THE REACTIVE AFFINE SHAKER: A BUILDING BLOCK FOR MINIMIZING FUNCTIONS OF CONTINUOUS VARIABLES

Mauro Brunato, Roberto Battiti

March 8, 2006

Technical Report # DIT-06-012

The Reactive Affine Shaker: a Building Block for Minimizing Functions of Continuous Variables

Mauro Brunato, Roberto Battiti
Department of Computer Science and Telecommunications,
Università di Trento, via Sommarive 14, I-38050 Trento — Italy.
E-mail: (battiti|brunato)@dit.unitn.it

March 8, 2006

Abstract

A novel adaptive random search algorithm for the optimization of functions of continuous variables is presented. The scheme does not require any assumptions about the function to be optimized, apart from the availability of evaluations $f(x)$ at selected test points. We assume that the main computational cost lies in the function evaluations and the main design criteria of the RASH scheme consists of the adaptation of a search region by an affine transformation which takes into account the local knowledge derived from trial points generated with a uniform probability. The aim is to scout for local minima in the attraction basin where the initial point falls, by adapting the step size and direction to maintain heuristically the largest possible movement per function evaluation. The design is complemented by the analysis of some strategic choices (like the double-shot strategy and the initialization) and by experimental results showing that, in spite of its simplicity, RASH is a promising building block to consider for the development of more complex optimization algorithms. The developed software is built to facilitate the scientific experimentation and the integration of RASH as a component in more complex schemes.

1 Introduction

Finding the global minimum of a function of continuous variables $f(\boldsymbol{x})$ is a well known problem for which substantial effort has been dedicated in the last decades, see for example the bibliography in [12]. Apparently, no general-purpose *panacea* method exists which can guarantee its solution at a desired accuracy within finite and predictable computing times. In fact, the different versions of the so called "no free lunch theorems" imply that "for any algorithm any elevated performance over one class of problems is paid for in performance over another class", see [16].

On the other hand, most real-world optimization tasks are characterized by a rich correlation structure between candidate solutions which are close, in a suitable metric defined over the independent variables. Local search techniques capitalize on this local structure by postulating that a better solution can usually be found in the *neighborhood* of the current tentative solution. In this manner, after starting from an initial configuration of the independent variables $\boldsymbol{x}^{(0)}$, a *search trajectory* of a discrete dynamical system is generated, in which point $\boldsymbol{x}^{(t+1)}$ is chosen in the neighborhood of point $\boldsymbol{x}^{(t)}$. Under suitable conditions (e.g., lower-bounded function, decreasing values of $f(\boldsymbol{x}^{(t)})$ during the search with a sufficiently fast rate of decrease) the trajectory will converge at a *local minimizer.* The set of initial points which are mapped to a specific local minimizer by the local search dynamical system is called the *basin of attraction* of the minimizer.

Many recent global optimization techniques deal with ways to use a local search technique without being trapped by local minimizers, notably the Simulated Annealing technique based on Markov chains, see for example [5] and [13].

Because of the growing awareness that no single general-purpose methods can be efficiently applied to different problems, recent research consider the appropriate integration of basic algorithmic building blocks, like various stochastic local search techniques [11], the so-called meta-heuristic techniques [7], the various combinations of genetic operators proposed by [8], the *algorithm portfolio* proposals [9]. The crucial issue is that of tailoring the appropriate combination of components and values of critical parameters of the algorithms to a specific optimization instance, a process that implies an expensive learning phase by the user and that can be partially automated by machine learning techniques, as it is advocated in the reactive search framework [2] (see also the web site `www.reactive-search.org`).

Research and applications demands a careful design of each component, which should be studied in isolation before considering integration in more complex schemes. In this manner, the added value of the combination w.r.t. the components can be judged in a statistically sound manner. *Software plays a crucial role* in this context, both to ensure reproducibility of results (in many cases published results cannot be duplicated by other researchers because the author forgot to mention and document some "implementation details"), and to facilitate a scientific methodology for the development of complex heuristic schemes, where the combination and appropriate determination of parameters should be guided by sound statistical techniques.

In this paper we focus on a "direct method" for optimization, see [10], which consider only function evaluations. We furthermore assume *no a priori knowledge* about f, the knowledge will be only that acquired during evaluations $f(\boldsymbol{x})$ at different values of the independent parameters. In particular we develop a component based on the stochastic (or random) local search framework originally proposed in [14]. We state the main design criteria and study some critical choices in the development of this component, in particular the initial phase and the adaptation of the search neighborhood based on the local structure of a given attraction basin, leading to a version which we term "Reactive Affine Shaker" (RASH) for reasons explained later. We develop a software component in the C++ object-oriented language, and demonstrate its applicability by experimenting on a widely used set of benchmark functions. Given the algorithmic simplicity of the component and the effectiveness demonstrated, we suggest the consideration of the RASH method and software in different and more complex optimization schemes.

This paper is structured as follows. In Section 2 the RASH technique is described and motivated and a pseudo-code description is provided. In Section 3 the object–oriented code structure of the continuous optimization framework is presented. In Section 4 the experimental results are shown, together with the comparison with other published algorithms.

For completeness, two appendices deal with the mathematical analysis and theorems to motivate some aspects of the RASH technique. (NOTE: depending on the reviewers' opinion, the appendices could be omitted from the final version of the paper).

2 The Reactive Affine Shaker Algorithm

The Reactive Affine Shaker algorithm (or RASH for short) is an adaptive random search algorithm based on function evaluations. The seminal idea of the scheme was presented for a specific application in neural computation in [1]. The current work motivates the scheme though a detailed analysis the specific form of search executed (called "double shot"), and it proposes a more effective strategy during the initial part of the search by analyzing the evolution of the search direction in the first iterations, when the search succeeds with a very high probability (close to one). Finally, it designs reusable software components implementing the scheme.

2.1 Motivation and analysis

The algorithm starts by choosing at random (in the absence of prior knowledge) an initial point $\boldsymbol{x}$ in the configuration space; this point is surrounded by an initial *search region* $\mathcal{R}$ where the next point along the trajectory is searched for.

In order to keep a low computation overhead, the *search region* is identified by n vectors, $\boldsymbol{b}_1, \dots, \boldsymbol{b}_n \in \mathbb{R}^n$ which define a "box" around the point $\boldsymbol{x}$:

$$\mathcal{R} = \left\{ \boldsymbol{x} + \sum_{i=1}^{n} \alpha_i \boldsymbol{b}_i, \qquad \alpha_1, \dots, \alpha_n \in [-1, 1] \right\}. \tag{1}$$

The search occurs by generating points in a stochastic manner, with a uniform probability in the search region. For a reason which will become clear in the following description, a single displacement $\boldsymbol{\Delta}$ is generated and two specular points $\boldsymbol{x}^{(t)} + \boldsymbol{\Delta}$ and $\boldsymbol{x}^{(t)} - \boldsymbol{\Delta}$ are considered in the region for evaluation (*double shot*, see also [3]). An evaluation is "successful" if the f value is better than the value at the current point.

By design, RASH is *an aggressive local minima searcher*: it aims at converging rapidly to the local minimizer corresponding to the attraction basin where the initial point falls.

We assume that most computational effort during the search is spent by calculating function values $f(\boldsymbol{x})$ at tentative points. Because of the algorithm simplicity, the assumption is valid for non-trivial real-world problems.

The search speed is related to the average size of the steps ($\|\boldsymbol{x}^{(t+1)} - \boldsymbol{x}^{(t)}\|$) executed along the search trajectory. Let's consider two extreme cases. If the search region is very small and the function is smooth, the "double shot" strategy will produce a new successful point with probability close to one, see Appendix A, but the step will be very small. *Vice versa*, if the search region is very large and it coincides with the initial range of interest, the search strategy will become that of naïve random search: points are generated at random in the entire search space. The step can be large, but the locality assumption is lost and, unless the problem is very simple, a potentially very large number of points will have to be evaluated before finding a successful one. Ideally, the maximize the usage of the information derived from the costly $f(\boldsymbol{x})$ computations, one should aim at the *largest possible step per function evaluation.* This optimal criterion cannot in general be fulfilled, in particular if the analytic form of the function is not known and values $f(\boldsymbol{x})$ are obtained by simulation.

RASH aims at maintaining the search region size as large as possible, while still ensuring that the probability of a success per evaluation will be reasonably close to one (success probabilities in the range 0.3 - 0.5 are considered acceptable). Now, the success probability is related both to the area of the search region, and to its form. For example, if the attractor basin consists of an elongated and narrow valley leading to a local minimizer, for a fixed area, a search region elongated along the bottom of the valley will guarantee a higher success rate of the double shot strategy, and therefore longer average step sizes.

