



Iterative optimisation - When a best algorithm does exist and other frustrating theorems

Maurice Clerc

► To cite this version:

Maurice Clerc. Iterative optimisation - When a best algorithm does exist and other frustrating theorems. 2021. hal-03230263

HAL Id: hal-03230263

<https://hal.science/hal-03230263>

Preprint submitted on 19 May 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ITERATIVE OPTIMISATION - WHEN A BEST ALGORITHM DOES EXIST AND OTHER FRUSTRATING THEOREMS

MAURICE CLERC

ABSTRACT. The No Free Lunch does not hold for some sets of continuous functions or that are not closed under permutations (not-c.u.p.) [1, 3]. On a computer everything is discrete finite so we consider more carefully not-c.u.p. sets of functions. It appears that except for very particular cases, not only the NFL does not hold, not only there is a best algorithm in average, but that an 'ultimate' algorithm does exist. However, when you compare some algorithms on a given benchmark, an algorithm that is the best can be the worst on another one.

1. INTRODUCTION

When we are looking for the minimum of a function but we know absolutely nothing about this function, except of course how to compute its value on each point of a search space, it is easy to prove that the first point to sample should be the barycenter of this search space. It maximises the probability of success, even if it is usually very low after just one sampling. Usually, but not always, and we see published papers that propose new algorithms using a benchmark of 'center-biased' test functions. In such a case, and when the algorithm is itself 'center-biased', even not explicitly, it is pretty good on this benchmark, but this is a very particular case, and, in practice, a good benchmark should not contain such functions.

So, an interesting question is: what should be the structure of a benchmark so that there exists on it a 'ultimate' or, at least a 'partial-ultimate' algorithm. We give formal definitions of these properties and, as we will see, the conditions are very weak in practice, meaning they hold for all classical benchmarks, which is a good point. However, unfortunately, proving that something exists is different from finding it, but at least it means it is worth trying to improve algorithms again and again, looking for this grail, even if it can also be proved that an algorithm that is good on some problems can be very bad on some other ones.

2. WARM UP

Let us consider the definition space

$$X = \{1, 2, 3\}$$

and the value space

$$Y = \{0, 1\}$$

This is obvious on a digital computer, but still true for an analogical one, even if the 'granularity' is far smaller, according to the quantum theory.

TABLE 1. The eight functions of the 'Warm Up' example.

Function	Values on X
f_1	(0,0,0)
f_2	(1,0,0)
f_3	(0,1,0)
f_4	(0,0,1)
f_5	(1,1,0)
f_6	(1,0,1)
f_7	(0,1,1)
f_8	(1,1,1)

TABLE 2. The six algorithms of the 'Warm Up' example.

Algorithm	Sequence
a_1	(1,2,3)
a_2	(2,1,3)
a_3	(1,3,2)
a_4	(2,3,1)
a_5	(3,2,1)
a_6	(3,1,2)

There are 2^3 possible functions $X \rightarrow Y$, defined by their values on X , as we can see on the table 1.

We call F the set of these functions.

Now we define an iterative search algorithm as a sequence of sampled points¹, *without repetition*, which can also be seen as their ranks in X . So they are the permutations of the positions, and we have $3!$ possible algorithms, detailed in the table 2.

We call A the set of these algorithms.

Finally, we consider the benchmark $B = \{f_2, f_3, f_4, f_6, f_7\}$. As for many artificial benchmarks, we do suppose we know the minimum value of all problems of the benchmark, here zero. For each pair (a_i, f_j) , we count the smallest number $s_{i,j}$ of samplings that are needed to find this minimum. For easier further comparisons, we compute in fact a rate in $[0, 1]$ and call it *efficiency*:

$$(2.1) \quad E_{a_i, f_j} = \frac{|X| - s_{i,j}}{|X| - 1}$$

So, if a solution is found at the very first sampling, the efficiency is 1, and for exhaustive search it is zero.

¹If you are a practitioner you probably see an algorithm as a process. You just have to imagine that the definition given here is the result of a run of such a process.

