector with minimum overhead learning costs might be efficient in solving the basic AS problem. Most state-of-the-art optimisation algorithms utilise restart to resample good potential solutions (Hansen, et al., 2021). Instead of restarting, a simple selector can decide to resample a new solution or select the right algorithm for the current search state based on the most recent performances of a set of optimisation algorithms. Based on this idea, a simple algorithm selector is proposed in this paper.

A set of optimisation algorithms was chosen as a portfolio of the algorithm selector. This set consists of one multi-point and three single-point optimisation algorithms. The multi-point algorithm is the Nelder-Mead downhill simplex (Nelder & Mead, 1965). The single-point algorithms are the one plus one Covariance Matrix Adaptation Evolutionary Strategy, i.e. (1+1)-CMA-ES (Igel, et al., 2006), the one plus one Evolution Strategy with one-fifth success rule, i.e. (1+1)-ES (Auger, 2009), and the Line Search with the STEP (Swarzberg, et al., 1994), LSStep algorithm. This set of single- and multi-point algorithms was selected to show that different types of algorithms can be unified easily in the proposed algorithm selector.

In the next section, the simple algorithm selector is introduced. Then, the algorithms that constitute the selector portfolio are briefly described. Before presenting and discussing the results of the benchmarking, the experimental setup is presented. The paper ends with the conclusion and future work.

THE SIMPLE ALGORITHM SELECTOR

Most local search algorithms and even global search algorithms rely on probabilistic restart to achieve globalisation (Pošík & Huyer, 2012). However, sharing the optimisation resources by more than one algorithm can reduce the risk of failure in solving a range of optimisation problems.

The concept of the proposed algorithm selector is to rely on a portfolio of optimisation algorithms instead of relying on a single algorithm with probabilistic restart to reach the global optimum. An algorithm selector with a simple discriminating mechanism based on the current state of the search can solve the AS problem with a minimum overhead cost. The main aim of the algorithm selector is to locate the global optimum and not to find the best algorithm for a given problem.

The algorithm selector should have the ability to unify single point and population-based search algorithms within its framework. The algorithm selector should be capable of distributing the search resources in an effective way. It should be able to select a suitable algorithm based on the current state of the search process and to build on search results of previously selected algorithms. The algorithm selector should enable different algorithms to compete with each other and cooperate to reach the global optimum.

We consider an objective function $f\colon \mathbb{R}^D \longrightarrow \mathbb{R}$, where $x \rightarrow f(x)$ to be minimised. To minimise this function, the proposed algorithm selector follows the algorithm shown in Algorithm 1. The algorithm selector starts by initialising the search environment, line 1 in Algorithm 1. The initialisation process includes selecting an initial point as a potential solution. Then, the selector chooses randomly an algorithm from the portfolio and applies it to the optimisation problem using the potential solution. The selector checks whether the global optimum is reached. In this case, the selector stops. Otherwise, it assigns the selected algorithm a score, which is equal to the change in the fitness of the potential solution. The selector repeats the previous steps until reaching the global optimum or all the algorithms of the portfolio are applied.

In the case of not reaching the global optimum, two algorithms from the portfolio are selected randomly. The algorithm with the best score out of these two algorithms is chosen for utilisation. If the last algorithm applied is selected again, and that algorithm has probably reached a local optimum, the algorithm selector resamples a new potential solution to replace the local reached optimum. Next, the new selected algorithm is applied and its score is updated. The previous steps (12-19 in algorithm 1) are repeated until reaching the target optimum or the search resources are consumed.

Algorithm 1 : The Simple Algorithm Selector

```
1   initiate search environment
2   while not all algorithms of the portfolio selected
3     current algorithm = select an algorithm from the
      algorithms portfolio randomly
4     apply current algorithm on the optimisation
      problem
5     if target value reached
6        print results and exit
7     end if
8     current algorithm score = change in fitness
9     last selected = current algorithm
10  end while
11  while global optimum not reached
12    selected algorithms = select randomly two
      algorithms
13    current algorithm = algorithm with best score of
      selected algorithms
14    if last selected == current algorithm
15      resample new solution
16    end if
17    apply current algorithm on the optimisation
      problem
18    current algorithm score = change in fitness
19    last selected = current algorithm
20  end while
```

The proposed algorithm selector can use a portfolio that consists of a number of single- and multi-point search algorithms. In this paper, a portfolio of four algorithms is implemented. These algorithms are classified as local search algorithms. These algorithms are briefly described in the following sections.