RASH obtains both design objectives: (i) success probability per sample close to one and (ii) largest possible step size per successful sample, trough a "reactive" determination of the search area during the search. For objective (i) the area is enlarged if the search is successful, reduced if unsuccessful, for objective (ii) the area is elongated along the last successful direction. Of course, "largest possible" has a heuristic meaning: given the partial knowledge about f and the lack of constraints about its functional form we are satisfied if a reasonably large step is determined by a simple reactive scheme.

With more detail, the algorithm proceeds by iterating the following steps:

1. A new tentative point is generated by sampling the local search region $\mathcal{R}$ with a uniform probability distribution and by using the "double shot" strategy. The second specular shot is evaluated only if the first one does not succeed.

2. The search region is modified according to outcome of the tentative point. It is compressed if the new function value is greater than the current one (unsuccessful sample), it is expanded otherwise (successful sample).

Modification of the search region is performed by taking into account the direction of the last tentative step. In RASH, the search area defined by vectors $\boldsymbol{b}_i$ undergoes an affine transformation, see equations (3)–(4) below.

3. If the sample is successful, the new point becomes the current point, and the search region $\mathcal{R}$ is translated so that it becomes centered around the new point.

A last design decision concerns the initial size of the search area, in the absence of initial information about the local attraction basin of f. Two simple options, which do not require critical parameters to be tuned, are to start with a search area corresponding to the initial search range, which will be rapidly compressed in the following iterations until it leads to a success, or, on the contrary, to start with a very small search area, which will be rapidly expanded. The first option is in conflict with the requirement that RASH should scout for the local minimizer corresponding to the basin of attraction of the initial point. If arbitrarily large jumps are permitted at the beginning, all attraction basins could be reachable, with a probability depending on their sizes. Therefore we adopted the second option.

As it is demonstrated in the Appendix A, when the function is smooth and the search region area goes to zero, the probability of success of the "double shot" strategy tends to one, no matter what the initial direction is. This fact creates an undesired effect: after picking the first tentative direction, one will have an uninterrupted sequence of successes. At each step, the search area will be expanded along the last direction, which in turn was be generated with uniform probability in an already elongated region. Through this self-reinforcing mechanism one may easily get an extremely elongated search region, where the elongation tends to be collinear with the first *random* direction (and with no influence from the current basin form). To avoid this spurious effect, the expansion of the search region is isotropic in the initial part of the search, until the first unsuccessful direction is encountered, i.e., all box vectors are expanded by the same factor.

The details about the evolution of directions during the initial phase of the search, as well as the experiments related to the correlation between initial search directions, are explained in Appendix B, see for example Fig. 10.

After explaining the design choices, let's now comment on the name (Reactive Affine Shaker). The solver's movements try to minimize the number of jumps towards the minimum region, and this is achieved by constantly changing the movement direction and size. Search region and therefore step adjustments are implemented by a feedback loop guided by the evolution of the search itself, therefore implementing a "reactive" self-tuning mechanism. The constant change in step size and direction creates a "shaky" trajectory, with abrupt leaps and turns. Last, modifications of the search parameters are through an affine transformations on the shape of the search region.

Variable	Scope	Meaning
f	(input)	Function to minimize
$\boldsymbol{x}$	(input)	Initial point
$\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$	(input)	Vectors defining search region $\mathcal{R}$ around $\boldsymbol{x}$
ρ_e, ρ_r	(input)	Box expansion and reduction factors
d	(input)	Dimension of the space
t	(internal)	Iteration counter
$\mathbf{P}$	(internal)	Transormation matrix
$\boldsymbol{x}$, $\boldsymbol{\Delta}$	(internal)	Current position, current displacement

```
1.  function AffineShaker ($f$, $\boldsymbol{x}$, ( $\boldsymbol{b}_j$), $\rho_e$, $\rho_r$)
2.      $t \leftarrow 0$;
3.      repeat
4.          $\boldsymbol{\Delta} \leftarrow \sum_j \mathrm{Rand}(-1,1)\boldsymbol{b}_j$;
5.          if $f(\boldsymbol{x}+\boldsymbol{\Delta}) < f(\boldsymbol{x})$
6.              $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{\Delta}$;
7.              $\mathbf{P} \leftarrow \mathbf{I} + (\rho_e - 1)\frac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\| \boldsymbol{\Delta} \|^2}$;
8.          else if $f(\boldsymbol{x}-\boldsymbol{\Delta}) < f(\boldsymbol{x})$
9.              $\boldsymbol{x} \leftarrow \boldsymbol{x} - \boldsymbol{\Delta}$;
10.             $\mathbf{P} \leftarrow \mathbf{I} + (\rho_e - 1)\frac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\| \boldsymbol{\Delta} \|^2}$;
11.         else
12.             $\mathbf{P} \leftarrow \mathbf{I} + (\rho_c - 1)\frac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\| \boldsymbol{\Delta} \|^2}$;
13.         $\forall j\ \boldsymbol{b}_j \leftarrow \mathbf{P}\ \boldsymbol{b}_j$;
14.         $t \leftarrow t+1$
15.     until convergence criterion;
16.     return $\boldsymbol{x}$;
```

Figure 1: The Affine Shaker algorithm

2.2 RASH pseudo-code

Details of the RASH algorithm are shown in Fig. 1. At every iteration, a displacement $\boldsymbol{\Delta}$ is generated so that the point $\boldsymbol{x}+\Delta$ is uniformly distributed in the local search region $\mathcal{R}$ (line 4). To this end, the basis vectors are multiplied by random numbers in the real range $[-1, 1]$ and added:

$$\boldsymbol{\Delta} = \sum_j \mathrm{Rand}(-1,1)\boldsymbol{b}_j. \tag{2}$$

If one of the two points $\boldsymbol{x}+\Delta$ or $\boldsymbol{x}-\Delta$ improves the function value, then it is chosen as the next point. Let us call $\boldsymbol{x}'$ the improving point. In order to enlarge the box along the promising direction, the box vectors $\boldsymbol{b}_i$ are modified as follows.
The direction of improvement is $\boldsymbol{\Delta}$. Let us call $\boldsymbol{\Delta}'$ the corresponding versor

$$\boldsymbol{\Delta}' = \frac{\boldsymbol{\Delta}}{\|\boldsymbol{\Delta}\|}.$$

Then the projection of vector $\boldsymbol{b}_i$ along the direction of $\boldsymbol{\Delta}$ is

$$\boldsymbol{b}_i|_{\boldsymbol{\Delta}} = \boldsymbol{\Delta}'(\boldsymbol{\Delta}'\cdot\boldsymbol{b}_i) = \boldsymbol{\Delta}'\boldsymbol{\Delta}'^T\boldsymbol{b}_i.$$

To obtain the desired effect, this component is enlarged by a coefficient $\rho_e > 1$, so the expression for the new vector $\boldsymbol{b}'_i$ is

$$\begin{aligned}\boldsymbol{b}'_i &= \boldsymbol{b}_i + (\rho_e - 1)\boldsymbol{b}_i|_{\boldsymbol{\Delta}} \\ &= \boldsymbol{b}_i + (\rho_e - 1)\boldsymbol{\Delta}'\boldsymbol{\Delta}'^T\boldsymbol{b}_i \\ &= \boldsymbol{b}_i + (\rho_e - 1)\frac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\|\boldsymbol{\Delta}\|^2}\boldsymbol{b}_i \\ &= P\boldsymbol{b}_i\end{aligned} \tag{3}$$

where

$$P = \mathbf{I} + (\rho_e - 1)\frac{\boldsymbol{\Delta}\boldsymbol{\Delta}^T}{\|\boldsymbol{\Delta}\|^2}. \tag{4}$$

The fact of testing the function improvement on both $\boldsymbol{x}+\boldsymbol{\Delta}$ and $\boldsymbol{x}-\boldsymbol{\Delta}$ is called *double-shot strategy*: if the first sample $\boldsymbol{x}+\boldsymbol{\Delta}$ is not successful, the specular point $\boldsymbol{x}-\boldsymbol{\Delta}$ is considered. This choice drastically reduces the probability of generating two consecutive unsuccessful samples. The motivation is clear if one considers differentiable functions and small displacements: in this case the directional derivative along the displacement is proportional to the scalar product between displacement and gradient $\boldsymbol{\Delta}\cdot\nabla f$. If the first is positive, a change of sign will trivially cause a negative value, and therefore a decrease in f for a sufficiently small step size. The empirical validity for general functions (not necessarily differentiable) is caused by the correlations and structure contained in most of the functions corresponding to real-world problems. See Appendix A for a thorough analysis motivating the double-shot strategy.

