PART 1

Randomness in Optimization

1

Necessary Risk

Il arrive souvent de ne rien obtenir parce que l'on ne tente rien
(Often, nothing is gained because nothing is attempted)

Jacques Deval

In using chance to solve a problem, there is always a risk of failure, unless an unlimited number of attempts are permitted: this is rarely possible. The basic idea involved in stochastic optimization is that this risk is necessary, for the simple reason that no other solution is available; however, it may be reduced by carefully controlling the use of random elements. This is generally true, in that a correctly-defined optimizer will produce better results than a purely random search for most test cases. However, this is not always the case, and the ability to identify these "anomalous" situations is valuable.

1.1. No better than random search

Let us take a set of permutation tests. A precise definition is given in the Appendices (section 7.1). Here, note simply that based on one discrete finite function, all of the other functions can be generated by permutations of possible values at each point.

The definition space is $E = (0, 1, 2, 3)$ and the value space is $V = (1, 2, 3)$. A function is therefore defined by its values at the points of E, for example $f_1 \equiv (1, 3, 2, 2)$. One possible permutation of this function is

$f_2 \equiv (1, 2, 3, 2)$; there are 12 such functions in total, each of which is a permutation of the others, shown in the first column of Table 1.1. Each function has a minimum value of 1 (to simplify our discussion, optimization in this case will always be taken to mean minimization). Now, let us consider three iterative algorithms, and calculate the probability that they will find the minimum of each function. These algorithms are all without repetition, and conserve the best position obtained along with the associated value (the ratchet effect). A brief, informal description of these algorithms is given below. For each, the result is given as a pair $(x^*, f(x^*))$, where x^* is the proposed solution.

1.1.1. *Uniform random search*

This algorithm, like those which follow, includes an initialization phase, followed by an iteration phase (see section 1.1.). Let us calculate the probability $p(t)$ of finding the solution after t position draws. As there is only one solution, $p(1) = \frac{1}{4}$, the probability of *not* obtaining the solution on the first try is therefore $1 - p(1)$. In this case, as three nonsampled permutations remain, the probability of obtaining the solution on the second try is $\frac{1}{3}$. Thus, the probability of obtaining the solution on the first *or* second try is $p(2) = p(1) + (1 - p(1))\frac{1}{3} = \frac{1}{4} + \frac{3}{4}\frac{1}{3} = \frac{1}{2}$. Similarly, the probability of obtaining the solution on the first, second *or* third try is $p(3) = p(2) + \left(1 - p(2)\frac{1}{2}\right) = \frac{3}{4}$. Evidently, as the algorithm is without repetition, the probability of having found the solution on the fourth try is 1, as an exhaustive search will have been carried out.

Algorithm 1.1. Random search without repetition

Initialization

– Draw a position x^* at random, following a uniform distribution (each position has the same selection probability).

Iterations

As long as the STOP criterion (for example a maximum number of iterations) has not been reached:

– draw a position x at random from the unsampled population;

– if $f(x) < f(x^*)$, then replace x^* by x.

1.1.2. *Sequential search*

This method consists of drawing positions one by one, not at random (without repetition), but in a predefined order, for example position 1, position 2, etc. To calculate $p(t)$, each function must be considered individually. For $f_4 \equiv (3,1,2,2)$, for example, a solution will definitely be found after two tries, compared to a probability of $\frac{1}{2}$ using the previous method.

However, the opposite situation also occurs, for example for $f_6 \equiv (3,2,2,1)$. After two tries, the solution can not be found, as the random method may find it, with a probability of $\frac{1}{2}$. Overall, this method is therefore equivalent to the previous method in terms of probabilities $p(t)$. Improvements are thus required.

