Evaluation of Parallel Hierarchical Differential Evolution for Min-Max Optimization Problems Using SciPy

Margarita Antoniou[1,2][0000−0002−0239−5857] and
Gregor Papa[1,2][0000−0002−0623−0865]

[1] Computer Systems Department, Jožef Stefan Institute, Jamova c. 39, Ljubljana, Slovenia
[2] Jožef Stefan International Postgraduate School, Jamova c. 39, Ljubljana, Slovenia
{margarita.antoniou,gregor.papa}@ijs.si

Abstract. When optimization is applied in real-world applications, optimal solutions that do not take into account uncertainty are of limited value, since changes or disturbances in the input data may reduce the quality of the solution. One way to find a robust solution and consider uncertainty is to formulate the problem as a min-max optimization problem. Min-max optimization aims to identify solutions which remain feasible and of good quality under even the worst possible scenarios, i.e., realizations of the uncertain data, formulating a nested problem. Employing hierarchical evolutionary algorithms to solve the problem requires numerous function evaluations. Nevertheless, Evolutionary Algorithms can be easily parallelized. This work investigates a parallel model for differential evolution using SciPy, to solve general unconstrained min-max problems. A differential evolution is applied for both the design and scenario space optimization. To reduce the computational cost, the design level optimization is parallelized. The performance of the algorithm is evaluated for a different number of cores and different dimensionality of four benchmark test functions. The results show that, when the right parameters of the algorithm are selected, the parallelization can be of high benefit to a nested differential evolution.

Keywords: Min-max optimization · Parallelization · Differential Evolution.

1 Introduction

Any kind of real-world optimization problem contains to a degree uncertainty in its data, be it by inherent stochasticity or due to errors. One way to consider this uncertainty and find a robust solution to the problem is to formulate it as a min-max optimization problem [8]. This formulation aims to find the best worst-case solution and hence the most robust.

Numerous approaches have been developed to solve the min-max optimization problem. Among the traditional ones -which require a specific mathematical formulation- one can find branch-and-bound algorithms [16] or approximation methods [2]. When such a formulation is not applicable, Evolutionary Algorithms (EA) can be used. One of the first such EAs can be found in [6, 7], where a genetic algorithm is used in a coevolutionary fashion, successfully solving min-max problems that hold specific conditions. Another EA approach is using the inherent hierarchical formulation, leading to nested algorithms and such a nested Particle Swarm Optimization algorithm can be found in [13]. What is common knowledge about the EAs though, is that they require numerous iterations and function evaluations, making the computational cost prohibitively high. A popular way to mitigate this problem is the use of surrogates, as suggested in [24], with some cost in the accuracy though. A surrogate-assisted min-max multifactorial EA is proposed in [23] employing evolutionary multitasking optimization and surrogate techniques. A min-max Differential Evolution (DE) is proposed in [18] with many improvements to the original DE, designed specially for min-max problems. Another nested approach was presented in [5], where a DE with an estimation of distribution algorithm is proposed and a priori knowledge of the previous generations is utilized to reduce the computational expense.

When dealing with populations, parallelism is inherent since each of the individuals who make up the population is an independent component [14]. There are numerous of studies that deal with different parallelization of EAs [1, 3, 20]. More specifically for DE -a popular EA that we use in this study- parallel models can be found in [12, 17, 21]. Most of them refer to EAs that deal with single level or multiobjective problems. Two parallel models of bilevel DE are suggested in [15], that reduced drastically the computational time in a number of test functions. There is a study of a co-evolutionary DE algorithm in C-CUDA for solving min-max problems in [10]. To the best of our knowledge, there is no study that researches a parallel model for a purely hierarchical (nested) DE for min-max problems.

One popular and user-friendly implementation of DE is in the SciPy python package [22]. It gives the users the option to use different workers and evaluate the population in parallel. Given its popularity and simplicity, it is a suitable choice to use this framework for this work. We apply a DE algorithm for both the design and scenario space, using the parallelization option for the design space. In this way, the design space population is evaluated in parallel, meaning that the second-level DE of the scenario space is run in parallel.

We then proceed to test the method in four benchmark test-functions with different properties, known from the literature. To test the scalability of the results, we scaled the test-functions up to 10 dimensions.

Our research questions are the following:

RQ 1: What kind of speedup do we achieve for different test-functions and different dimensionality of the problem?
RQ 2: Does the population size of the design space affect the speedup and how?
RQ 3: Does the DE mutation strategy of the design space affect the runtime?

In this study, we approach the first question and scratch the surface for the other two.

The organization of this paper is as follows. Section 2 defines the min-max optimization problem and Section 3 gives an overview of the Differential Evolution and its nested form for solving min-max problems and explains how it is parallelized in this work. The experimental setup, the test functions used, along with the performance of the method are described in Section 4. Finally, Section 5 summarizes the paper and gives some ideas for future work.

2 Definition of the Problem

The general unconstrained min-max problem can be described as:

$$\min_{x \in X} \max_{s \in S} f(x, s) \tag{1}$$

where x is a solution selected from search space X and s a scenario chosen from the scenario space S. The objective is to locate a solution $x* \in X$ that minimizes the worst-case objective $\max_{s \in S} f(x,s)$. The problem is considered symmetrical when the following condition is true

$$\min_{x \in X} \max_{s \in S} f(x,s) = \max_{s \in S} \min_{x \in X} f(x,s)$$

Symmetrical problems have independent feasible regions of the search and scenario space, making their solution more tractable.