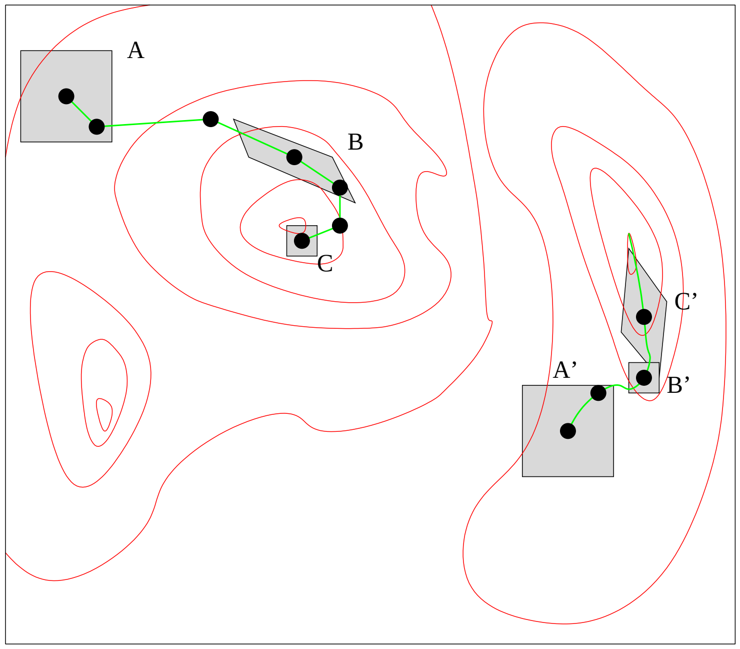


Figure 2: Affine Shaker geometry: two search trajectories leading to two different local minima. The evolution of the search regions is also illustrated.

An illustration of the geometry of the Reactive Affine Shaker algorithm is illustrated in Fig. 16, where the function to be minimized (in this case the domain is a square in $d = 2$ dimensions) is represented by a contour plot showing *isolines* at fixed values of f, and two trajectories (ABC and A'B'C') are plotted. The search regions are shown for some points along the search trajectory. A couple of independent vectors define the search region as a parallelogram centered on the point. The design criteria are given by an *aggressive search for local minima*: the search speed is increased when steps are successful (points A and A' in Figure 16), reduced only if no better point is found after the double shot. When a point is close to a local minimum, the repeated reduction of the search frame produces a very fast convergence of the search (point C in Figure 16). Note that another cause of reduction for the search region can be a narrow descent path (a "canyon", such as in point B' of Figure 16), where only a small subset of all possible directions improves the function value. However, once an improvement is found, the search region grows in the promising direction, causing a faster movement along that direction.

2.3 Termination and repeated runs

For most continuous optimization problems, an effective estimation of the number of steps required for identifying a global minimum is clearly impossible. Even when a local minimum is located, it is generally impossible to determine whether it is the global one or not, in particular if the knowledge about the function derives only from evaluations of $f(x)$ at selected points.

Because RASH does not include mechanisms to escape from local minima, it should be stopped as soon as the trajectory is sufficiently close to a local minimizer. Suitable termination criteria can be derived, for instance a single RASH run can be terminated if the step size $\|\boldsymbol{\Delta}\|$ is smaller than a threshold value ε for a predefined number S of consecutive steps. The step size, in fact, depends on the search box size, and the box tends to reduce its diameter in proximity of a local minimum because of repeated failures in improving the function value. A sequence of S consecutive steps is considered before termination to reduce the probability that, by chance, a small step is executed because of the randomized selection and not because the point is close to a local minimum.

By design, RASH searches for local minimizers and is stopped as soon as one is found. A simple way to continue the search after a minimizer is found is to restart from a different initial random point. This approach is equivalent to a "population" of RASH searchers where each member of the population is *independent*, completely unaware of what other members are doing.

In the experimental Section, parallel execution of RASH optimizers is considered.

Variable	Scope	Meaning
f	(input)	Function to minimize
ρ_e, ρ_r	(input)	Box expansion and reduction factors
$L_1, \dots, L_d, U_1, \dots, U_d$	(input)	Search range
$L'_1, \dots, L'_d, U'_1, \dots, U'_d$	(input)	Initialization range
d	(input)	Dimension of the space
n	(input)	Number of parallel optimizers
$\boldsymbol{b}_1, \dots, \boldsymbol{b}_d$	(internal)	Vectors defining search region $\mathcal{R}$ around $\boldsymbol{x}$
$\boldsymbol{x}$, $\boldsymbol{x}'$	(internal)	Current position, final position of run

```
1. function ParallelAffineShaker ($f$, $\rho_e$, $\rho_r$, $(L'_j)$, $(U'_j)$, $(L_j)$, $(U_j)$)
2.     $\forall j\ \boldsymbol{b}_j \leftarrow \frac{U_j - L_j}{4} \cdot \boldsymbol{e}_j$;
3.     pardo
4.         $\boldsymbol{x} \leftarrow$ random point $\in [L'_1, U'_1] \times \cdots \times [L'_d, U'_d]$;
5.         $\boldsymbol{x}$' $\leftarrow$ AffineShaker($f$, $\boldsymbol{x}$, ( $\boldsymbol{b}_j$), $\rho_e$, $\rho_r$);
6.     return best position found;
```

Figure 3: The Repeated RASH algorithm

3 Software structure

The main purpose of the developed software is to facilitate the comparison among different techniques and the integration of basic building blocks into more complex schemes. The basic software artifacts therefore correspond to a library of "solvers" and "test functions".

The library, written in C++, defines two pure virtual classes, `Function` and `Solver`, which expose all methods that are necessary for normal operation. A `Function` provides means for evaluating it, getting the number of dimensions of its domain, its search range; a `Solver` exposes methods for associating a `Function` to it and for executing an optimization step.

While specific implementations differ according to the particular function and solver, virtual definitions provide a convenient abstraction layer that allows every possible matching between specific solvers and function, while having a reasonably small main program to control both objects.

The main design features of the library are shown in Fig. 4. The pure virtual classes `Function` and `Solver` are inherited by specialized functions and solvers, respectively. Every specialized class has its own construction parameters, so object construction must be handled in a possibly different way for every specialized object. However, once a specific function object (e.g. a 4-dimensional Shekel 5 function) and a specific solver (e.g. RASH) have been created, the remaining code is completely independent of them, and only refers to the virtual classes. The function object is "registered" within the solver object (meaning that a reference to the function is stored inside the solver), then a single opti-

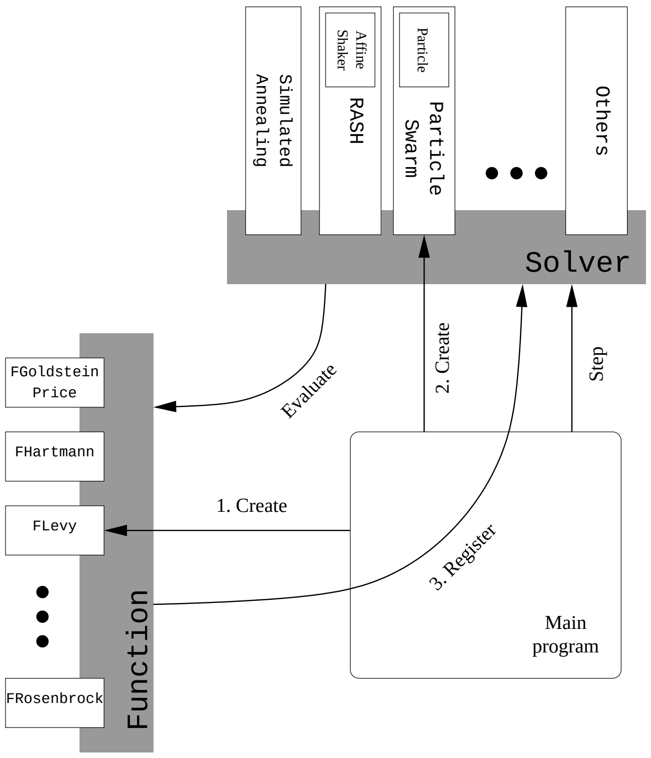


Figure 4: Overall object structure of the code.

```
template <class Targ, class Tres> class Function {
  private:
    mutable int fevals;
  protected:
    mutable Tres Result;
    virtual void body(const Targ &x) const = 0;
    inline Tres& eval (Targ &x) const
      { ++fevals; body(x); return Result; }
  public:
    const int dimension;
    const Targ minarg, maxarg;
    Function (int d) : dimension (d), fevals (0) {}
    virtual ~Function (void) {}
    const Tres& operator() (void) const { return Result;}
    const Tres& operator() (Targ &x) const { return eval(x); }
    int getEvaluations (void) const { return fevals; }
};
```

Figure 5: Pure virtual template `Function`: all functions are extension of this class and must implement the constructor, and the `body` method.

```
class FZakharov: public Function <vector<double>, double> {
  public:
    FZakharov(int = 30);
  protected:
    void body (const vector<double> &) const;
};
```

Figure 6: Definition of an $\mathbb{R}^d \to \mathbb{R}$ function.

mization step method can be repeatedly invoked by the main program. This optimization step will require the `Solver` object to evaluate the `Function` object, accessed through its reference.