1.1.3. *Partial gradient*

Using this method, the first two positions are drawn sequentially. Next, if the two values obtained are decreasing, the sequential approach is retained, as the "direction" of the search appears to be correct. Otherwise, positions are drawn at random from the remaining population. This means that differences from the previous method will only emerge at draw $p(3)$. Once again, each function must be examined individually for calculation purposes. Take, for example, a function such as $f_6 \equiv (3,2,2,1)$. The first two draws give results of 3 and 2. As the direction appears promising, the third position is drawn: the value is 2. This is not the minimum, as $p(3) = 0$. With a function such as $f_9 \equiv (2,2,1,3)$, there is no clear preferred direction, and so the third point is drawn at random from the two remaining points, giving a probability of success of $\frac{1}{2}$.

The probabilities of success for these three methods, $p(t)$ for $t = 1,2,3$, using the 12 function test case defined above, are given in Table 1.1. Naturally, all of these algorithms obtain the solution with the same probability, 1 (certainty), after four attempts, as they are repetition-free. However, their average performance will not be necessarily identical after one, two or three attempts. The partial gradient algorithm, which is slightly more sophisticated, might be expected to be somewhat more efficient; it is the only method which has a chance of finding a solution to f_{10} or f_{11} after three

attempts. However, success is not guaranteed for f_9 and f_{12}. Finally, as demonstrated in the final line of the table, the three algorithms give the same average performance over the set of test cases.

	One attempt		Two attempts		Three attempts		
	Random	Seq.	Rand.	Seq.	Rand.	Seq.	Grad.
$f_1 \equiv (1,3,2,2)$	$1/4$	1	$1/2$	1	$3/4$	1	1
$f_2 \equiv (1,2,3,2)$	$1/4$	1	$1/2$	1	$3/4$	1	1
$f_3 \equiv (1,2,2,3)$	$1/4$	1	$1/2$	1	$3/4$	1	1
$f_4 \equiv (3,1,2,2)$	$1/4$	0	$1/2$	1	$3/4$	1	1
$f_5 \equiv (3,2,1,2)$	$1/4$	0	$1/2$	0	$3/4$	1	1
$f_6 \equiv (3,2,2,1)$	$1/4$	0	$1/2$	0	$3/4$	0	0
$f_7 \equiv (2,1,3,2)$	$1/4$	0	$1/2$	1	$3/4$	1	1
$f_8 \equiv (2,1,2,3)$	$1/4$	0	$1/2$	1	$3/4$	1	1
$f_9 \equiv (2,2,1,3)$	$1/4$	0	$1/2$	0	$3/4$	1	$1/2$
$f_{10} \equiv (2,2,3,1)$	$1/4$	0	$1/2$	0	$3/4$	0	$1/2$
$f_{11} \equiv (2,3,2,1)$	$1/4$	0	$1/2$	0	$3/4$	0	$1/2$
$f_{12} \equiv (2,3,1,2)$	$1/4$	0	$1/2$	0	$3/4$	1	$1/2$
Average	**1/4**	**1/4**	**1/2**	**1/2**	**3/4**	**3/4**	**3/4**

Table 1.1. *Permutation test cases. Probability of success after one, two and three attempts. The three algorithms are repetition-free and present the same average performance, as the conditions of the No Free Lunch Theorem (NFLT) are fulfilled*

This is due to the fact that the conditions of the NFLT [WOL 97, IGE 03] are met. Without going into the mathematical formalization, these conditions are:

– finite discrete definition space;

– finite discrete value space;

– set of test cases closed under permutations (c.u.p.).

Under these conditions, any repetition-free algorithm, no matter how sophisticated, will present the same average performance in terms of random search, however performance is measured. Note that the first two conditions are necessarily fulfilled if calculations are carried out digitally, as a computer always has a limited number of bytes, and thus the numbers which may be represented are limited. This means that an algorithm can only out-perform random search methods in a non-c.u.p. set of test cases. This is the case for most sets of test cases. However, there will always be at least one function

where the algorithm will perform less well than a purely random search (without repetition). This is the first risk which must be taken. We might hope that functions of this type would be so "monstrous" as to make this situation practically impossible, but this is too optimistic; in reality, the situation can arise in cases where a function presents a sufficiently high proportion of plateaux, as discussed in section 1.2.2.