3 Differential Evolution for MinMax Problems

3.1 Overview of Differential Evolution

The traditional definition of Differential Evolution can be found in [19]. Following the standard evolutionary algorithm schema, a population of candidate solutions undergoes the operations of mutation, crossover, and selection. There exist numerous strategies, that DE can apply to mutate the individuals through a difference mechanism. The most regularly used variant is to select two random individuals from the current population and add their scaled vector difference to the individual vector to be mutated (rand/1/bin). Another popular variant is best/1/bin, where as base vector the best vector in the population is used. The pseudocode of the DE algorithm can be seen in Algorithm 1.

Algorithm 1: Pseudocode of DE.

```
Input  : Population Size (NP), Max Generations (MaxGen), CR, F, Dimension D
Output: Optimal Solution
//Initialization
Generate $NP$ individuals randomly
while budget condition do
    for $k$ = 1 to $NP$ do
        calculate $fit(x_k)$
    end
    for $k$ = 1 to $NP$ do
        //mutation
        Generate three random indexes $r_1$, $r_2$ and $r_3$, where $r_1 \neq r_2 \neq r_3 \neq k$
        $V_k^G = X_{r_1}^G + F * (X_{r_2}^G - X_{r_3}^G)$ /* rand/1/bin */
        /* best/1/bin: $V_k^G = X_{best}^G + F * (X_{r_1}^G - X_{r_2}^G)$ */
        //crossover
        for $i$ = 1 to $n$ do
            if rand(0,1)< CR then
                $U[i] = V_k[i]$
            else
                $U[i] = X_k[i]$
            end
        end
    end
    //selection
    if $fit(U_k^G) \leq fit(X_k^G)$ then
        $X_k^{G+1} = U_k^G$
    end
end
```

3.2 Hierarchical (Nested) Differential Evolution and Parallel Model

The main steps of the nested algorithm can be seen in Figure 1. Each time a Design Space individual has to be evaluated, the algorithm runs a nested DE in order to find the optimal maximization solution in the scenario space. This can be seen with red dotted lines in the figure, where in the sequential manner, the algorithm has to wait till the nested DE is done to proceed and evaluate the next individual.

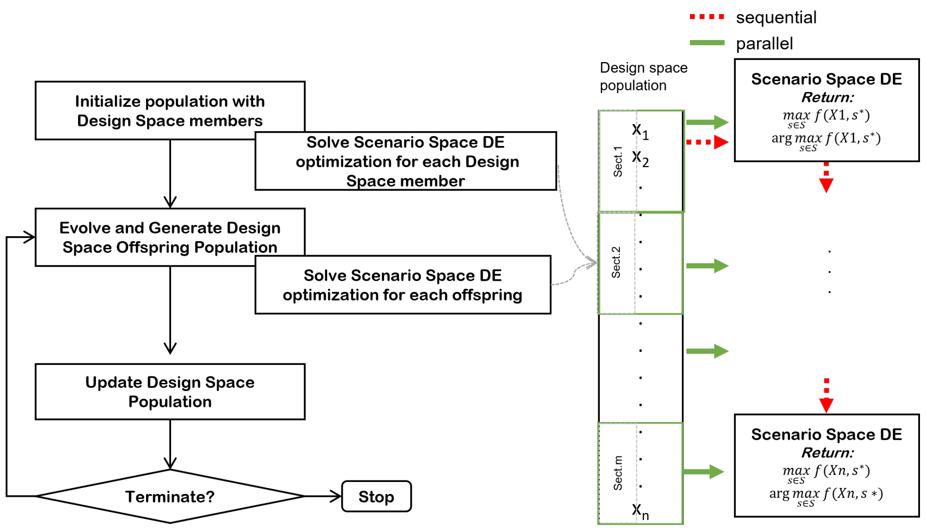

sequential
parallel
Design space population
Initialize population with Design Space members
Solve Scenario Space DE optimization for each Design Space member
Evolve and Generate Design Space Offspring Population
Solve Scenario Space DE optimization for each offspring
Update Design Space Population
Terminate?
Stop
Sect.1
Sect.2
Sect.m
x_1
x_2
x_n
Scenario Space DE
Return:
$\max_{s \in S} f(X1, s^*)$
$\arg\max_{s \in S} f(X1, s^*)$
Scenario Space DE
Return:
$\max_{s \in S} f(Xn, s^*)$
$\arg\max_{s \in S} f(Xn, s*)$

Fig. 1: Flowchart of nested Differential Evolution Algorithm (DE).

The model implemented in this work makes use of the workers argument in scipy.optimize.differential_evolution of the optimization package. The population is subdivided into workers sections and evaluated in parallel, by using Python multiprocessing.Pool [3]. It should be noted, that only the objective function evaluations are carried out in parallel, after the new population has evolved.

The model focuses on parallelization of the design space population, and it is similar to the parallel upper-level model for bilevel problems presented in [15] [4]. The population is subdivided into workers sections and evaluated in parallel. Denoting $Npop$ the design space population size and $nproc$ the number of processes, each section handles $Npop/nproc$ individuals. Each process handles a section of the evolved population and solves the scenario space problem with DE in order to evaluate the new individuals. In Figure 1, the procedure is shown with green lines, where the population is divided into sections and evaluated in parallel.