An excerpt of the definition of the `Function` class is shown in Fig. 5. It is actually defined as a template whose arguments represent the input and output types of the represented function. The generic constructor initializes the number of dimensions and the counter of function evaluations. All extensions of the class are required to call the base function's constructor and define the pure virtual protected `eval()` method.

All data structures used by the library are based on the STL templates, so the input values are defined as `vector<double>`. Since the library is used for continuous optimization, a function mapping a subrange of $\mathbb{R}^d$ onto $\mathbb{R}$, shall be defined as in Fig. 6.

The `Solver` base class is shown in Fig. 7. The class only defines members

```
class Solver {
  protected:
    Function<vector<double>,double> &function;
    vector<double> last_evaluated_point;
    vector<double> best_point;
    double best_value;
  public:
    Solver (Function<vector<double>,double>&);
    virtual ~Solver (void) {}
    virtual const vector<double>& next (double&) = 0;
    const vector<double>& getLastPoint (void) const
        { return last_evaluated_point; }
    const vector<double>& getBestPoint (void) const
        { return best_point; }
    double getBestValue (void) const
        { return best_value; }
};
```

Figure 7: Pure virtual class `Solver`: all solvers are extensions of this class and must implemnt the constructor and the `next` method.

```
FZakharov f (10);
RASH r (f);
double v;
while ( f.getEvaluations() < MAXEVAL )
    r.next (v);
```

Figure 8: Excerpt of a main program performing local search.

that are common to all local search methods, namely a reference to the function to be minimized, the last point being evaluated, the point with the best evaluation and the corresponding function value. Currently, the registration of a function is performed at construction. The pure method `next()` must be implemented by child classes, and must be used to perform a single iteration of the local search procedure.

Because of the object-oriented approach and the use of virtual methods, the RASH solver can accept any function object that is defined as a mapping $\mathbb{R}^d \to \mathbb{R}$. Fig. 8 shows an example of code that can be used to actually perform an optimization run ff the chosen RASH optimizer on the Zakharov function in 10 dimensions. A complete example of usage is included in the software distribution package RASH v1.0.

4 Experimental results

Several tests have been performed in order to both validate and motivate the chosen strategy and to compare it with other state-of-the-art techniques. The proposed RASH algorithm has been tested on various classical test functions, whose analytical formulation and properties are reported in many optimization papers, see for example [4]. After some experiments on the success rate of the double-shot strategy (Section 4.1), and a discussion of some spurious effects during the initial phase of the search which are solved by our version (Section 4.2), we introduce the experimental results on the benchmark suite (Section 4.3), and an analysis of the effectiveness of the heuristic when compared with alternative techniques (Section 4.4).

4.1 Success rate of the double-shot strategy

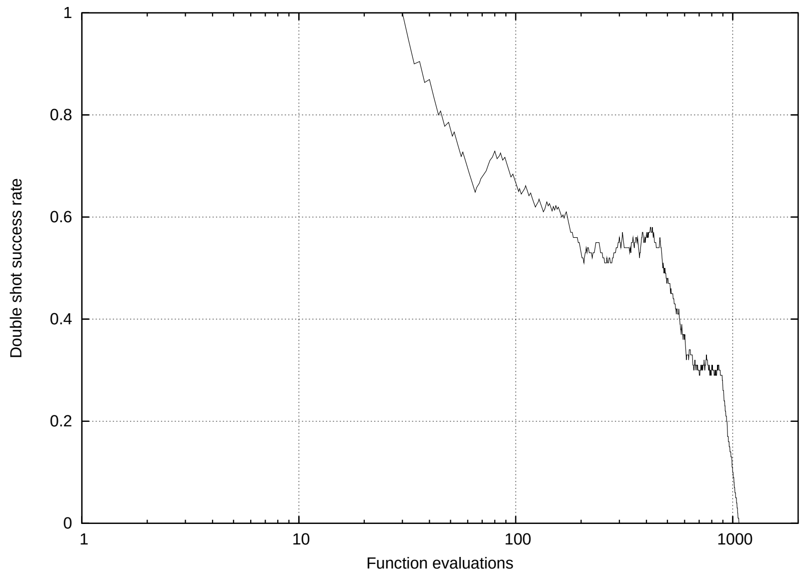
A measure of the effectiveness of the RASH heuristic can be the double-shot success rate during the search. In Appendix A we show that the success rate must be very high at the beginning, when the search region is very small; however, after the initial transient the rate should reduce, due to the fact that the algorithm operates by dynamically setting a balance between step size and success rate. Fig. 9 shows two representative cases. In both plots, the x axis reports the number of function evaluations, while the y axis reports a simple moving average of the rate of double-shot successes over the previous 100 steps on a representative run. After an initial transient when the success rate is very high due to the small size of the search region (which confirms the analysis of Appendix A), during a significant portion of the search the double-shot success ratio varies from 30% to 55%.

In the first plot, where a local minimum of a Shekel 4,5 function is reached, we observe a sharp reduction of success rate at the end. This happens when a local minimum is reached, and further improvement becomes impossible. The steep reduction of the double-shot success rate can therefore be used as a restart criterion. In the Rosenbrock case (lower plot), the system proceeds slowly towards better values, by following the very narrow valley leading to the global optimum. The success rate remains close to 50% during the descent.

These results confirm the "aggressive" attitude of the RASH heuristic, whose search region $\mathcal{R}$ is continuously adjusted to allow steps as large as possible while still ensuring a large success probability of each double shot trial.

4.2 Influence of initial conditions

An issue that deserves further study is the dependence of the search strategy on the initial conditions. In particular, the choice of the first displacement vector $\mathbf{\Delta}$ may influence the subsequent behavior in an improper way, because on successful moves the search box will expand along $\mathbf{\Delta}$, thus influencing the next choice of $\mathbf{\Delta}$, and so on. At the beginning, with a very small search region, the double-shot strategy succeeds with probability close to one, and the local



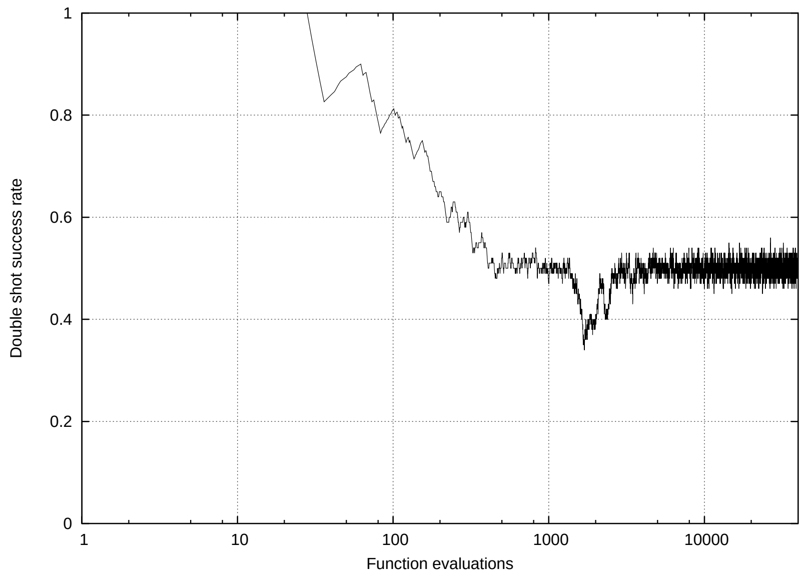


Figure 9: Success rate of the double-shot strategy for a Shekel 4,5 (top) and a Rosenbrock 10 (bottom) search.

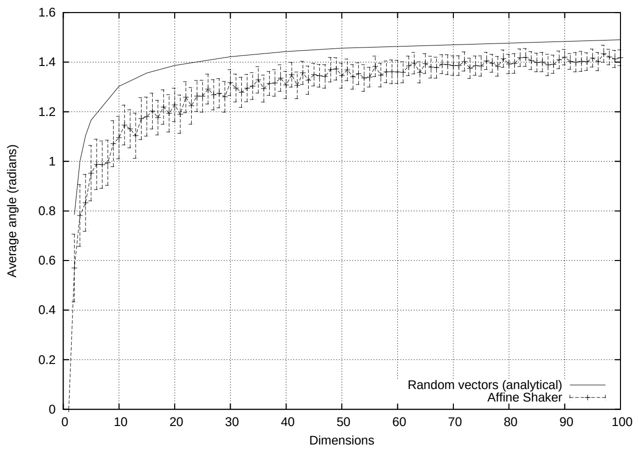


Figure 10: Angle between the first and the tenth move. The solid line represents the theoretical angle if the two directions were random (see Appendix B), the dashed series is obtained after 10 subsequent successes of the double shot procedure.

characteristics of the f function do not have a chance of influencing the evolution of the search region in an effective way. The successes depend on the very small size of the box more than on the local properties.