THE NELDER-MEAD DOWINHILL SIMPLEX

The Nelder–Mead algorithm (Nelder & Mead, 1965) is also known as the downhill simplex method. It is an optimisation algorithm for real-value problems. The Nelder-Mead method starts with a set of $D + 1$ initial solutions, a simplex, where *D* is dimension of the search space. A new solution is generated through reflecting the worst solution on the centroid of the remaining *D* solutions. Other operations are conducted to either further improve the new generated solution or to focus on the most promising region of the search space.

A pseudocode of Nelder-Mead method for function minimisation is shown in algorithm 2. The simplex method modifies the vertices of the simplex using four operations to generate better solution based on the fitness of the vertices. These operations are reflection, expansion, contraction and shrinking. The coefficients of these operations areχ, γ, ρ and σ, respectively. Table 1 shows the formulas for executing these operations in addition to the formula for calculating the centroid. The reflection, expansion and the contraction operations are applied to the worst vertex. Meanwhile, the shrinking operation is applied to all vertices except the best one. The standard values for the operations coefficients are $\rho=0.5$, $\chi=2$, $\gamma=0.5$, and $\sigma=0.5$ (McKinnon, 1998) .

Table 1: Operations of Nelder-Mead Method

Operations on simplex	Formula
centroid calculation	$x_c = \frac{1}{n}\sum_{i=2}^{n+1} x_i$
reflection	$x_r = x_c + \rho(x_c - x_{n+1})$
expansion	$x_e = (1+\rho\chi)x_c - x_c x_{n+1}$
outside contraction	$x_{out\_c} = (1+\rho\gamma)x_c - \rho\gamma x_{n+1}$
inside contraction	$x_{in\_c} = (1-\gamma)x_c - \gamma x_{n+1}$
shrinking	$x_{i=2,n+1} = x_i + \sigma(x_i - x_1)$

Starting with an initial solution, a simplex can be generated around this solution. This initial solution together with *D* generated points can be used as starting points, which represent the vertex of the simplex. The simplex can be generated by adding a small value (delta) to each component of the initial solution. In this paper, the value of delta is set to 0.00025 for components with values of zero and a delta of 0.05 for other component values. These values are used by Matlab in the *fminsearch* function (Hansen, 2009).

EVOLUTION STRATEGIES

Evolution strategies (ES) for real-valued optimisation usually rely on Gaussian random variations. They assume that the space around the global optimum can be represented by a multi-variant distribution and the global optimum is at the distribution's centre. Variant ES algorithms have been proposed to locate the global optimum and determine the multi-variant distribution around it by sampling points in the search space.

The (1+1)-ES algorithms start with an initial solution (x_i, $i = 0$) and assumes this is the mean of the distribution. They resample a new solution according to the adopted distribution. Once a better solution is found, this solution becomes the new mean of the distribution. The algorithm keeps a track of the changes in the objective function values and the change in solutions' locations in the search space. The algorithm uses these changes to amend the shape of the distribution to improve the quality of new sampled solutions. In this section, two variants of the (1+1)-ES are concisely described.

Algorithm 2 : The Nelder-Mead Algorithm

```
Input: (D+1) points
while not terminated do
  order the vertices according to their fitness
  $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{D+1})$
  calculate the centroid point of all vertices except
$x_{D+1}, x_c$
  calculate the reflection point $x_r$
  if $f(x_1) \leq f(x_r) < f(x_n)$
    $x_{D+1} = x_r$
  else
    if $f(x_r) < f(x_1)$
      calculate the expansion point $x_e$
      if $f(x_e) < f(x_r)$
        $x_{D+1} = x_e$
      else
        $x_{D+1} = x_r$
      end
    else
      if $f(x_D) \leq f(x_r) < f(x_{D+1})$
        calculate the outside contraction point $x_{out\_c}$
        if $f(x_{out\_c}) =< f(x_r)$
          $x_{n+1} = x_{out\_c}$
        else
          shrink the simplex
       end
     else
       if $f(x_r) \geq f(x_{D+1})$
         calculate the inside contraction point $x_{in\_c}$
         if $f(x_{in\_c}) < f(x_{D+1})$
           $x_{n+1} = x_{cin}$
         else
            shrink the simplex
         end
       end
     end
  end
end
```

The (1+1)-ES with One-Fifth Success Rule Algorithm

This (1+1)-ES algorithm is based on the idea that the search step from the current solution should increase in a case of successive successful steps and should decrease otherwise. Many successful steps indicates that the search can be improved by taking a larger step. On the other hand, very few successful steps indicates the search step might be too large and need to be reduced. According to the one-fifth success rule (Schumer & Steiglitz, 1968), the step-size should not change if the success probability of the sampled solutions is about one-fifth, increase if the success probability is larger than one-fifth and decrease otherwise.