Note that, in theory, there is nothing to prevent the existence of a single, ultimate algorithm, better than any other, for a set of non-c.u.p. functions [CLE 07]. Authors occasionally claim that an algorithm is better than all others for the majority of problems, but not all, in a set of test cases, "due to the No Free Lunch theorem". In reality, the theorem should not be used in this way, if only because the prerequisites are not satisfied for the test set in question. However, a real, widely-noted phenomenon does exist: for a "good" test set (this notion will be discussed later), the higher the efficiency of an algorithm for certain problems, the weaker its performance will be in other cases. Although, superficially, this appears to be a result of the theorem, it is, in reality, due to another factor. This is illustrated by the fact that the algorithm presents better overall performance than a random search for the problems in the test set, despite the performance differences for individual examples. This type of behavior will be discussed in the following section.

1.2. Better or worse than random search

Clearly, all of the algorithms developed over the years present better results than those of a random search in the majority of cases. Moreover, improvements are regularly put forward, although these are always for specific classes of problems. However, two problems exist. First, with a few exceptions, the definition of the classes in question is insufficiently precise (for example, how, precisely, do we define "weakly multimodal"?); second, as metaheuristics are often used in a "black box" context, we do not know, *a priori*, what class the problem in question belongs to. Therefore, there is a risk of using an unsuitable algorithm, producing poor results.

However, if we know that a problem belongs to a clearly-defined class, such as the class of unimodal continuous functions, then it should be possible to find an ultimate, unbeatable algorithm. However, many test sets are constructed using problems from a variety of different classes (unimodal,

multimodal, continuous, discrete, mixed, separable or otherwise, high or low dimensionality, etc.), in an attempt to represent the variety of situations encountered in the real-world. An iterative algorithm will only be more efficient than a random search if, over the course of iterations, it collects and uses information relating to the structure of the problem. As a starting point, a structure needs to exist: as this is always the case in practice, the risk may be left aside. More importantly, the information needs to be used in a relevant manner, and this is where many metaheuristics fall short.

All metaheuristics presume, explicitly or implicitly, that the problem to solve belongs to the broad "positive correlation" class [CLE 07, CLE 11]. This notion will be discussed below, without going into mathematical detail.

1.2.1. *Positive correlation problems*

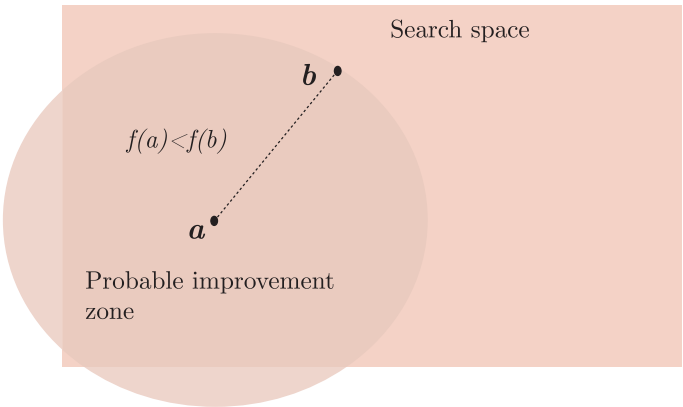
This class covers all functions where, if a point x is better than a point y, then the average probability of finding a better position than y is higher in the domain defined by a set of points z such that $distance(x,z) < distance(x,y)$ than outside of this area. Using Euclidean distance (2-distance), this is a hypersphere of center x and radius $\|y-x\|_2$. Other distances may be used, such as 1-distance (taxi-distance or Manhattan distance) or, for combinatorial problems, the distance between two permutations, such as the Cayley distance[1] or the Kendall-Tau distance[2], which is easier to calculate, but less intuitive.