4 Experimental setup and Results

All cases have been independently run 20 times for each test instance on an Intel(R) Xeon(R) with 2 CPU E5-2680 v3 @ 2.50GHz that have 12 cores each and the Ubuntu 21.04 operating system. The algorithms are implemented in Python 3.7 (Python SciPy library [22]), using the parallelization of differential evolution for SciPy. The relevant code can be found in [4].

[3] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html

[4] The specific model uses the traditional synchronous parallelization of the DE, where also the operators are applied in parallel to produce the population.

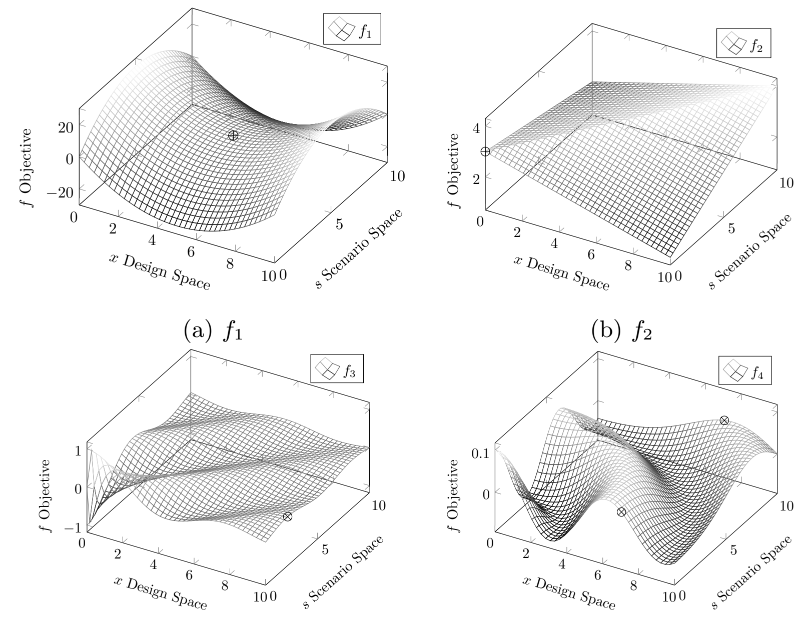
4.1 Benchmark Test Functions

The performance of the proposed algorithm is tested on 4 benchmark problems of min-max optimization, as found in [11]. The functions are the following:

Test function f_1 :

$$\min_{x\in X}\max_{s\in S} f_1(x,s) = (x_1-5)^2 - (s_1-5)^2 \tag{2}$$

with $x \in [0,10], s \in [0,10]$. The known solution is $x^* = 5$ and $s^* = 5$ with an optimal value of $f_1(x^*, s^*) = 0$. This test function is a saddle point function. The function along with the known optimum is plotted in Fig.2a and it serves as an example of a symmetric function.



(a) f_1 (b) f_2

(c) f_3 (d) f_4

Fig. 2: 3D mesh plots of the test functions. The black circle corresponds to the known optimum.

Test function f_2:

$$\min_{x\in X}\max_{s\in S} f_2(x,s) = \min\{3 - 0.2x_1 + 0.3s_1, 3 + 0.2x_1 - 0.1s_1\} \tag{3}$$

with $x \in [0,10], s \in [0,10]$. The optimal points are $x^* = 0$ and $s^* = 0$ and the optimal value is approximated at $f_2(x^*, s^*) = 3$. It is a two-plane asymmetrical function. The 3-D plot of this function, along with the known optima, are shown in Fig. 2b.

Test function f_3:

$$\min_{x\in X}\max_{s\in S} f_3(x,s) = \frac{\sin(x_1 - s_1)}{\sqrt{x_1^2 + s_1^2}} \tag{4}$$

with $x \in [0,10], s \in [0,10]$. The known solution is $x^* = 10$ and $s^* = 2.1257$ with an optimal value of $f_3(x^*, s^*) = 0.097794$. It is a damped sinus asymmetrical function, as shown in Fig. 2c.

Test function f_4:

$$\min_{x\in X}\max_{s\in S} f_4(x,s) = \frac{\cos\left(\sqrt{x_1^2+s_1^2}\right)}{\sqrt{x_1^2+s_1^2}+10} \tag{5}$$

with $x \in [0,10], s \in [0,10]$. The known optimal solutions are $x^* = 7.0441$ and $s^* = 10$ or $s^* = 0$ and the optimal value is $f_4(x^*, s^*) = 0.042488$. It is a damped cosine wave asymmetrical function, as shown in Fig. 2d

To further evaluate the performance, the 4 test functions are modified to be scalable as follows

$$\min_{x\in X}\max_{s\in S} f_1(x,s) = \sum_{n=1}^{D}(x_n-5)^2-(s_n-5)^2 \tag{6}$$

$$\min_{x\in X}\max_{s\in S} f_2(x,s) = \min\{\sum_{n=1}^{D} 3-0.2x_n+0.3s_n, \sum_{n=1}^{D} 3+0.2x_n-0.1s_n\} \tag{7}$$

$$\min_{x\in X}\max_{s\in S} f_3(x,s) = \frac{\sum_{n=1}^{D}\sin(x_n-s_n)}{\sum_{n=1}^{D}\sqrt{x_n^2+s_n^2}} \tag{8}$$

$$\min_{x\in X}\max_{s\in S} f_4(x,s) = \frac{\sum_{n=1}^{D}\cos\left(\sqrt{x_n^2+s_n^2}\right)}{\sum_{n=1}^{D}\sqrt{x_n^2+s_n^2}+10} \tag{9}$$

where D is the dimensionality of the problem and with $x \in [0,10]^D, s \in [0,10]^D$. The first two problems can also be found in [18], while the other two are scaled for the first time. For all the instances we test the following dimensionality: D=[1,2,5,10]. The dimensionality is scaled for both the design and scenario space.