In order to study this effect, we simulated the algorithm's behavior during a sequence of 10 successful applications of the double-shot procedure, by repeatedly generating a $\boldsymbol{\Delta}$ vector in the search box $\mathcal{R}$, then updating $\mathcal{R}$ according to equation (3). Finally, the angle between the directions of the first and the tenth value of $\boldsymbol{\Delta}$ is computed.

Fig. 10 shows the average and standard error of 100 runs on different problem dimensions. The continuous plot shows the theoretically calculated average angle between two random directions, as obtained in Appendix B. It can be seen how a memory effect can be detected after 10 iterations. The initial and final directions are correlated, not random. As expected, they tend to be collinear. Because the box elongation can be very large (the search region becomes "needle-like"), a potentially large number of iterations can be spent to adapt it to the structure of the local attraction basin.

In order to avoid this spurious effect, as mentioned, in the initial phase of RASH the box is enlarged in an isotropous manner, by multiplying each basis vector $\boldsymbol{b}_i$ by the same ρ_e factor, until the first lack of success is encountered and the affine transformation (3) is used.

Table 1: Number of successes, average function evaluations and average minimum found for 100 optimization runs on the test functions.

f	d	Success	Evals	CPU time	ΔMin
Goldstein-Price	2	76	476	0.121	$2.85 \cdot 10^{-3}$
Hartmann d,4	3	96	2227	1.73	$4.26 \cdot 10^{-4}$
	6	63	257	0.995	$2.24 \cdot 10^{-3}$
Shekel 4,5	4	35	170	0.338	0.261
Shekel 4,7	4	31	306	0.542	0.370
Shekel 4,10	4	30	164	0.362	0.438
Zakharov	10	100	2473	13.9	$9.46 \cdot 10^{-7}$
	20	100	12259	464	$9.86 \cdot 10^{-7}$
	50	100	83605	42099	$9.95 \cdot 10^{-7}$
	100	100	260358	1843765	$9.86 \cdot 10^{-7}$
Rosenbrock	3	100	3595	2.06	$7.98 \cdot 10^{-7}$
	4	65	12085	11.0	$9.33 \cdot 10^{-5}$
	5	1	15122	—	$1.54 \cdot 10^{-3}$

4.3 Benchmarks

This section reports results obtained by running the RASH algorithm on a benchmark suite of various classical test functions (see for example [4] for definitions).

Table 1 shows the results for 100 independent optimization runs for each function. A run is considered successful if the heuristic finds a point x such that

$$f(x) - f_{\text{min}} < \varepsilon_{\text{rel}}|f_{\text{min}}| + \varepsilon_{\text{abs}} \tag{5}$$

where f_{min} is the known global minimum. Following [4], we have set $\varepsilon_{\text{rel}} = 10^{-4}$ and $\varepsilon_{\text{abs}} = 10^{-6}$. Runs are stopped, and lack of success is recorded, if the global minimum is not located after $5000d$ function evaluations. In the experiments, the termination criteria described in Section 2.3 are disabled in order to evaluate the effectiveness of RASH when applied to the known test cases. The only retained criterion is the maximum number of iterations, while the execution is artificially interrupted, for the experimenters' convenience, when the known global minimum is located with the given degree of accuracy.

The number of successful runs is shown in column *Success*. The average number of function evaluations required in successful runs is shown in column *Evals*, (unsuccessful runs are truncated). Column Δ*Min* reports the average value of the difference between the found minimum and the actual global minimum achieved by all 100 runs, including unsuccessful ones.

Column *CPU time* reports the average execution time of successful runs, given in standard CPU time units as defined in [6].

It can be noted that, for some functions like Goldstein-Price, Hartmann, and Zakharov, the success rate is large and the number of function evaluations is comparable to, and in some cases better than, the number of function evaluations used by more complex techniques like Enhanced Simulated Annealing, see

Table 2: Number of successes, average function evaluations and average minimum found for 100 optimization runs on the test functions on $2d$ parallel threads.

f	d	Threads	Success	Evals	CPU time	ΔMin
Goldstein-Price	2	4	99	337	0.152	$1.25 \cdot 10^{-4}$
Hartmann d,4	3	6	100	856	0.556	$2.55 \cdot 10^{-4}$
	6	12	100	2420	5.38	$2.41 \cdot 10^{-4}$
Shekel 4,5	4	8	93	1296	1.28	$1.2 \cdot 10^{-3}$
Shekel 4,7	4	8	94	1323	1.28	$1.03 \cdot 10^{-3}$
Shekel 4,10	4	8	85	1336	1.34	$2.53 \cdot 10^{-3}$

for comparison Table I of [13]. These results confirm that the standard behavior of RASH is to rapidly locate a local minimum in the basin of attraction where the initial point lies. However, by design, RASH has no mechanism to escape local minima after they are identified. Therefore it is not surprising that the percentage of success is lower for other functions (like for example for Shekel). While the RASH algorithm shows a good performance on most test functions, a very ill-conditioned problem, such as the Rosenbrock function, is solved in a satisfactory way only for a small number of dimensions. The effects of high dimensionality are also apparent on the *CPU time* column of Table 1. Due the relative simplicity of the benchmark functions, the dominating factor for a large number of dimensions is the affine transformation of the search region vectors, amounting to d vector multiplications by a $d \times d$ matrix, totaling to an $O(d^3)$ time per optimization step. For high-dimensional problems and functions requiring small computation more specialized techniques like ESA of [13] should be considered. In any case, let's note that many functions are extremely costly to compute, see for example evaluations requiring the simulation of an industrial plant or a real-world experimentation.

In order to obtain higher success rates, we exploited the fast convergence speed of the successful runs by *parallelizing* independent solvers on the same function. Considering independent repetitions is in fact the simplest way to use the simple RASH component to obtain a more robust scheme. In this case, an optimization session is achieved by iterating through $2d$ independent solvers (where d is the dimension of the function's domain) until either one of the solvers finds a value that satisfies equation (5) or the maximum number of function evaluations is reached (counting the total number of evaluations by all independent threads).

The results of interest, where parallelization actually leads to an improvement, are shown in Table 2. Note that most problem instances benefit from parallel search. However, highly dimensional problems such as Zakharov are already solved with a single thread. In this case, a single solver is more effective, and the success rate for a fixed number of iterations decreases if parallel threads are exploited.

Table 3: Comparison with other techniques — number of successful minimizations; see text for the description of the techniques

Method	$G. - P.$	H_3	H_6	$S_{4,5}$	$S_{4,7}$	$S_{4,10}$
RASH	99	100	100	93	94	85
ECTS	100	100	100	75	80	75
ESA	100	100	100	54	54	50
ISA 1	n.a.	99	97	7	1	3
ISA 2	n.a.	100	0	19	28	18

4.4 Comparison with other techniques

The RASH algorithm behavior has been compared with other local search heuristics, and results are presented in Table 3. In particular, we focused on two recent proposals, Enhanced Simulated Annealing [13] and Enhanced Continuous Tabu Search [4], which, like RASH, aim at minimizing functions of continuous variables. Another classical proposal, the Improved Simulated Annealing algorithm [15] is shown in two different variants (wide and narrow search domain). Techniques have been selected on the basis of similar hypotheses (continuous functions, no analytical tools other than evaluation) and on similar criteria for estimating success and efficiency.

For comparison purposes, we rely on data provided in [4] and on the original sources. The definition of "successful" run takes into account the criteria defined in [4, 13], where the maximum number of allowed evaluations is $5000d$, as described before. For RASH, we chose to employ the multi-thread results shown on Table 2.

The results clearly show that the RASH heuristic achieves state-of-the-art results on various classical problems, ranging from 85% to 100% successful minimizations. This result is of interest because of the simplicity of the technique, which can be an effective building block for more complex heuristic schemes.

5 Conclusions

We proposed and analyzed the Reactive Affine Shaker adaptive random search algorithm based on function evaluations, which builds on a seminal proposal of [1]. The main algorithmic contributions of this paper consist of a careful analysis of the double-shot strategy motivating and quantifying what was originally proposed based on intuition, the proposal of a modified initial phase to avoid a dangerous effect during the initial growth of the search region and the evaluation of a simple "portfolio" consisting of independent runs of the local searcher.