This property is shown in Figure 1.1. Taking the value of a position to be its *quality* (in this case, the lower the value, the higher the quality), and referring to the distance between two positions as *proximity*, then the term "correlation" may be taken to mean the average relationship between proximity and quality. Table 1.2 gives a number of examples. Note the case of combinatorial problems, which present coefficients which are as positive as for certain "continuous" problems (these problems are, in fact, always discrete when processed digitally). There is no fundamental difference between a discrete problem and a combinatorial problem, as explained in [CLE 06]: a search space and metrics are used in both cases. This suggests

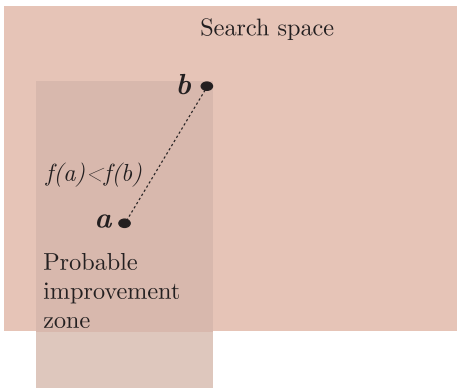
1 The minimum number of transpositions needed to pass from one permutation to the other.

2 The number of pairs which are not in the same order in the two permutations.

that efficient methods which appear to be specific to combinatorial problems may also be suitable for use with discrete problems; however, this is a separate issue.



a) Euclidean distance.



b) Taxi-distance.

Figure 1.1. *Illustration of the "positive correlation" property. In average terms, across a definition space, a position is more likely to be improved by moving toward another, better position than vice versa. A variety of distances may be used*

In practice, the positive correlation hypothesis is reflected in the search algorithm through a number of rules:

– if a position is good, then search "around" this position;

– if a better position is found by moving in a certain direction, continue to move in this direction;

– etc.

Most algorithms, with the exception of "greedy" algorithms, also include opposite rules in an attempt to limit premature convergence, notably in the case of local minima; however, it is important to note that rules of the type described above are applied more frequently than these opposite rules. Consequently, if the problem does not belong to a c.u.p. class, there is a risk that the algorithm will perform less well than a random search. Some examples of this type will be presented in the following section.

Note that this is an average property. Generally, a problem is closed under permutations for one or more subdomains of the definition space, and not for others. The problem will be more or less easy to solve for an algorithm, which presumes that the property is globally verified, according to the relative importance of these subdomains. Moreover, certain problems which appear very simple demonstrate negative correlation (again, as an average value).

	Coeff.	Type of distance
Alpine 2D	0.51	Euclidean
Rosenbrock 2D	0.51	Euclidean
Pressure vessel	0.54	Euclidean, over normalized search space
Alpine 10D	0.41	Euclidean
Rosenbrock 10D	0.50	Euclidean
Sphere 30D	1	Euclidean
Traveling salesman 6 towns	0.37	Cayley
Traveling salesman 14 towns (Burma 14)	0.43	Cayley

Table 1.2. *Examples of positive correlation problems. The precise definitions of these problems are given in the appendix*

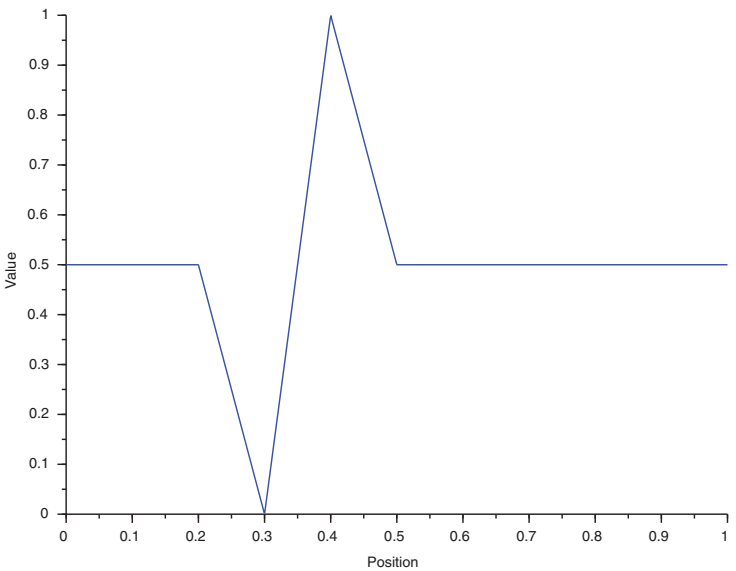
1.2.2. *Negative correlation problems*

Certain problems, even those with relatively simple definitions, do not present positive correlation. Table 1.2 and Figure 1.2 provide a number of examples. This does not necessarily indicate that a random search will give the best performance, as explained in the chapter 6 about comparisons of