4.2 Parameter Settings

The control parameter values used are reported in Table 1, unless stated otherwise. We kept the default SciPy crossover, mutation, and strategy values. The values of population and generation size are selected to be close to the ones in similar experiments done in [9]. Note that to reach better accuracy for the higher dimensionality, a larger number of population and/or generation sizes are needed. Nevertheless, we kept the sizes the same for reasons of computational budget and clear comparison in terms of running times. It is -in any case- not in the scope of this paper to examine the accuracy.

4.3 Results and Discussion

4.3.1 Case study 1: Evaluation of different test functions and dimensionality

In Tables 2, 3, 4 and 5 the statistical results for the test-functions are reported. More specifically, we report the median and standard deviation of the runtime in seconds and the speedup. The speedup measures the ratio between the sequential and the parallel execution times (runtime).

In Figure 3 the runtime and the speedup curve for each test function and instance with respect to the different number of cores used are depicted. More specifically, the Figures 3b, 3d,3f and 3h

Table 1: Selected control parameters.

	Design Space	Scenario Space
Crossover rate	0.7	0.7
Mutation rate	U(0.5, 1)	U(0.5, 1)
Population size	20	10
Number of generations	5	10
Strategy	best1bin	best1bin

show the speedup curve for the test functions under analysis and the different dimensionality. It is noticed that for f_1,f_3 and f_4 the speedup considering up to 16 processors is increasing for each dimensionality. Same for f_2, but only for 2/2D,5/5D, and 10/10D. A different behavior is spotted for f_2 and 1/1D, as already for 2 cores, the speedup is decreasing, only to start improving again till we reach the 8 cores. For all test functions, for the 1 dimension case, the speedup decreases after the 16 processors. In general, the speedup increases with the increase of the dimension as it is expected, since calculating the objective function is more expensive. This can also be seen in Figures 3a,3c,3e and 3g showing the running times in seconds for each function and each dimensionality with respect to the number of cores used. It is clear the required running times are also "scaled" to the dimension for all the test functions.

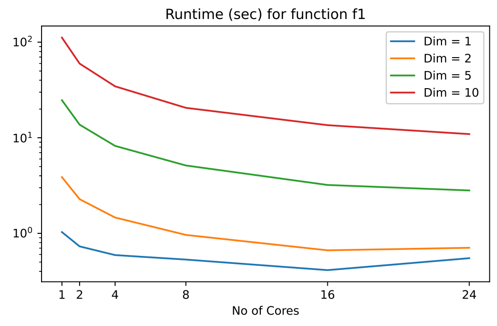
For the higher dimension of the problems, meaning 5/5D and 10/10D, the speedup is always increasing, except for f_4 and 10/10D, which decreases slightly for 24 cores. It is worth noting that for 5/5D and 10/10D dimensionality, there is almost a 10 times speedup when using 24 processors for f_1. For example, in the sequential case for f_1 and 10/10D, the algorithm needs 111.7 seconds or almost 2 minutes to run, while after the parallelization needs only 10 seconds. For the rest of the test functions the speedup for 5/5D and 10/10D ranges around 4-5. As an example, for f_4 and 10/10D, the sequential algorithm needs around 260 seconds or almost 4 minutes to run, while after the parallelization needs almost 1 minute. This indicates the positive effects of parallelization on the min-max problem especially when the dimension is higher.

For f_3 and f_4, 5/5D shows greater improvement with respect to the number of cores used than 10/10D. These results are very close though, and more experiments and sample runs are needed to correct the accuracy. For lower dimensions, especially for 1/1D, the performance drops and the speedup is worsening with the number of cores.

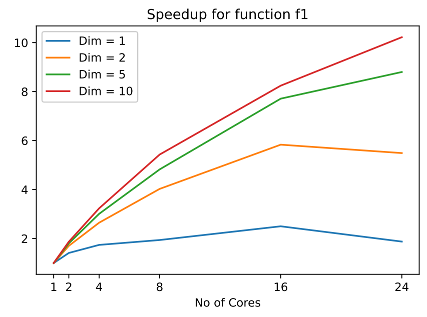
To show that the parallelization has little effect on the accuracy of the results found, in Tables 2, 3, 4 and 5 we report the median accuracy in % for all the test functions and instances. We calculate the error rate as the absolute differences between the best objective function values provided by the algorithm and the known global optimal objective values of each test function. The accuracy formula provides accuracy as a difference of error rate from 100%. This is expressed as

$$Acc\% = 100 - |f^{'} - f^{*}|{*}100/f^{*} \tag{10}$$

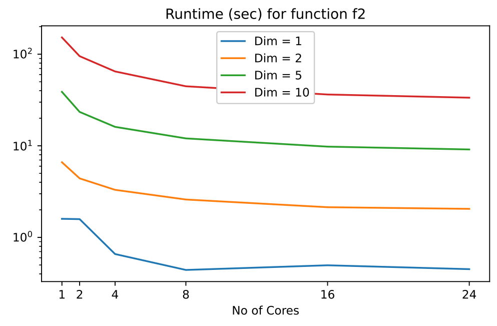
where $f^{'}$ and f^{*} are the best found and the true optimal values, respectively. As is expected, the accuracy achieved in all cases does not significantly fluctuate among different cores. The accuracy for f_4 and for 10/10D is in general lower (almost half). Asymmetrical test functions constitute a more complicated problem and a different parametrization of the DE might be needed (eg. larger population size, more generations, etc.) Nevertheless, the algorithm yields to the known global optima in the other cases, meaning the nested approach can solve both symmetrical and asymmetrical problems.