TABLE 3. Number of needed sampled points and efficiency table.
On the benchmark B not all algorithms are equivalent. a_1 and a_2 are the best ones in average, but not the best on all functions. However, if you keep just f_2 , f_4 , and f_6 , then a_2 and a_4 are better on all these functions.

	a_1	a_2	a_3	a_4	a_5	a_6
f_2	2	1	2	1	1	1
f_3	1	2	1	2	1	1
f_4	1	1	1	1	2	2
f_6	2	1	3	1	2	3
f_7	1	2	1	3	3	2
Total	7	7	8	8	9	9

	a_1	a_2	a_3	a_4	a_5	a_6
f_2	0.5	1	0.5	1	1	1
f_3	1	0.5	1	0.5	1	1
f_4	1	1	1	1	0.5	0.5
f_6	0.5	1	0	1	0.5	0
f_7	1	0.5	1	0	0	0.5
Mean	0.8	0.8	0.603	0.603	0.6	0.6

	a_1	a_2	a_3	a_4	a_5	a_6
f_2	0.5	1	0.5	1	1	1
f_4	1	1	1	1	0.5	0.5
f_6	0.5	1	0	1	0.5	0
Mean	0.67	1.0	0.5	1.0	0.67	0.5

We can then see on the efficiency table 3 that not all algorithms are equivalent on our benchmark B . This is of course for B is not closed under permutation (c.u.p.)² [2], and therefore, the NFLT [4] is not applicable³.

Definitions, relatively to a given benchmark.

- *complete-best*: an algorithm that is, in average, equivalent or better than all the possible ones, and strictly better than at least one. If the benchmark is c.u.p. such an algorithm does not exist.
- *partial-best*(A), or simply *best*(A): an algorithm that is, in average, equivalent or better than all the ones of a subset A , and strictly better than at least one of this subset. This is the usual definition used when comparing algorithms.
- *complete-ultimate*: an algorithm that is equivalent or better than all the possible ones, on all functions of the benchmark, and strictly better on at least one function. Can exist only for some special benchmarks.
- *partial-ultimate*(A): an algorithm that is equivalent or better than all the ones of a subset A of all possible ones, on all functions of the benchmark, and strictly better on at least one function. In practice this is the most interesting property.

So, according to the table 3:

- a_1 and a_2 are complete-best (but not complete-ultimate) for the benchmark $\{f_2, f_3, f_4, f_6, f_7\}$.

²But $B' = B \cup \{f_5\}$ is c.u.p., although it is just a subset of all possible functions. On B' , all algorithms have the same mean efficiency $2/3$. On all the eight possible functions, the common mean efficiency would be $5/8$, a bit smaller. It is greater than 0.5 for some functions have several global minima.

³On too many published papers one can read something like 'On this benchmark our new algorithm outperforms the others on most problems, but not on all, because of the NFLT'. As, in practice the classical benchmarks are never c.u.p., such a claim is not valid.

- a_2 and a_4 are complete-ultimate for the benchmark $\{f_2, f_4, f_6\}$.
- a_1 is partial-ultimate for the benchmark $\{f_2, f_3, f_4, f_6, f_7\}$ and the set of algorithms $\{a_1, a_3\}$.

This very simple example is sufficient to prove the following theorem :

Theorem 1. *On some not-c.u.p. benchmarks of discrete finite functions, there may exist a complete-best algorithm, and even a complete-ultimate one*

But this is too vague, and we will now try to be more precise.

You may have noted that in this example the number of test functions (5) of the benchmark is greater than the number of possible positions in the search space (3). There exist complete-best algorithms, but we have complete-ultimate algorithms only if we reduce the benchmark so that its size is at most equal to the one of the search space. In practice this is of course always the case, and, moreover, we never consider all possible algorithms. So we are mainly interested on partial-ultimate algorithms: under which conditions do they exist?

3. GENERALISATION

On a computer, the number N of possible distinct values is finite, for example 2^{64} . For a search space of dimension D , the different positions X can be numbered from 1 to $|X| = N^D$. The number of possible algorithms is $|A| = N^D!$ and the number of possible functions is $|F| = N^{ND}$ (because on each position there are N possible values)⁴. Knowing these values is not really useful for what follows, but having an idea of how big they are in practice is interesting, compared to how small are the classical benchmarks and the number of search methods that are usually compared to each other, even if you run them many times. And, therefore, how risky is a conclusion about the non-existence of a partial-ultimate algorithm without any formal proof, for, when N and D increase, the number of algorithms increases far quicker than the number of functions (for $D > 1$).