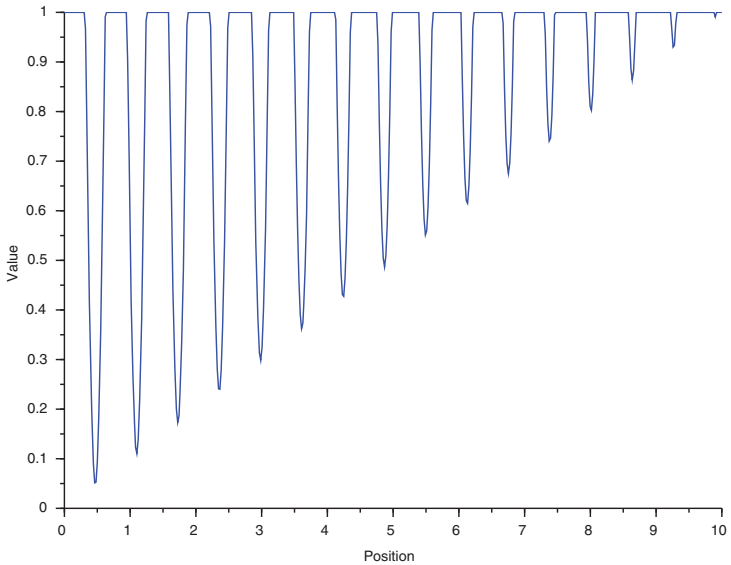
optimizers; however, this is a risk. In case of doubt, i.e., in practice, if the problem "landscape" is suspected of containing plateaux, it is as well to check whether or not any new stochastic algorithm is genuinely superior to a random search approach. Even if this is not the case, improvements may be possible. However, this requires a clear understanding of the precise definition of the term "stochastic".

	Distance-quality correlation
Deceptive 1 (Flash)	-0.3
Deceptive 2 (Comb)	-0.34
Deceptive 3 (Brush)	-0.90

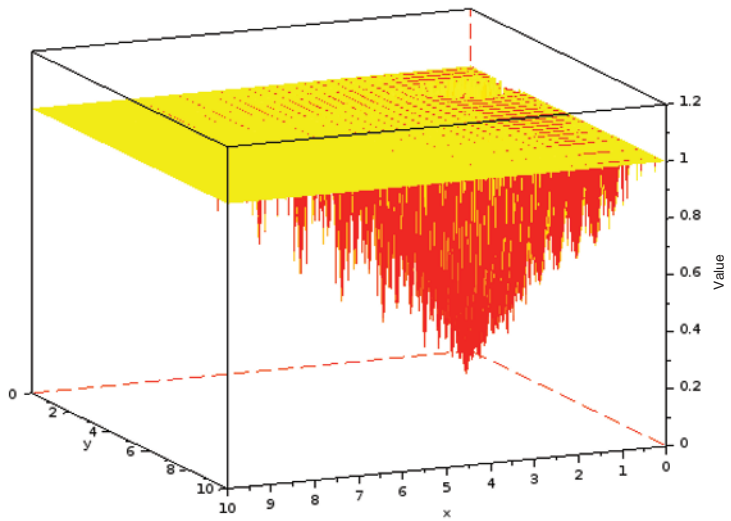
Table 1.3. *Three simple functions showing negative correlation*



a) Deceptive 1 (Flash). Correlation coefficient of around -0.3.



b) Deceptive 2 (Comb). Correlation coefficient of around -0.34.



c) Deceptive 3 (Brush). Correlation coefficient of around -0.90.

Figure 1.2. *Examples of functions with no "positive correlation" property, due to the existence of plateaux. In these cases, sophisticated optimization algorithms may perform more poorly than purely random search methods. For a color version of the figure, see www.iste.co.uk/clerc/randomness.zip*