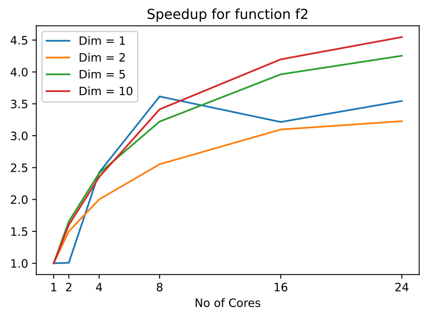
(a) Median Runtime of the test function f_1



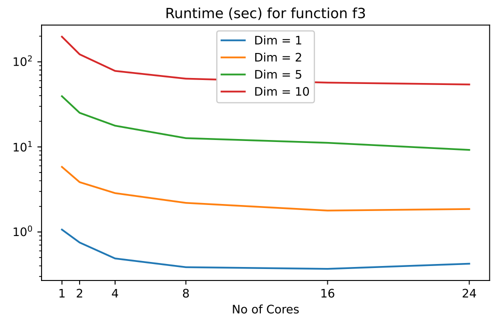
(b) Speedup of the test function f_1



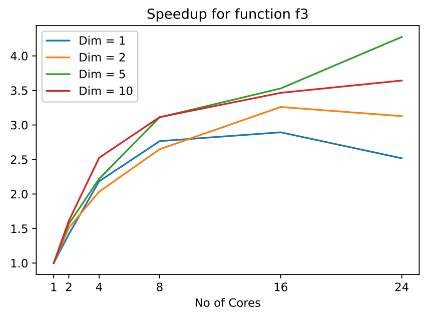
(c) Median Runtime of the test function f_2



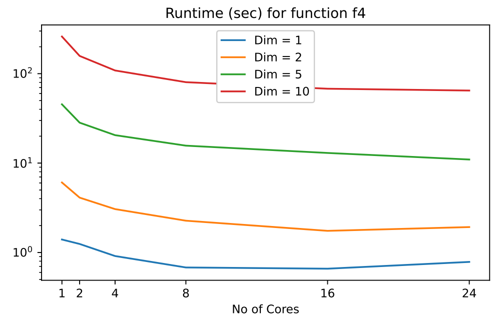
(d) Speedup of the test function f_2



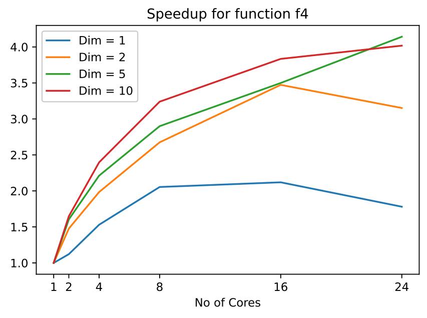
(e) Median Runtime of the test function f_3



(f) Speedup of the test function f_3



(g) Median Runtime of the test function f_4



(h) Speedup of the test function f_4

Fig. 3: Runtime and speedup plots of the test-functions.

Table 2: Running Time Results in seconds and Median Accuracy for the f_1.

Dimension	Cores	Median	Std.dev.	Speedup	Median Acc %
1	1	1.030820	0.104606	-	99.99
	2	0.730346	0.124426	1.411412	99.99
	4	0.592372	0.097924	1.740156	99.99
	8	0.531529	0.116901	1.939347	99.99
	16	0.412243	0.082602	2.500514	99.99
	24	0.549554	0.104442	1.875738	99.99
2	1	3.876020	0.038887	-	100.0
	2	2.277507	0.044546	1.701870	100.0
	4	1.464805	0.048612	2.646099	100.0
	8	0.962618	0.189352	4.026541	100.0
	16	0.664449	0.039447	5.833436	100.0
	24	0.706010	0.086775	5.490039	100.0
5	1	24.739887	0.136860	-	99.99
	2	13.712247	0.146743	1.804218	99.99
	4	8.226063	0.114659	3.007500	99.99
	8	5.129729	0.109766	4.822844	100.0
	16	3.207961	0.146968	7.712028	99.99
	24	2.811062	0.128049	8.800903	99.99
10	1	111.715032	1.033029	-	99.99
	2	59.729019	0.520408	1.870364	99.99
	4	34.562721	0.520408	3.232241	99.99
	8	20.576803	0.492896	5.429174	99.99
	16	13.552745	0.520146	8.242982	99.99
	24	10.929520	0.637464	10.221403	99.99

Table 3: Running Time Results in seconds and Median Accuracy for the f_2.