The factors of 1.5 and $1.5^{-1/4}$ for increasing and decreasing the step-size can implement the idea of the one-fifth success rule (Auger, 2009). Pseudocode of the (1+1)-ES with one-fifth success rule is shown in Algorithm 3. A sample from the standard multivariate normal distribution is selected randomly (z_i, line 5). This sample is multiplied by the step size σ and is added to the current mean of the distribution to generate a new solution. The algorithm assumes the distribution is symmetric around the global optimum.

Algorithm 3 : (1+1)-ES with One-Fifth Success Rule

```
1   Input: $x_0, \sigma_0$
2   $x_c = x_0$
3   $\sigma = \sigma_0$
4   while *not terminated* do
5     $z_i = N(0, I)$
6     $x_i = x_c + \sigma z_i$
7     if $f(x_i) \leq f(x_c)$
8       $x_c = x_i$
9       $\sigma = 1.5\,\sigma$
10    else
11      $\sigma = 1.5^{-1/4}\,\sigma$
12    end if
13  end while
```

The (1+1)-CMA-ES Algorithm

CMA-ES algorithm (Hansen, 2006) add a covariance matrix, $C \in \mathbb{R}^{n \times n}$, component to the multi-variant distribution, which is used to generate new solutions. By appropriate adaptation of the covariance matrix, a more accurate representation of the space around the global optimum can be achieved. This help in locating the global optimum, especially, for ill-conditioned problems.
The (1+1)-CMA-ES algorithm starts with an initial solution (x_0), or initial mean, as in any (1+1)-ES algorithm. It also starts with an initial covariance matrix (C_0) of the distribution. Usually, the algorithm starts with a standard multi-variate normal distribution with an initial global step size or sigma (σ_0). As in the case with (1+1)-ES with one-fifth success rule, the changes in the new sampled solution and its cost value are used to adapt the mean the global step size. However, in the (1+1)-CMA-ES, they are also used to adapt the covariance matrix of the distribution.

The covariance matrix C_i needs to be decomposed into Cholesky factors in order to sample a general multivariate normal distribution (i.e. $C_i = A_i A_i^T$). A new solution is generated using the multi-variant distribution as shown in line 10 of Algorithm 4, which shows pseudocode of the (1+1)-CMA-ES algorithm. The global step size is then updated based on the average success rate $p_{succ} \in [0, 1]$. The covariance matrix C is updated in the case of a decrease in the cost values of new solutions compared the cost of the current mean of the distribution. The update is done based on the values of the average success rate p_{succ} , the global step size σ and the evolution path p_c. Table 2 shows the rules for updating these parameters and Table 3 shows the default parameter values (Igel, et al., 2006).

Table 2: Updating rules of (1+1)-CMA-ES

Parameter Name	Updating rules

average success rate	if $f(x_{i+1}) < f(x_i)$
	$p_{succ_{i+1}} = (1 - c_p)p_{succ_i} + c_p$
	else
	$p_{succ_{i+1}} = (1 - c_p)p_{succ_i}$

step size	$\sigma_{i+1} = \sigma_i \times \exp(\frac{1}{d}\left(\frac{p_{succ_i} - p_{succ}^{target}}{1 - p_{succ}^{target}}\right))$

evolution path	if $p_{succ} < p_{thresh}$
	$p_{c_{i+1}} = (1 - c_p)p_{c_i} + \sqrt{c_c(2 - c_c)}Az_i$
	else
	$p_{c_{i+1}} = (1 - c_p)p_{c_i}$

Covariance matrix	if $p_{succ} < p_{thresh}$
	$C_{i+1} = (1 - c_{cov})C_i + c_{cov}.p_c{p_c}^T$
	else
	$C_{i+1} = (1 - c_{cov})C_i + c_{cov}.(p_c{p_c}^T + c_c(2 - c_c)C_i$

Table 3: Parameters of the (1+1)-CMA-ES and their Default Values

Step size control	Covariance matrix adaptation
d: the damping parameter controls the rate of the step size adaptation $d = 1 + \frac{n}{2}$	c_c : the learning rate for the evolution path $c_c = \frac{2}{n+2}$
p_{succ}^{target}: the target success rate $p_{succ}^{target} = \frac{2}{11}$	c_{cov}: the learning rate for the covariance matrix $c_{cov} = \frac{2}{n^2+6}$
c_p : the learning rate for the step size $c_p = \frac{1}{12}$	p_{thresh} : the threshold of the success rate to prevent fast increase of C matrix axes with small step sizes $p_{thresh} = 0.44$