3.1. An easy particular case. By definition our benchmark is a subset B of F . For each function $f_j \in B$ there exists at least a position (a rank) in X on which the function reaches its minimum. Note that, we sometimes do not know when it happens, and stop the algorithm say after a fixed number of samplings and evaluations that is always smaller than the size of the search space, so a partial-ultimate algorithm may exist but we are not able to say which one for sure.

A *sufficient* structure is the following one:

Condition 1: For all $f_j \in B$ the position of the minimum (i.e. its rank x_ in X) is the same.*

It implies that B is not-c.u.p., and in such a case, any algorithm that samples first the position x_* is ultimate ■

It looks too easy? It is. But, as said, such benchmarks are indeed used in some published papers.

⁴For simplicity we assume all functions have the same search space, which is always possible by normalisation.

3.2. A more general case. Now let us consider a better benchmark in which not all functions have the same solution point. Let us prove first that under another weak condition there exists at least one complete-best algorithm. More precisely, if $B = \{f_1, f_2, \dots, f_M\}$ we assume this:

Condition 2: There exists a point x_0 of X that is not solution of any f_i of B

Again, it implies that B is not closed under permutations. Also, it is just an easy and simple *sufficient* condition. It does not hold for the example of the table 3 but nevertheless there are two best algorithms for the benchmark $\{f_2, f_3, f_4, f_6, f_7\}$. However, it always hold in practice.

Theorem 2. *Under the Condition 2, there exists a complete-best algorithm*

The positions of the minimums of the functions in B , more precisely their first ranks in X , are $\{x_1, x_2, \dots, x_M\}$. It implies that the number of test functions must be at most equal to the number of points of the search space, but this is in practice always largely true.

Let us define $X' = \{x_1, \dots, x_{M'}\}$, with $M' \leq M$, the set of the different solutions points for the functions of B . And we consider the algorithm a_* whose definition sequence precisely begins with $(x_1, x_2, \dots, x_{M'})$. We do not know how to find it, but it exists for sure among the N^D possible ones.

There is obviously no better algorithm but maybe they are all equivalent. To prove that it is not the case we just have to find an algorithm that is strictly outperformed by a_* .

In the worst case, i.e. if $M' = M$, a_* finds a solution of f_1 at the very first sampling, a solution of f_2 at the second sampling, etc. Its efficiency on f_i is therefore equal to

$$(3.1) \quad E_{a_*, f_i} = \frac{|X| - j}{|X| - 1}$$

and its mean efficiency on B

$$(3.2) \quad \begin{aligned} E_{a_*, B} &= \frac{1}{M} \frac{1}{|X|-1} \left(M |X| - \frac{M(M+1)}{2} \right) \\ &= \frac{1}{2(|X|-1)} (2 |X| - |B| - 1) \end{aligned}$$

It linearly decreases from 1 to 0.5 when the size of the benchmark increases from 1 to $|X|$ and is an upper bound for the efficiency of any algorithm⁵. Now let us consider the algorithm a_0 that first samples x_0 and then like a_*

$$a_0 = (x_0, x_2, \dots, x_{M'}, \dots)$$

It finds a solution of f_1 only after more than M' samples. Therefore it is worse than a_* ■

Note there are usually many possible algorithms equivalent to a_* , given by the permutations of $(x_1, x_2, \dots, x_{M'})$, and the permutations of the other points of X . Let us call A_* their set. We have

⁵If you know both the maximum and the minimum values of a function f , you can define something more sophisticated, by assigning a 'quality' to each sampled point x , for example $\frac{f_{max} - f(x)}{f_{max} - f_{min}}$, and combining then all the quality values over the sampled sequence. But this is another topic.

$$|A_*| = M'!(|X| - M')!$$

3.3. Conditions for a partial-ultimate algorithm. However, as said, what is really interesting is to know under which conditions, both on the benchmark B and on the set of algorithms A we compare, there exists at least one of these algorithms that outperforms all the ones of A on all functions of the benchmark, or, more precisely, that is partial-ultimate.