Last but not least, the paper presents a design and implementation of modular and reusable software components to facilitate the experimentation with the technique and the development of more advanced schemes by starting from algorithmic building blocks. The software package is made available to researchers as RASH v 1.0 for non-profit usage at the URL `http://netmob.unitn.it/software/RASH.tar.gz`

The conclusions of the experiments show a performance which is in some cases comparable or better w.r.t. competitive techniques. The results are unexpected given the algorithmic simplicity of RASH, in particular its design based on converging rapidly to the local minimizer in the attraction basin of the initial point and are due to a rapid and effective adaptation of the search region based on feedback from function evaluations at random points. This work motivates the consideration of this component for more complex meta-heuristic schemes.

References

[1] Roberto Battiti and Giampietro Tecchiolli. Learning with first, second and no derivatives: A case study in high energy physics. *Neurocomp*, 6:181–206, 1994.

[2] Roberto Battiti and Giampietro Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[3] Roberto Brunelli and Giampietro Tecchiolli. On random minimization of functions. *Biological Cybernetics*, 65(6):501 – 506, 1991.

[4] Rachid Cheluoah and Patrick Siarry. Tabu search applied to global optimization. *European Journal of Operational Research*, 123:256–270, 2000.

[5] Angelo Corana, Michele Marchesi, Claudio Martini, and Sandro Ridella. Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm. *ACM Trans. Math. Softw.*, 13(3):262–280, 1987.

[6] Lawrence C. W. Dixon and Gábor P. Szegő, editors. *Towards Global Optimization 2*. North Holland, Amsterdam, The Netherlands, 1978.

[7] Fred W. Glover and Gary A. Kochenberger. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research and Management Science*. Kluwer Academic Publishers, Norwell, MA, 2003.

[8] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, Boston, MA., 1989.

[9] Carla P. Gomes and Bart Selman. Algorithm portfolios. *Artif. Intell.*, 126(1-2):43–62, 2001.

[10] Robert Hooke and T. A. Jeeves. "direct search" solution of numerical and statistical problems. *J. ACM*, 8(2):212–229, 1961.

[11] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search Foundations and Applications*. Morgan Kaufmann / Elsevier, 2004.

[12] Panos M. Pardalos and Mauricio G. C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, NY, USA, 2002.

[13] Patrick Siarry, Gérard Berthiau, François Durbin, and Jacques Haussy. Enhanced simulated annealing for globally minimizing functions of many-continuous variables. *ACM Transactions on Mathematical Software*, 23(2):209–228, June 1997.

[14] Francisco J. Solis and Roger J.-B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[15] Ah C. Tsoi and M. Lim. Improved simulated annealing technique. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, pages 594–597, Piscataway, NJ (USA), 1988. IEEE Press.

[16] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997.

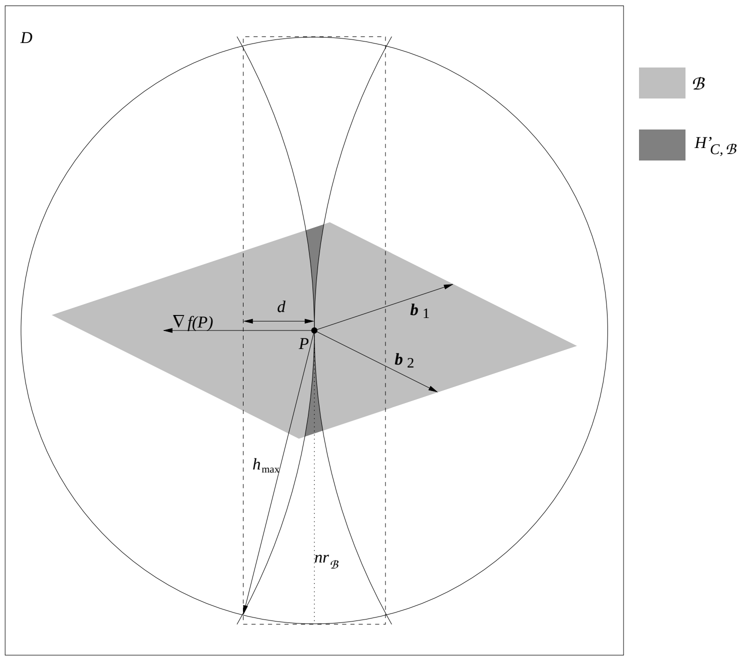


Figure 11: A description of the settings of Theorem A.1. Note that $H_{\mathcal{B}}$ is a subset of $H'_{C,\mathcal{B}}$.

APPENDIX

A Double-shot success probability

If the function f is linear, an increase of the function value at the displaced point $\boldsymbol{x} + \boldsymbol{\Delta}$ implies a decrease of the value at the specular point $\boldsymbol{x} - \boldsymbol{\Delta}$, and therefore the "double shot" strategy is trivially bound to be successful.

The intuition supporting strategy for a general function is that, if the function f is smooth, it can be approximated around a given point by a tangent hyperplane with a good accuracy for small displacements. The "double shot" should therefore be successful with a high probability if the search region, and therefore the displacement, becomes very small. The purpose of this section is to analyze in detail the probability of the strategy used in RASH.

Let $\mathcal{B}$ be the current box, shown in light grey in Figure 11:

$$\mathcal{B} = \left\{ \sum_{i=1}^{n} c_i \boldsymbol{b}_i : \forall i \quad -1 \le c_i \le 1 \right\}. \tag{6}$$

Let us define $r_{\mathcal{B}} = \max\{\|\boldsymbol{b}_1\|, \dots, \|\boldsymbol{b}_n\|\}$ the *radius* of the box. Of course, the box $\mathcal{B}$ is contained in the circle with radius $nr_{\mathcal{B}}$.

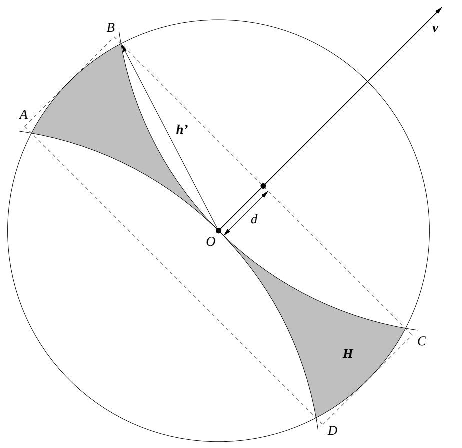


Figure 12: Description of the setting of Lemmata A.2 and A.3.

Let $H_{\mathcal{B}}$ be the subset of $\mathcal{B}$ where the double shot strategy does not succeed:

$$H_{\mathcal{B}} = \{\boldsymbol{h} \in \mathcal{B} : f(P + \boldsymbol{h}) \geq f(P) \wedge f(P - \boldsymbol{h}) \geq f(P)\}. \tag{7}$$

In Figure 11 the set $H_{\mathcal{B}}$ is contained in the dark grey area, whose exact meaning shall be made clear later. We want to show that as the box becomes smaller and smaller the probability of failure of the double shot strategy tends to zero. Since the choice of the vector $\boldsymbol{h} \in \mathcal{B}$ is uniform, we just need to show that the ratio between the measure of $H_{\mathcal{B}}$ and the measure of $\mathcal{B}$ tends to zero.

More formally, we want to prove the following.

Theorem A.1. *Let $D \subseteq \mathbb{R}^n$, a point $P \in D$, a function $f : D \to \mathbb{R}$ continuous in P with continuous first partial derivatives, a real constant $K > 0$.*

Then, for every $\varepsilon \in \mathbb{R}$, $\varepsilon > 0$, there exists $\delta \in \mathbb{R}$ such that, for every set of vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n \in \mathbb{R}^n$, defining the box $\mathcal{B}$ (where $P + \mathcal{B} \subseteq D$) with $r_{\mathcal{B}} < \delta$ and $\text{measure}(\mathcal{B}) \geq K r_{\mathcal{B}}^n$, *we have*

$$\frac{\text{measure}(H_{\mathcal{B}})}{\text{measure}(\mathcal{B})} < \varepsilon.$$

In order to prove Theorem A.1 we need the following lemma (see Figure 12 for a visual representation):

Lemma A.2. *Let $\boldsymbol{v} \in \mathbb{R}^n$, let $H = \{\boldsymbol{h} : \|\boldsymbol{h}\| \leq 1 \wedge |\boldsymbol{h} \cdot \boldsymbol{v}| \leq \|\boldsymbol{h}\|^2\}$. If $\boldsymbol{h}' \in \mathbb{R}^n$ is such that $\|\boldsymbol{h}'\| = 1$ and $\boldsymbol{h}' \cdot \boldsymbol{v} = C\|\boldsymbol{h}\|^2 = C$, then H lies outside the cone generated by rotating $\boldsymbol{h}'$ around $\boldsymbol{v}$.*

In other words, all vectors in H are "more perpendicular" than $\boldsymbol{h}'$ with respect to $\boldsymbol{v}$.