Dimension	Cores	Median	Std.dev.	Speedup	Median Acc %
1	1	1.596429	0.204669	-	100.0
	2	1.584016	0.300120	1.007836	99.96
	4	0.658811	0.278364	2.423197	100.0
	8	0.441545	0.269987	3.615554	100.0
	16	0.496514	0.329453	3.215277	99.99
	24	0.450400	0.268212	3.544468	100.0
2	1	6.620913	1.047487	-	98.88
	2	4.412415	0.717296	1.500519	98.90
	4	3.308857	0.868970	2.000967	98.76
	8	2.593604	0.842654	2.552785	98.76
	16	2.137756	0.640007	3.097133	99.17
	24	2.051248	0.961823	3.227749	99.21
5	1	38.835937	2.556689	-	85.28
	2	23.445822	3.648188	1.656412	83.84
	4	16.113869	1.323346	2.410094	85.42
	8	12.053100	1.986224	3.222070	85.08
	16	9.801519	1.462326	3.962236	85.07
	24	9.129921	2.965911	4.253699	83.98
10	1	152.548321	3.390158	-	39.78
	2	95.240152	9.687155	1.601723	42.61
	4	64.895123	10.153455	2.350690	41.13
	8	44.688494	5.504860	3.413593	44.40
	16	36.344648	7.485144	4.197271	41.55
	24	33.554359	5.100844	4.546304	45.15

Table 4: Running Time Results in seconds and Median Accuracy for the f_3.

Dimension	Cores	Median	Std.dev.	Speedup	Median Acc %
1	1	1.066014	0.111083	-	99.99
	2	0.751404	0.059305	1.418697	99.99
	4	0.488195	0.061816	2.183584	99.99
	8	0.385446	0.087810	2.765667	99.99
	16	0.368403	0.063252	2.893611	99.99
	24	0.423447	0.14840	2.517468	99.97
2	1	5.818274	0.672975	-	96.86
	2	3.851147	0.495885	1.510790	98.08
	4	2.861794	0.544807	2.033086	97.97
	8	2.196644	0.623258	2.648710	97.12
	16	1.784786	0.588991	3.259928	97.14
	24	1.860299	0.584155	3.127601	97.00
5	1	39.398736	1.696303	-	98.20
	2	25.141406	1.523187	1.567086	97.97
	4	17.753222	1.242934	2.219244	97.68
	8	12.665715	1.918927	3.110660	98.52
	16	11.165424	1.870048	3.528638	97.44
	24	9.216848	1.711269	4.274643	97.08
10	1	197.311041	12.023453	-	97.91
	2	122.471174	8.766168	1.611081	97.60
	4	78.167577	10.112983	2.524206	97.87
	8	63.370734	5.421352	3.113599	97.41
	16	56.940333	7.925534	3.465225	98.21
	24	54.158177	7.496917	3.643236	97.64

Table 5: Running Time Results in seconds and Median Accuracy for the f_4.

Dimension	Cores	Median	Std.dev.	Speedup	Median Acc %
1	1	1.398317	0.171784	-	94.14
	2	1.246892	0.255854	1.121442	93.94
	4	0.913910	0.189214	1.530038	97.63
	8	0.680622	0.194613	2.054470	97.19
	16	0.659713	0.161889	2.119585	97.53
	24	0.785280	0.204446	1.780661	97.86
2	1	6.071163	0.607564	-	96.89
	2	4.111252	0.752849	1.476719	97.67
	4	3.060104	0.416516	1.983972	97.57
	8	2.269734	0.754214	2.674834	98.52
	16	1.748137	0.437059	3.472932	97.68
	24	1.926225	0.412750	3.151845	97.31
5	1	45.444606	2.641320	-	96.83
	2	28.333389	3.517377	1.603924	98.03
	4	20.544451	2.007804	2.212014	97.92
	8	15.674804	2.192337	2.899214	97.79
	16	12.987283	2.854605	3.499162	96.18
	24	10.972779	2.310476	4.141577	97.86
10	1	260.356936	14.329717	-	94.35
	2	157.981917	11.599310	1.648017	93.44
	4	108.636768	14.879940	2.396582	95.48
	8	80.369076	10.209701	3.239516	95.98
	16	67.907552	7.578134	3.833991	94.72
	24	64.801424	8.651000	4.017766	96.68

4.3.2 Case study 2: Influence of the population size

To evaluate the effect of the design space population size on the speedup of the algorithm, test function f_1 with 1/1D is run for different population sizes for 20 independent runs. The population size = 4,8,16,24,48 were selected to match the number of cores, as the design space population is the one that is divided into a number of processes and then calculated in parallel and might give some insight. Also, the population sizes of 5,10,20, and 30 are tested, as they are more commonly selected values.

In Figure 4 the speedup is reported. Also, in Figure 5 the stacked bar charts of speedup for the different population sizes and the number of cores are shown. In this graph, each value of the different population size speedup is placed after the previous one and the total value of each bar is all the segment values added together. As is expected, the higher the population size, the higher the speedup achieved by adding cores. Moreover, in most cases, as the number of cores exceeds the population size, the speedup is worsened. This can be specifically noted for $npop = 8$ and the number of $cores > 8$, as well as for $npop = 20$ and the number of $cores > 16$. The combination of the number of cores and the population size of the design space is affecting the speedup of the parallel model. There is a significant decrease in the speedup for $npop = 48$ and $cores = 24$, which might be due to the communication costs. More experiments along with profiling tools are needed to showcase the exact influence of selecting population size in analogy to the available cores, especially to higher dimensionality and larger population sizes, which are expected to show larger differences in the runtimes and are the cases that will be most benefit by parallelization.