Again we consider the worst case, in which the set of different solutions is $R = \{x_1, x_2, \dots, x_M\}$. A first condition, which always holds in practice, is that R is strictly included in the search space X . An obvious second one is that the benchmark B is not closed under permutations.

Let a_0 now be an algorithm for which none of its M first samples is in R . It is easy to see that a_* outperforms it on all functions of B . Let us define A_0 as the set of all algorithms $a_{0,i}$ that have the same property as a_0 . Then a_* is partial-ultimate on B for A_0 .

But this case is too rudimentary. We need a more realistic and weaker condition. For example we can define the algorithm $a_{0,i}$ by the

Condition 3:

- for $k \neq i$ it samples x_k , like a_* .
- for $k = i$ it samples a point that is not solution of f_i .

Then a_* is partial-ultimate on B for $A = \bigcup_{k=1}^M a_{0,k}$.

Many similar sufficient conditions are of course possible, meaning there usually exist many sets of algorithms for which a_* is partial-ultimate.

4. FRUSTRATIONS

Let B be a set of M problems and A a set of algorithms. According to the classical NFLT, if B is closed under permutations, no matter A is, there is not best algorithm in it.

And according to the theorems we have seen, in practice B is not c.u.p. and there almost always exists a algorithm a_* that is the best for A , but no way to find it.

But there is something even more frustrating: in practice (i.e. B not c.u.p. and $|A|$ small compared to size of the search space) there almost always exists a set of problems B' on which a_* is now the worst for A .

Let $R = \{x_{1,1}, \dots, x_{|A|,1}\}$ be the first points of the algorithms/sequences of A .

The conditions are:

- (1) a_* is not in A
- (2) no $x_{i,1}$ is also the first point of a_*
- (3) or a stronger condition, no $x_{i,1}$ is one of the $|A|$ first points of a_*

Note that on classical studies (comparison of a few sophisticated algorithms on a 'decent' benchmark) these conditions always hold.

Example 1. Under conditions 1 and 2 we can define a B' that contains just one function, f_1 , which has $|A|$ global minima on the positions of R . On f_1 each algorithm of A finds the solution at the very first sampling, as a_* doesn't ■

Example 2. Under conditions 1 and 3, we define f_i as a function whose minimum is on $x_{i,1}$, and $B' = \{f_1, \dots, f_{|A|}\}$. Then, on B' , and after $|A|$ evaluations, each a_i has found at least one solution (for f_i), as a_* found none. More formally, the mean efficiency of a_i is at least $\frac{1}{|A|}$ as the one of a_* is zero.

So each a_i is strictly better than a_* ■

Example 3. Now you want to prove that a_* can be the worse on a benchmark B' , even if the efficiencies are computed after the $|X|$ possible samplings (exhaustive search). To explain it, let us consider a simple case with just two functions $B = \{f_1, f_2\}$, and two algorithms $A = \{a_1, a_2\}$. The solutions are $R = \{x_1, x_2\}$. We do suppose we found, as seen, an a_* that is the best, or even ultimate on B for A , and the set of such algorithms is A_* .

Let us call α the first point of a_1 and β the first point of a_2 . They are both supposed to be different from x_1 and x_2 .

We define a new benchmark by

$$B' = \{f_\alpha, f_\beta\}$$

where the minimum of f_α is on α and the one of f_β on β .

As we sample all points, we necessarily have something like $a_1 = (\alpha, \dots, \beta, \dots)$ and $a_2 = (\beta, \dots, \alpha, \dots)$.