Proof. We just need to show that for every $\boldsymbol{h} \in H$ the normalized projection of $\boldsymbol{h}$ on $\boldsymbol{v}$, i.e., the cosine of their angle, is smaller than that between $\boldsymbol{h}'$ and $\boldsymbol{v}$. Indeed,

$$\frac{|\boldsymbol{h} \cdot \boldsymbol{v}|}{\|\boldsymbol{h}\|\|\boldsymbol{v}\|} \leq \frac{C\|\boldsymbol{h}\|^2}{\|\boldsymbol{h}\|\|\boldsymbol{v}\|} = \frac{C\|\boldsymbol{h}\|}{\|\boldsymbol{v}\|} \leq \frac{C}{\|\boldsymbol{v}\|} = \frac{\boldsymbol{h}' \cdot \boldsymbol{v}}{\|\boldsymbol{h}'\|\|\boldsymbol{v}\|}.$$

□

This leads to the following corollary (again, see Figure 12 for a visual representation):

Lemma A.3. *Let $\boldsymbol{v} \in \mathbb{R}^n$, let $H = \{\boldsymbol{h} : \|\boldsymbol{h}\| \leq 1 \wedge \boldsymbol{h} \cdot \boldsymbol{v} \leq \|\boldsymbol{h}\|^2\}$. If $\boldsymbol{h}' \in \mathbb{R}^n$ is such that $\|\boldsymbol{h}'\| = 1$ and $\boldsymbol{h}' \cdot \boldsymbol{v} = C\|\boldsymbol{h}\|^2 = C$, then H lies in the n-dimensional cylinder centered in the origin, with axis along the direction of vector $\boldsymbol{v}$, having height $2d$, where*

$$d = \frac{\boldsymbol{h}' \cdot \boldsymbol{v}}{\|\boldsymbol{v}\|}$$

and $(n-1)$-dimensional basis of radius 1.

Proof. Such cylinder is the set of vectors $\boldsymbol{w}$ such that the projection of vector $\boldsymbol{w}$ along the direction of $\boldsymbol{v}$ is less than d, and the norm of the component of $\boldsymbol{w}$ perpendicular to $\boldsymbol{v}$ is less than 1:

$$X_{\boldsymbol{v},d} = \left\{ \boldsymbol{w} \in \mathbb{R}^n : \frac{\boldsymbol{w} \cdot \boldsymbol{v}}{\|\boldsymbol{v}\|} \leq d \wedge \|\boldsymbol{w}\|^2 - \left(\frac{\boldsymbol{w} \cdot \boldsymbol{v}}{\|\boldsymbol{v}\|} \right)^2 \leq 1 \right\}.$$

Both conditions are fulfilled for all $\boldsymbol{w} \in H$. In fact, by Lemma A.2,

$$\boldsymbol{w} \in H \qquad \Rightarrow \qquad \frac{\boldsymbol{w} \cdot \boldsymbol{v}}{\|\boldsymbol{v}\|} \leq \frac{\boldsymbol{h} \cdot \boldsymbol{v}}{\|\boldsymbol{h}\|\|\boldsymbol{v}\|} \leq \frac{\boldsymbol{h}' \cdot \boldsymbol{v}}{\|\boldsymbol{h}'\|\|\boldsymbol{v}\|} = \frac{\boldsymbol{h}' \cdot \boldsymbol{v}}{\|\boldsymbol{v}\|} = d.$$

and

$$\boldsymbol{w} \in H \qquad \Rightarrow \qquad \|\boldsymbol{w}\| \leq 1 \qquad \Rightarrow \qquad \|\boldsymbol{w}\|^2 - \left(\frac{\boldsymbol{w} \cdot \boldsymbol{v}}{\|\boldsymbol{v}\|} \right)^2 \leq 1.$$

□

The last lemma enables us to find a convenient upper bound on the measure of the set $H_{\mathcal{B}}$.

Proof of Theorem A.1. Since f has continuous first derivatives, we have

$$\begin{aligned} f(P + \boldsymbol{h}) &= f(P) + \boldsymbol{h} \cdot \nabla f(P) + \sigma(\boldsymbol{h}), \\ f(P - \boldsymbol{h}) &= f(P) - \boldsymbol{h} \cdot \nabla f(P) + \sigma(-\boldsymbol{h}), \end{aligned}$$

where

$$\sigma(\boldsymbol{h}) = O(\|\boldsymbol{h}\|^2). \tag{8}$$

Let $\sigma'(\boldsymbol{h}) = \max\{\sigma(\boldsymbol{h}), \sigma(-\boldsymbol{h})\}$. Then (7) implies:

$$H_{\mathcal{B}} \subseteq H'_{\mathcal{B}} = \{\boldsymbol{h} \in \mathcal{B} : |\boldsymbol{h} \cdot \nabla f(P)| \leq \sigma'(\boldsymbol{h})\}. \tag{9}$$

Equation (8) is still valid for σ', therefore we can find constants C and r_0 such that $\sigma'(\boldsymbol{h}) \leq C\|\boldsymbol{h}\|^2$ as soon as $\|\boldsymbol{h}\| \leq r_0$. Therefore,

$$\forall \mathcal{B}, r_{\mathcal{B}} \leq r_0, \qquad H_{\mathcal{B}} \subseteq H'_{C,\mathcal{B}} = \left\{\boldsymbol{h} \in \mathcal{B} : |\boldsymbol{h} \cdot \nabla f(P)| \leq C\|\boldsymbol{h}\|^2\right\}.$$

Figure 11 shows the set $H'_{C,\mathcal{B}}$ in dark grey.

Given a box $\mathcal{B}$ having $r_m B \leq r_0$ and constrained by the theorem's hypothesis, let us choose a vector $\boldsymbol{h}_{\max}$ such that $\|\boldsymbol{h}_{\max}\| = nr_{\mathcal{B}}$ (so that its "tip" lies on the sphere) and $\boldsymbol{h}_{\max} \cdot \nabla f(P) = C(nr_{\mathcal{B}})^2$ (so that it lies at the border of the set $H'_{C,\mathcal{B}}$).

As proved in Lemma A.3, $H'_{C,\mathcal{B}}$ is contained in the n-dimensional cylinder $P + nr_{\mathcal{B}} \cdot X_{\nabla f(P), \frac{d}{nr_{\mathcal{B}}}}$, i.e. centered in P, with axis directed as $\nabla f(P)$ having base radius $nr_{\mathcal{B}}$ and height $2d$, where

$$d = \frac{\boldsymbol{h}_{\max} \cdot \nabla f(P)}{\|\nabla f(P)\|} = \frac{Cnr_{\mathcal{B}}^2}{\|\nabla f(P)\|}$$

is the projection of $\boldsymbol{h}_{\max}$ along the direction of $\nabla f(P)$. Consequently, whenever $r_{\mathcal{B}} \leq r_0$, as $H_{\mathcal{B}} \subseteq H'_{C,\mathcal{B}} \subseteq P + nr_{\mathcal{B}} \cdot X_{\nabla f(P), \frac{d}{nr_{\mathcal{B}}}}$,

$$\text{measure}(H_{\mathcal{B}}) \leq M(nr_{\mathcal{B}})^{n-1} \cdot 2d = \frac{2MCn^{n-1}}{\|\nabla f(P)\|} r_{\mathcal{B}}^{n+1},$$

where M is the measure of the $(n-1)$-dimensional sphere with unit radius.

Therefore,

$$\frac{\text{measure}(H_{\mathcal{B}})}{\text{measure}(\mathcal{B})} \leq \frac{2MCn^{n-1}}{\|\nabla f(P)\|} r_{\mathcal{B}}^{n+1} \cdot \frac{1}{Kr_{\mathcal{B}}^n} = \frac{2MCn^{n-1}}{K\|\nabla f(P)\|} r_{\mathcal{B}},$$

therefore, given $\varepsilon > 0$, it is sufficient to let

$$\delta = \min\left\{r_0, \frac{K\|\nabla f(P)\|\varepsilon}{2MCn^{n-1}}\right\}$$

to obtain the thesis. □

Note that the hypothesis constraining the measure of $\mathcal{B}$ to be greater of $Kr_{\mathcal{B}}^n$ for some constant $K > 0$ is necessary in order to avoid degenerate cases where two vectors generating the box tend to align.