Algorithm 4 : (1+1)-CMA-ES

```
1   Input: $x_0, o_0$
2   $x_c = x_0, \sigma = \sigma_0$
3   $p_{succ} = p_{succ}^{target}, p_c = 0, C = I$
4   while *not terminated* do
5     decompose $C_i$ such that $C_i = AA^T$
6     $z_i = rnadom\ sample\ of\ N(0, I)$
7     $x_{i+1} = x_c + \sigma_i Az_i$
8     $p_{succ_{i+1}} = updated\_average\_success\_rate$
9     $\sigma_{i+1} = updated\_step\_size$
10    if $f(x_i) \leq f(x_c)$
11      $x_c = x_{i+1}$
12      $C_{i+1} = updated\_covariance\_matrix$
13    end if
14  end while
```

LINE SEARCH WITH STEP

Line search algorithms are simple optimisation algorithms. They are effective and efficient in solving separable optimisation problems. They can be used to discriminate between separable and non-separable optimisation test functions (Pošík & Huyer, 2012). The algorithm starts with a randomly selected solution and iterates through individual directions. It optimises the function with respect to the chosen direction while keeping the other components of the solution fixed. Once the optimum in one direction is found, it switches to another direction starting from the best-found solution. No change in the objective value of the solution after going through all the directions indicates a local optimum and the algorithm stops.

The STEP method (Swarzberg, et al., 1994) is a univariate global search algorithm with interval division. The basic idea of the STEP is to sample a new solution with the greatest chance of exceeding the best-found solution.

Therefore, it starts with an initial interval as shown in Algorithm 5. It calculates the objective values of the endpoints of that interval. Then, it divides the interval into two halves by sampling the middle point of the interval. It determines the interval, which has the greatest chance to include a solution, which is better than the current best solution. It repeats the division process for the most promising interval until reaching the stopping criteria. The interval difficulty is used as criteria for selecting the most promising interval. The algorithm selects the interval that enables sampling a solution, which is better than the current best solution.

To determine the difficulty of an interval, STEP assumes a quadratic function $y = ax^2 + bx + c$ that goes through the interval boundaries and $y = f_{best} - \epsilon$, where f_{best} is the objective function of the best found solution and ϵ is a small positive number. The value of ϵ determines the value by which f_{best} is to be at least exceeded. In other words, it determines the tolerance in objective function values. In this paper, the value of ϵ is set to 10^{-8} (Pošík & Huyer, 2012). STEP uses the value of the coefficient a of this quadratic function to measure the interval difficulty. The interval with the smallest value of a is more likely to enable sampling better solutions (Swarzberg, et al., 1994).

Algorithm 5 : The STEP algorithm

```
1   Input: boundaries of the initial interval
2   evaluate the boundaries of the initial interval
3   sample the middle point of the interval
4   evaluate the middle point
5   add the intervals to the interval list
6   while *not terminated* do
7     determine the interval with the lowest difficulty
8     sample the middle point of this interval
9     evaluate the middle point
10    add the new intervals to the interval list
11  end while
```

EXPERIMENTS

This work tries to answer the question 'Can a simple algorithm selector outperform the individual algorithms, those constitute its portfolio, on a range of optimisation problems?'. To answer this question, the performance of the algorithm selector needs to be compared with the performances of the individual algorithms. Experiments were conducted to evaluate the performance of different algorithms on a range of optimisation problems.

Experimental Framework Description

The experiments were carried out using the Comparing Continuous Optimisers (COCO) framework (Hansen, et al., 2021). This framework was also used for the Black-box Optimisation Benchmarking workshop at the GECCO-2009 and 2010 conferences.

The experiments were conducted using the BOBB test suite, which consists of 24 test functions (Hansen, et al., 2009). The functions are classified based on their properties as multimodality, ill-conditioning, global structure and separability. All functions are scalable in terms of dimensionality. The search domain is $[-5; 5]$ for each dimension. Different instances of the same function can be produced by rotating and shifting each function.

Each algorithm was tested for 15 trials on different instances of each function for different dimensions [2, 3, 5, 10, 20, 40].