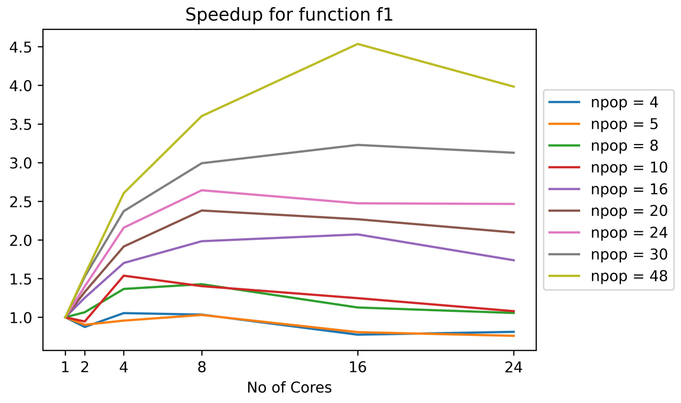


Fig. 4: Speedup plots of the function f_1 for dim=1 and different design space population size.

4.3.3 Case study 3: Influence of the strategy

Since the implementation of the parallelization is synchronous, it would be interesting to see if there is any influence of the design space strategy. Therefore, we tested the algorithm by changing the strategy to rand1bin - the most commonly used. The runs refer to function f_1 and 1/1D. In Figure 6 the speedup bar chart is shown for the different strategies and dimensionality with respect to the number of cores. In the cases that there is a difference, rand1bin shows a larger speedup. There is an inherent sequential operation of finding the best individual of the current generation,

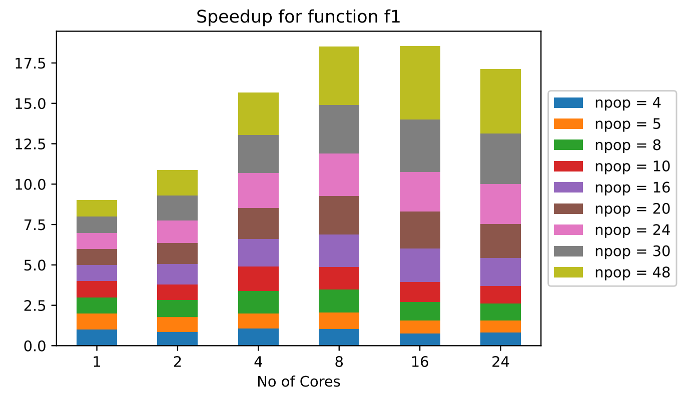


Fig. 5: Speedup stacked bar charts of the function f_1 for dim=1 and different design space population size (*npop*).

and this result agrees with what was noted also in [10]. More experiments are needed to reach safe conclusions and are in our future steps.

5 Conclusion and Future Work

In this work, the parallel model for solving min-max optimization problems via the user-friendly Python SciPy library was evaluated. The approach was tested for a nested DE and on four test functions from the literature. The four test functions were symmetrical and asymmetrical to test the behavior of the model on different min-max problems. Moreover, the problems were scaled to test the effect of the model also on different dimensions of the problems. In addition, a first insight was given about the effect of the population size and the strategy used on the speedup. The results show that the model is drastically reducing the computational time when the correct combination of the number of cores and population size of the design space is selected. The results especially indicated the large decrease of computational time on problems of higher dimensionality and when a larger population size is needed. This can motivate research of large-scale min-max problems via metaheuristics.

As noted above, the effect of the population size and the strategy on the speedup is our ongoing research. Though SciPy is used in this work, other more sophisticated parallel frameworks, such as CUDA with multi-CPU and multi-GPU can be used to further take advantage of the natural parallelization of DE of both levels. Finally, it would be interesting to test the current implementation on engineering applications and real-world problems.

Acknowledgements

This work was supported by the European Commission's H2020 program under the Marie Skłodowska-Curie grant agreement No. 722734 (UTOPIAE) and by the Slovenian Research Agency (research core funding No. P2-0098). The authors would like to thank Peter Korošec and Gašper Petelin for interesting discussions regarding this work.