Let $r_{1,\beta}$ be the rank of β in a_1 and $r_{2,\alpha}$ the rank of α in a_2 . Therefore the efficiencies are

$$\begin{aligned} E_{a_1, f_\alpha} &= 1 \\ E_{a_1, f_\beta} &= \frac{|X| - r_{1,\beta}}{|X| - 1} \\ E_{a_2, f_\alpha} &= \frac{|X| - r_{2,\alpha}}{|X| - 1} \\ E_{a_2, f_\beta} &= 1 \end{aligned}$$

Any algorithm in A_* is a sequence like $a_* = (x_1, x_2, \dots, \alpha, \dots, \beta, \dots)$ or a similar one by permuting (x_1, x_2) and (α, β) . Let r_α be the rank of α in a_* , and r_β the rank of β . The efficiencies are

$$\begin{aligned} E_{a_*, f_\alpha} &= \frac{|X| - r_\alpha}{|X| - 1} \\ E_{a_*, f_\beta} &= \frac{|X| - r_\beta}{|X| - 1} \end{aligned}$$

The claim is that we can have

$$E_{a_*, f_\alpha} < E_{a_2, f_\alpha}$$

and

$$E_{a_*, f_\beta} < E_{a_1, f_\beta}$$

Actually we of course just have to prove we can have

$$(4.1) \quad \begin{aligned} r_\alpha &> r_{2,\alpha} \\ r_\beta &> r_{1,\beta} \end{aligned}$$

It is *not* possible if $r_{2,\alpha} = |X|$ or $r_{1,\beta} = |X|$, but this cases are rare. For example the first one means that the algorithm a_2 never finds the solution of f_α , or, more precisely, only at the very end of the exhaustive search.

So, let us suppose we have $\max(r_{2,\alpha}, r_{1,\beta}) < |X| - 1$. Among the $2(|X| - 2)!$ algorithms of A_* there is then at least one a_* for which α and β are found only after, respectively, $r_{2,\alpha}$ and $r_{1,\beta}$ samplings. And, therefore, the inequalities 4.1 hold.

Back to the Warm Up example. Let us define $B = \{f_2, f_4\}$, and $A = \{a_5, a_6\}$. Remember we have

$$\begin{aligned} f_2 &\equiv (1, 0, 0) \\ f_4 &\equiv (0, 0, 1) \\ a_5 &= (3, 2, 1) \\ a_6 &= (3, 1, 2) \end{aligned}$$

According to the table 3 we can choose $a_* = (2, 1, 3)$ as ultimate algorithm.

Now, if we define $B' = \{f_3, f_5\}$, then the efficiencies of a_* are 0.5 and 0, as the efficiencies of a_5 and a_6 are 1 and 1.

So, on B' , a_* is now the worst compared to A .

5. CONCLUSION

On a computer the search space and the value space contain a finite number of points. When you define a benchmark in order to compare some algorithms, you may therefore think a best algorithm in average can not exist because of the NFLT. But in fact it is highly unlikely for a benchmark to meet the conditions of this theorem. So unlikely that it is virtually impossible, unless you build it precisely for that. It means that in practice, an algorithm that outperforms in average all the other ones always exists.

Moreover, when you consider a subset of all possible algorithms, it also almost surely exists a partial-ultimate one that is better or equivalent on each function, and strictly better on at least one.

But in both cases, knowing such good algorithms do exist does not mean it is easy to find them!

Worse, even if, on a given benchmark, you find an algorithm that is better than all the ones of a set of other algorithms, in practice there exists another benchmark on which it is on the contrary outperformed by all these other algorithms.

It means that we should never simply claim 'This algorithm is better than these ones', but we should precise 'on *this* class of problems', ... assuming we are able to rigorously define what such a class is, and, of course, assuming we are able to prove such an assertion, which may be extremely difficult. So, finally, quite often, when comparing some algorithms, the most honest claim should be something like '*This* one is better than *these* ones on *these* problems, which seem to have *these* properties in common'.

REFERENCES

- [1] Anne Auger and Olivier Teytaud. *Continuous lunches are free*. GECCO, 2007.
- [2] C. Schumacher, M. D. Vose, and L. D. Whitley. The No Free Lunch and problem description length. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, GECCO'01, pages 565–570, San Francisco, CA, USA, July 2001. Morgan Kaufmann Publishers Inc.
- [3] Matthew Streeter. Two Broad Classes of Functions for Which a No Free Lunch Result Does Not Hold. pages 1418–1430, July 2003.
- [4] David H. Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.