Algorithm and Experiment Parameter Settings

For fair comparison between the different algorithms, the maximum number of function evaluations for the different algorithms was set to $D \times 10^4$, where D is the dimensionality of the problem. The algorithms were benchmarked using the BBOB2009 settings, i.e. the algorithms were run on the 24 benchmark functions, 5 instances each, 3 trials per instance.

No specific parameter tuning has been done during the experiments. The settings were identical for all functions such that the crafting effort is zero (Hansen, et al., 2010). The control parameters of algorithms within the simple algorithm selector are the same as that of the individual algorithms. The values of the control parameters for the individual algorithms were set as defined in research papers that benchmarked these algorithms using the COCO framework (Auger, 2009; Auger & Hansen, 2009; Posík, 2009). To avoid the impact of the implementation details on the evaluation process, all algorithms have been implemented by the authors and the comparison was done based on results of the implemented algorithms, not on the archived data of the COCO framework. The implementation process was done to accurately replicate the algorithms as described in the papers (Auger, 2009; Auger & Hansen, 2009; Posík, 2009; McKinnon, 1998).

RESULTS AND DISCUSSION

The results of benchmarking the individual algorithms, as implemented by the authors, were compared with the archived data on https://numbbo.github.io/data-archive/bbob/. The comparison shows a big difference in the results of the (1+1)-CMA-ES compared with the archived results on f_5. It also shows a significant difference on f_7. For the (1+1)-ES with one-fifth rule, the comparison shows that there is a significant difference in the performances on f_5, f_8, f_9, f_{10} and f_{11} for the different dimensions. The comparison also shows a significant difference in the results of the STEP algorithm, on f_6, f_8, f_9, f_{10}, f_{15}, f_{21}, and f_{22} for small dimensions (i.e. d=[2, 3, 5]). In most cases, the archived results are better than the results of the conducted experiments. The comparison results are not shown in this paper due to limitation on the number of pages.

The results of the experiments that compares the simple algorithm selector with individual algorithms on the test functions for different dimensions are shown in Figures 1-6. The post-processing tools of the COCO platform were used to generate these plots. In these plots, the best algorithm is the algorithm, which is able to solve the highest fraction of test functions for different target values. The best 2009 line shown in the figures corresponds to the algorithms from BBOB-2009 with the best expected run time for each of the targets considered. Whereas, the Selector, CMA, OneFifth, Simplex and LSStep lines correspond to the algorithms the simple algorithm selector, (1+1)-CMA-ES, (1+1)-ES with fifth rule, the Nelder-Mead and the line search STEP algorithms respectively.

The results show that in general the simple selector algorithm performs better than or at least as good as the best individual algorithm of the algorithms portfolio. The performance in terms of the fraction of function-target pairs of the simple algorithm selector that is better than the individual algorithms for the dimensions of 2, 5, 10, 20 and 40 as shown in Figures 1, 3, 4, 5 and 6. However, for three-dimensional test functions, the Selector and CMA show similar performance as depicted in Figure 2. For larger dimensions as illustrated in Figures 3-6, the difference in the performances between the simple algorithm selector and other algorithms becomes more significant.

The simple algorithm selector has solved the functions f_1, f_2, f_3, f_4, f_5, f_8, f_9, f_{10}, f_{12}, and f_{14}. It was able to reach a target of 10^{-8} for all dimensions and in all the experiments for these test functions. As in any learning mechanisms, an additional learning cost was expected, which can affect its expected run-time to reach different targets. However, the algorithm selector shows a performance, which is better than that of the best 2009 algorithm on f_4 as shown in Figure 7. It was also able to locate the global optimum of f_4 in 15 experiments compared only 6 experiments out of 15 for the best 2009 algorithm.

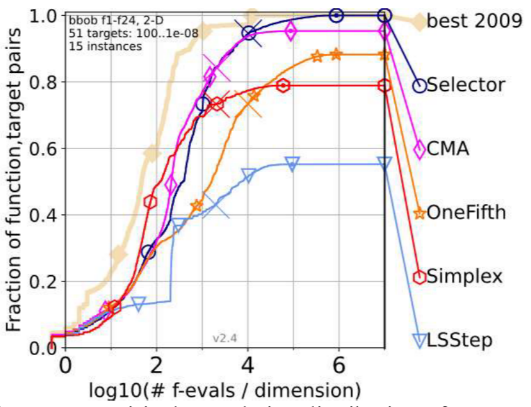

Figure 1: Empirical cumulative distribution of expected run time (ERT) over dimension for 51 targets in $10^{[2..-8]}$ for all functions in 2-D.