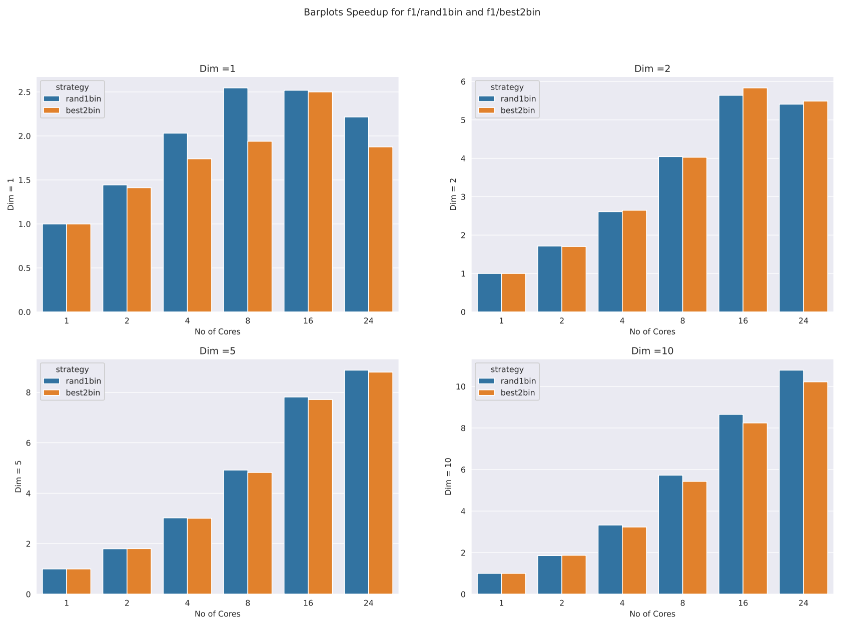


Fig. 6: Speedup bar charts of the function f_1 for dim=1 and different strategy.

References

1. Adamidis, P.: Parallel evolutionary algorithms: A review. In: Proceedings of the 4th Hellenic-European Conference on Computer Mathematics and its Applications (HERCMA 1998), Athens, Greece. Citeseer (1998)
2. Aissi, H., Bazgan, C., Vanderpooten, D.: Min–max and min–max regret versions of combinatorial optimization problems: A survey. European journal of operational research **197**(2), 427–438 (2009)
3. Alba, E.: Parallel evolutionary algorithms can achieve super-linear performance. Information Processing Letters **82**(1), 7–13 (2002)
4. Antoniou, M.: https://github.com/MargAnt0/ParallelMinMax_Scipy (2022)
5. Antoniou, M., Papa, G.: Differential evolution with estimation of distribution for worst-case scenario optimization. Mathematics **9**(17), 2137 (2021)
6. Barbosa, H.J.: A genetic algorithm for min-max problems. In: Goodman, editors, Proceedings of the First International Conference on Evolutionary Computation and Its Applications. pp. 99–109 (1996)
7. Barbosa, H.J.: A coevolutionary genetic algorithm for constrained optimization. In: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406). vol. 3, pp. 1605–1611. IEEE (1999)
8. Beyer, H.G., Sendhoff, B.: Robust optimization–a comprehensive survey. Computer methods in applied mechanics and engineering **196**(33-34), 3190–3218 (2007)
9. Cramer, A.M., Sudhoff, S.D., Zivi, E.L.: Evolutionary algorithms for minimax problems in robust design. IEEE Transactions on Evolutionary Computation **13**(2), 444–453 (2008)
10. Fabris, F., Krohling, R.A.: A co-evolutionary differential evolution algorithm for solving min–max optimization problems implemented on gpu using c-cuda. Expert Systems with Applications **39**(12), 10324–10333 (2012)

11. Jensen, M.T.: A new look at solving minimax problems with coevolutionary genetic algorithms. In: Metaheuristics: computer decision-making, pp. 369–384. Springer (2003)
12. Kozlov, K., Samsonov, A.: New migration scheme for parallel differential evolution. In: Proceedings of the international conference on bioinformatics of genome regulation and structure. pp. 141–144 (2006)
13. Laskari, E.C., Parsopoulos, K.E., Vrahatis, M.N.: Particle swarm optimization for minimax problems. In: Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600). vol. 2, pp. 1576–1581. IEEE (2002)
14. Luong, T.V., Melab, N., Talbi, E.G.: Gpu-based island model for evolutionary algorithms. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. p. 1089–1096. GECCO '10, Association for Computing Machinery, New York, NY, USA (2010). https://doi.org/10.1145/1830483.1830685, https://doi.org/10.1145/1830483.1830685
15. Magalhães, T., Krempser, E., Barbosa, H.: Parallel models of differential evolution for bilevel programming. In: Proceedings of the 2018 International Conference on Artificial Intelligence (2018)
16. Montemanni, R., Gambardella, L.M., Donati, A.V.: A branch and bound algorithm for the robust shortest path problem with interval data. Operations Research Letters **32**(3), 225–232 (2004)
17. Pedroso, D.M., Bonyadi, M.R., Gallagher, M.: Parallel evolutionary algorithm for single and multi-objective optimisation: Differential evolution and constraints handling. Applied Soft Computing **61**, 995–1012 (2017)
18. Qiu, X., Xu, J.X., Xu, Y., Tan, K.C.: A new differential evolution algorithm for minimax optimization in robust design. IEEE transactions on cybernetics **48**(5), 1355–1368 (2017)
19. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization **11**(4), 341–359 (1997)
20. Sudholt, D.: Parallel evolutionary algorithms. In: Springer Handbook of Computational Intelligence, pp. 929–959. Springer (2015)
21. Tasoulis, D.K., Pavlidis, N.G., Plagianakos, V.P., Vrahatis, M.N.: Parallel differential evolution. In: Proceedings of the 2004 congress on evolutionary computation (IEEE Cat. No. 04TH8753). vol. 2, pp. 2023–2029. IEEE (2004)
22. Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2
23. Wang, H., Feng, L., Jin, Y., Doherty, J.: Surrogate-assisted evolutionary multitasking for expensive minimax optimization in multiple scenarios. IEEE Computational Intelligence Magazine **16**(1), 34–48 (2021)
24. Zhou, A., Zhang, Q.: A surrogate-assisted evolutionary algorithm for minimax optimization. In: IEEE Congress on Evolutionary Computation. pp. 1–7. IEEE (2010)