B Angle between random directions in $\mathbb{R}^d$

As we mentioned in Section 2, if RASH starts with a very small and isotropic search region and enjoys an uninterrupted sequence of successes afterwards

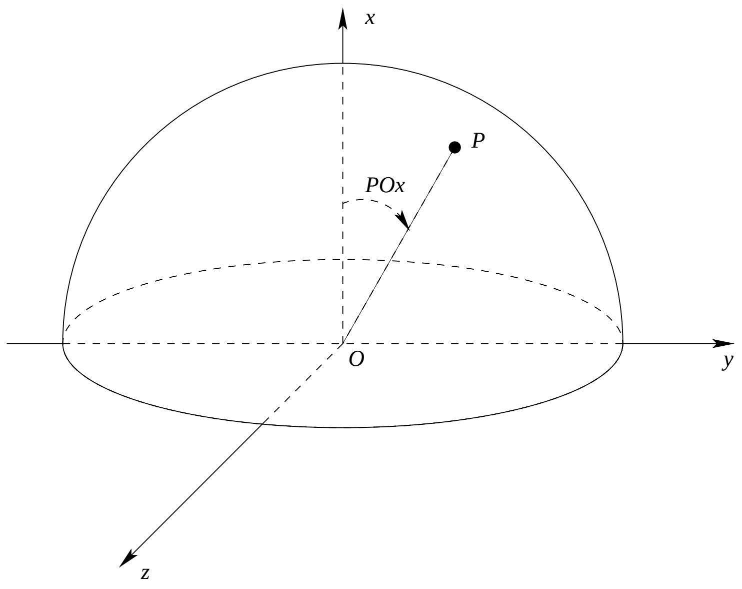


Figure 13: Angle $\widehat{POx}$ between a random point in the positive-x hemisurface and the positive x axis

(caused by the fact that the search region is very small and not by the fact that the chosen directions are suited to the local attraction basin), the average direction obtained after some steps can be very different from a random direction as it can "remember" the initial step. In fact, the affine transformation will elongate the region along the first direction, and therefore the second directions will tend to be approximately collinear with the first one, an effect that will continue for the future iterations. In order to quantify this initial "memory effect", it is of interest to compare the probability distribution of directions obtained after a sequence of affine expansions with a uniform probability.

The problem we are addressing is therefore the following one:

Problem B.1. *If we draw two random lines in $\mathbb{R}^d$ intersecting at the origin, what is the average angle between them?*

Of the two possible supplementary *angles*, we always choose the one which is less than $\pi/2$, otherwise the answer would always be zero.

The problem can also be stated as follows.

Problem B.2. *Given a d-dimensional hypersphere, consider the positive-x hemisphere (the case $d = 3$ is shown in Fig. 13). Choose a random point P on the surface, i.e., a point $P = (x, y, z, \dots) \in \mathbb{R}^d$ such that $x \geq 0$ and $\|P\| = 1$. What is the average value for the angle $\widehat{POx}$?*

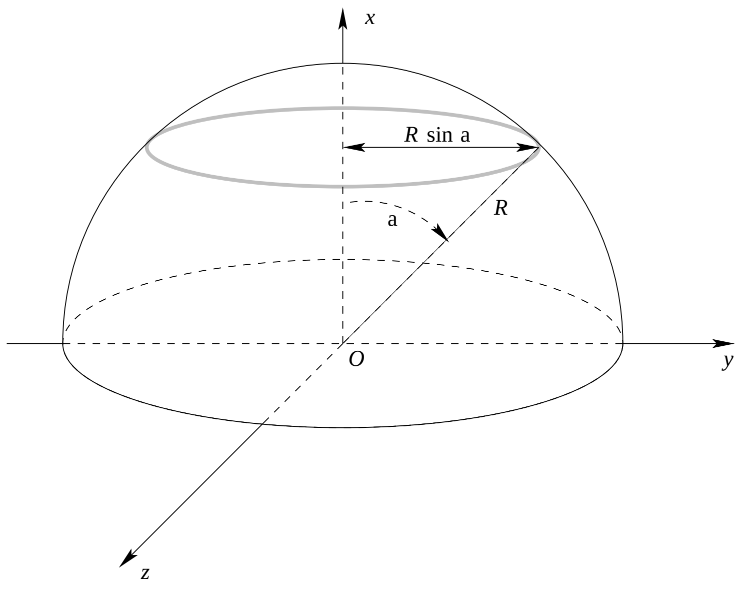


Figure 14: The ring whose integration provides the hemisurface.

B.1 An inductive expression for the hemisurface

Let $S_d(R)$ be the value of the hemisurface (half the surface of the sphere) for a given number d of dimensions and a given radius R of the d-dimensional hypersphere.

Then, the hemisurface of the $d+1$-dimensional hypersphere of radius R can be obtained by integrating the grey ring surface of Fig. 14 for α moving from 0 (the upper point) to $\pi/2$ (the equator):

$$S_{d+1}(R) = \int_0^{\frac{\pi}{2}} 2S_d(R\sin\alpha)R\,\mathrm{d}\alpha. \tag{10}$$

Consider in fact that the grey ring has radius $R\sin\alpha$, and thus its perimeter is $2S_d(R\sin\alpha)$ (twice the hemiperimeter) and its "width" (in the $d+1$-th dimension) is equal to $R\,\mathrm{d}\alpha$, since α is expressed in radians.

Notice that the hemisurface of the d-hypersphere is also proportional to the $d-1$-th power of R (it is a $d-1$-dimensional variety):

$$S_d(R) = C_d R^{d-1} \tag{11}$$

for some positive real constant C_d. This expression shall be useful in the following.

B.2 The average angle

Equation (10) is very helpful in calculating the average value of α, which we are looking for. In fact, let $\bar{\alpha}_d$ be the average value of α in d dimensions. Then

$$\bar{\alpha}_d = \frac{\displaystyle\int_S \widehat{POx}\,\mathrm{d}S}{|S|} \tag{12}$$

where S is the hemisurface, the point P scans S and $|S|$ is the measure of S. Consider that angle $\widehat{POx}$ is precisely the angle α of equation (10), and that it is constant within the same ring; then, the probability distribution function of the angle $\widehat{POx}$ is

$$f_d(\alpha) = \frac{2S_d(R\sin\alpha)R}{\displaystyle\int_0^{\frac{\pi}{2}} 2S_d(R\sin\alpha)R\,\mathrm{d}\alpha}, \tag{13}$$

and equation (12) becomes

$$\bar{\alpha}_d = \int_0^{\frac{\pi}{2}} \alpha f_d(\alpha)\,\mathrm{d}\alpha = \frac{\displaystyle\int_0^{\frac{\pi}{2}} \alpha\cdot 2S_d(R\sin\alpha)R\,\mathrm{d}\alpha}{\displaystyle\int_0^{\frac{\pi}{2}} 2S_d(R\sin\alpha)R\,\mathrm{d}\alpha}.$$

Considering the due simplifications and equation (11), we get

$$\bar{\alpha}_d = \frac{J_{d-2}}{I_{d-2}}, \tag{14}$$

where

$$I_d = \int_0^{\frac{\pi}{2}} \sin^d\alpha\,\mathrm{d}\alpha \tag{15}$$

$$J_d = \int_0^{\frac{\pi}{2}} \alpha\sin^d\alpha\,\mathrm{d}\alpha. \tag{16}$$

With the notation introduced by equations (15) and (16), the probability density function (13) can be written as

$$f_d(\alpha) = \frac{\sin^{d-2}\alpha}{I_{d-2}}.$$

B.2.1 Determining I_d and J_d

The value of I_d can be obtained by straightforward calculations (integration by parts) leading to the following inductive expression:

$$I_d = \begin{cases} \frac{\pi}{2} & \text{if } d = 0 \\ 1 & \text{if } d = 1 \\ \frac{(d-1)I_{d-2}}{d} & \text{if } d \geq 2. \end{cases} \tag{17}$$

Table 4: Values of $\bar{\alpha}_d$ for increasing dimensions

d	$\bar{\alpha}_d$ (radians)	$\bar{\alpha}_d$ (degrees)
2	0.785398	45.0000
3	1.000000	57.2958
4	1.103708	63.2378
5	1.166667	66.8451
10	1.302778	74.6437
15	1.356416	77.7169
20	1.387008	79.4697
30	1.422228	81.4877
40	1.442770	82.6646
50	1.456625	83.4584
100	1.490539	85.4016

The value of J_d is obtained by an analogous procedure and has a similar expression:

$$J_d = \begin{cases} \frac{\pi^2}{8} & \text{if } d = 0 \\ 1 & \text{if } d = 1 \\ \frac{(d-1)J_{d-2}}{d} + \frac{1}{d^2} & \text{if } d \geq 2. \end{cases} \tag{18}$$

B.3 Values of $\bar{\alpha}_d$ for different values of d

While the analytical expression for $\bar{\alpha}_d$ is possible, and all values are in $\mathbb{Q}(\pi)$, it is easier to write a short C program to print out some values. Table 4 reports some representative values.