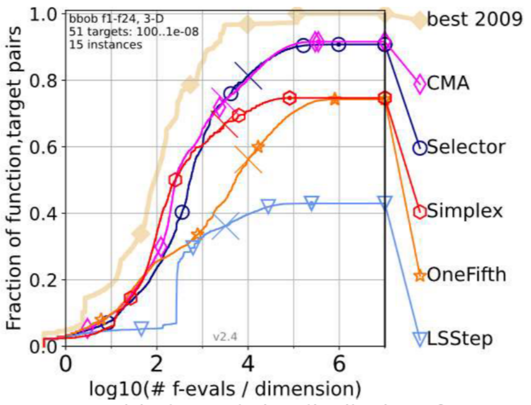

Figure 2: Empirical cumulative distribution of ERT over dimension for all functions in 3-D.

The results show that the simple algorithm selector is able to reach a target of 10^{-8} for the different test functions and for the tested dimensions, more than any of the individual algorithms. Table 4 compares the success rate of the different algorithms for reaching this target for different dimensions. The table clearly shows the superiority of the simple selector over the individual algorithms on all test functions for different dimensions. The success rate is calculated as the ratio of the number of experiments that have reached the target of 10^{-8} to the total number of conducted experiments for a specific dimension on all test functions.

Table 4: The success rate of the algorithms in finding the ultimate precision of 10^{-8} in the 24 function for different dimensions.

Algorithm	Dimension (D)					
	2	3	5	10	20	40
Selector	0.89	0.73	0.62	0.59	0.54	0.49
CMA	0.77	0.63	0.53	0.46	0.42	0.40
Simplex	0.65	0.58	0.48	0.37	0.25	0.07
OneFifth	0.53	0.33	0.21	0.15	0.14	0.09
LSStep	0.23	0.21	0.21	0.20	0.19	0.21

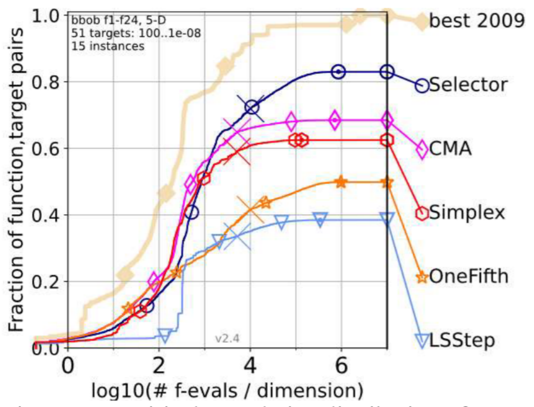

Figure 3: Empirical cumulative distribution of ERT over dimension for all functions in 5-D.

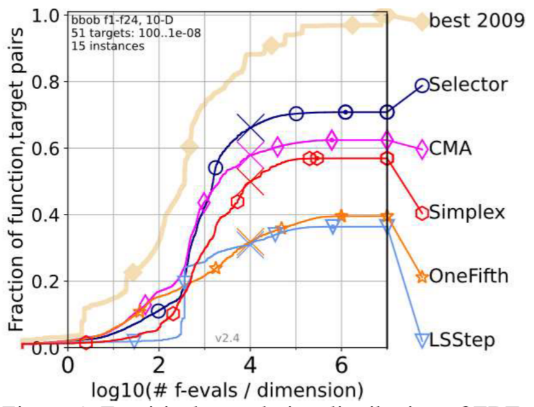

Figure 4: Empirical cumulative distribution of ERT over dimension for all functions in 10-D.

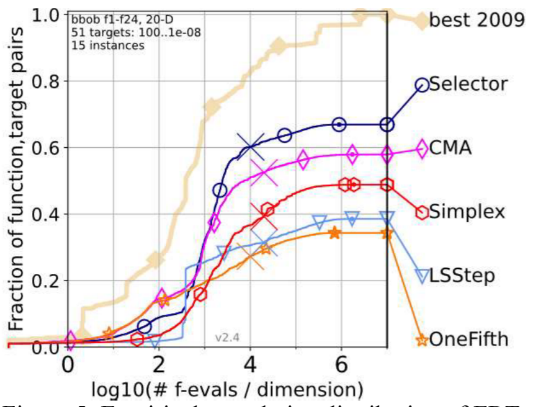

Figure 5: Empirical cumulative distribution of ERT over dimension for all functions in 20-D.

In Table 5, the success rate of the different algorithms in reaching the precision of 10^{-8} for different function groups and for all dimensions is shown. The table shows that the simple algorithm selector has a success rate of 100% on the separable functions, i.e. f_1-f_5 for all dimensions. It also demonstrates that the simple selector has a success rate better than the success rate of the individual algorithms on the unimodal functions with moderate conditioning, i.e. f_6–f_9, and the multimodal functions, i.e. f_{15}–f_{19}. However, the (1+1)-CMA-ES algorithm has a slightly better success rate than the simple algorithm selector on the unimodal ill-conditioned functions, i.e. f_{10}–f_{14}, and the multimodal functions with weak structure, i.e. f_{20}–f_{24}. The performance on the group of multimodal functions with weak structure can be explained with the shape of the fitness landscapes of this group. They do not enable the algorithm selector to benefit from utilising previous experience for deciding on the best algorithm for the current state. For the unimodal ill conditioned functions, selecting the (1+1)-ES with one fifth rule or the LSStep, which have a success rate of approximately 0, can lead to waste a high fraction of the search resources.

Table 5: The success rate of the algorithms in finding the ultimate precision of 10^{-8} for different function groups.

Algorithm	Function group				
	f_1-f_5	f_6-f_9	f_{10}-f_{14}	f_{15}-f_{19}	f_{20}-f_{24}
Selector	1.00	0.78	0.92	0.13	0.41
CMA	0.53	0.70	0.94	0.08	0.44
Simplex	0.56	0.50	0.59	0.03	0.35
OneFifth	0.39	0.31	0.00	0.10	0.41
LSStep	0.97	0.00	0.01	0.01	0.01

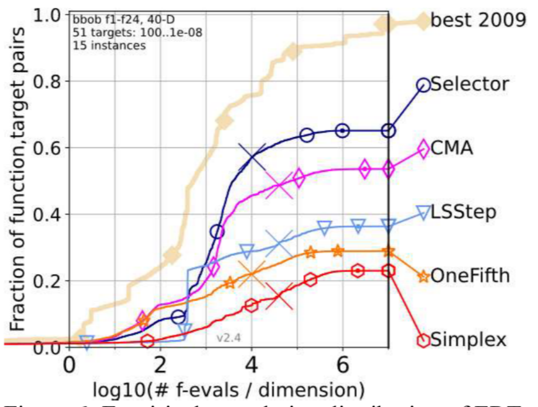

Figure 6: Empirical cumulative distribution of ERT over dimension for all functions in 40-D.

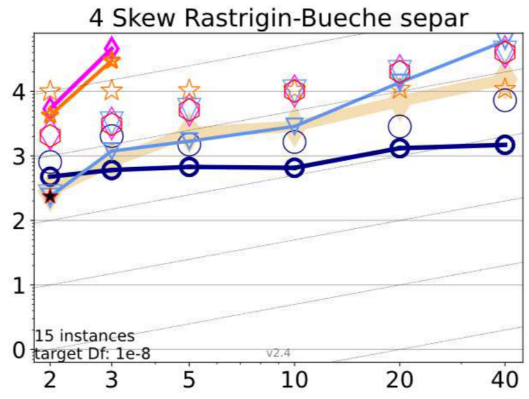

Figure 7: Expected running time (ERT in number of f−evaluations as log10 value), divided by dimension for target function value 10^{-8} versus dimension. Legend: ○: Selector, ◇: CMA, ☆: OneFifth, ▼: LSStep, ⬡: Simplex

CONCLUSION AND FUTURE WORK

A simple algorithm selector that differentiate between the performances on a set of four search algorithms is proposed. A portfolio, which unifies single point and multi-point search algorithms to enable utilising them within an algorithm selector, is constructed. A simple discrimination method is used to decide on a suitable search algorithm based on the search experience. The simple algorithm selector outperforms the stand-alone search algorithms on the noise-free BBOB-2009 test suite. It was able to solve f_1,f_2,f_3,f_4,f_5,f_8,f_9,f_{10},f_{12}, and f_{14} to the ultimate precision of 10^{-8} for dimensions of 2, 3, 5, 10, 20 and 40.

The next step in this research is to investigate reinforcement learning techniques, such as q-learning, to decide on the right algorithm based on the accumulated search experience

REFERENCES

Auger, A., 2009. Benchmarking the (1+1) evolution strategy with one-fifth success rule on the BBOB-2009 function testbed. Montréal Québec, Canad, s.n.

Auger, A. & Hansen, N., 2009. Benchmarking the (1+1)-CMA-ES on the BBOB-2009 Functions testbed. Montreal, Canada, s.n.

Belkhir, N., Dréo, J., Savéant, P. & Schoenauer, M., 2017. Per Instance Algorithm Configuration of CMA-ES with Limited Budget. Berlin, German, s.n.

Cruz-Reyes, L. et al., 2012. Algorithm Selection: From Meta-Learning to Hyper-Heuristics. In: Intelligent Systems. s.l.:InTech, pp. 77-102.

Cuevas, E., Fausto, F. & González, A., 2020. Metaheuristics and Swarm Methods: A Discussion on Their Performance and Applications. In: New Advancements in Swarm Algorithms: Operators and Applications. s.l.:Springer, Cham, pp. 43-67.

Drake, J., Kheiri, A., Özcan, E. & Burk, E., 2019. Recent advances in selection hyper-heuristics. European Journal of Operational research, pp. 1-24.

El-Mihoub, T., Hopgood, A. A. & Aref, I., 2014. Self-adaptive Hybrid Genetic Algorithm using an Ant-based Algorithm. Kuala lumpur, IEEE, pp. 166-170.

Hansen, N., 2006. The CMA Evolution Strategy: A Comparing Review. In: J. Lozano, P. Larrañaga, I. Inza & E. Bengoetxea, eds. Towards a New Evolutionary Computation. Studies in Fuzziness and Soft Computing. Berlin, Heidelberg: Springer, pp. 75-102.

Hansen, N., 2009. Benchmarking the Nelder-Mead Downhill Simplex Algorithm With Many Local Restarts. Montreal, Canda, ACM, p. 2403–2408.

Hansen, N., Auger, A. & Finck, S. ,. R., 2010. Real-parameter black-box optimization benchmarking BBOB-2010: Experimental setup, s.l.: INRIA.

Hansen, N. et al., 2021. COCO: a platform for comparing continuous optimizers in a black-box setting. Optimization Methods and Software, pp. 1-36.

Hansen, N., Finck, S., Ros, R. & Auger, A., 2009. Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions Technical Report RR-6829, s.l.: INRIA.

Huang, C., Li, Y. & Yao, X., 2019. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. IEEE Transactions on Evolutionary Computation, pp. 1-16.

Igel, C., Suttorp, T. & Hansen, N., 2006. A Computational Efficient Covariance Matrix Update and a (1+1)-CMA for Evolution Strategies. Seattle, Washington, USA, ACM.

Janković, A. & Doerr, C., 2019. Adaptive Landscape Analysis. Prague, Czech Republic, s.n., p. 2032–2035.

Kerschke, P., Hoos, H. H., Neumann, F. & Trautmann, H., 2018. Automated Algorithm Selection: Survey and Perspectives. Evolutionary Computation, 27(1), p. 3–45.

McKinnon, K., 1998. Convergence of the Nelder--Mead Simplex Method to a Nonstationary Point. SIAM Journal on Optimization, p. 148–158.

Miranda, P., Prudêncio, R. B. & Pappa, G. L., 2017. H3AD: A hybrid hyper-heuristic for algorithm design. Information Sciences, Volume 414, pp. 340-354.

Nelder, j. & Mead, R., 1965. A simplex method for function minimization. The Computer Journal, p. 308–313.

Pošík, P. & Huyer, W., 2012. Restarted Local Search Algorithms for Continuous Black-Box Optimization. Evolutionary Computation, 20(4), pp. 575-607.

Posík, P., 2009. BBOB-benchmarking two variants of the line-search algorithm. Montréal Québec, Canada, s.n.

Rice, J. R., 1976. The algorithm selection problem. Advances in Computers, Volume 15, pp. 65-118.

Schumer, M. & Steiglitz, K., 1968. Adaptive step size random search. EEE Transactions on Automatic Control, 13(3), pp. 270-276.

Swarzberg, S., Seront, G. & Bersini, H., 1994. S.T.E.P.: the easiest way to optimize a function. Orlando, FL, USA, IEEE, pp. 519-524 .

Vermetten, D., Wang, H., Doerr, C. & Bäck, T., 2020. Integrated vs. sequential approaches for selecting and tuning CMA-ES variants. Cancun, Mexico, s.n.

Whitley, D., 2019. Next Generation Genetic Algorithms: A User's Guide and Tutorial. In: Handbook of Metaheuristics. International Series in Operations Research & Management Science. s.l.:Springer, Cham, pp. 245-274.

Wolpert, D. H. & Macready, W. G., 1997. No Free Lunch Theorems for Optimization. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, pp. 67-82.