RASH: A Self-adaptive Random Search Method

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

Abstract

This paper presents an adaptive stochastic search algorithm for the optimization of functions of continuous variables where the only hypothesis is the pointwise computability of the function.

The main design criterion of the proposed 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 uniform probability. Heuristic adaptation of the step size and direction allows the largest possible movement per function evaluation. Our results show that the proposed technique is, in spite of its simplicity, a promising building block to consider for the development of more complex optimization algorithms, particularly in those cases where the objective function evaluation is expensive.

Keywords — Stochastic local search, Function optimization, Reactive search.

1 Introduction

Significant effort has been spent during the last decades to the problem of finding the global minimum of a function of continuous variables [7]. Indeed, many real-world problems can be modeled as functions of real-valued parameters; however, such models are not always analytical, and the evaluation of objective functions on a given set of parameter values may involve costly physical experiments. It is therefore important to locate optimal points with as few function evaluations as possible.

While no method can guarantee a desired accuracy within predictable computing times for all functions [11], most real-world problems are characterized by a rich structure underlying candidate solutions, and local search techniques take advantage of this fact by assuming that a better solution can usually be found in the *neighborhood* of the current tentative solution: after starting from an initial configuration of the independent variables $\boldsymbol{x}^{(0)}$, a *search trajectory*, where point $\boldsymbol{x}^{(t+1)}$ is chosen in the neighborhood of point $\boldsymbol{x}^{(t)}$, is generated. Under suitable conditions, the trajectory will converge to a *local minimizer*. The set of initial points for trajectories that converge to the same local minimizer 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 [8].

Current research considers tailoring the appropriate set of basic algorithmic building blocks 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][1]. Each component demands a careful design and a sound statistical analysis as an isolated part before considering integration in more complex schemes.

[1]see also the web site www.reactive-search.org.

Variable	Meaning
f	Function to minimize
$\boldsymbol{x}$	Initial point
$\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$	Vectors defining search region $\mathcal{R}$ around $\boldsymbol{x}$
ρ_e, ρ_r	Box expansion and reduction factors
t	Iteration counter
$\mathbf{P}$	Transormation matrix
$\boldsymbol{x}$, $\boldsymbol{\Delta}$	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 $\boldsymbol{x}$;
```

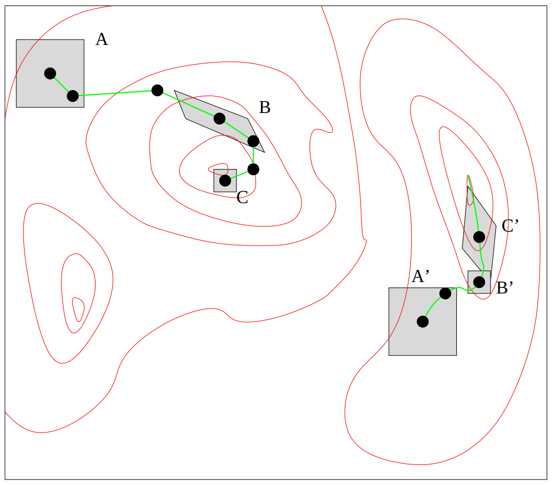

Figure 1: Affine Shaker algorithm (left) and geometry (right).

In this paper we describe the "Reactive Affine Shaker" (RASH) method, based on the stochastic (or random) local search framework [9]. The seminal idea of the scheme was presented for a specific application in neural computation [1], while a complete description can be found in [3]. The method has been implemented in the C++ language, and its applicability is demonstrated on a widely used set of benchmark functions.

2 The Reactive Affine Shaker Algorithm

The Reactive Affine Shaker algorithm (RASH for short) is an adaptive random search algorithm based on function evaluations alone. Let $f : D \to \mathbb{R}$ be the function to be minimized, and let $D \subseteq \mathbb{R}^d$ be its d-dimensional domain.

Figure 1 shows the behavior of the algorithm, both by presenting its pseudocode and by some sample search trajectories. The algorithm starts by choosing an initial, possibly random, point $\boldsymbol{x}$ in the configuration space; this point is surrounded by an initial *search region* $\mathcal{R}$ (grey boxes A and A') where the next point along the trajectory is searched for. In order to keep a low computation overhead, the search region is identified by d vectors, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d \in \mathbb{R}^d$ which define a "box" around the point $\boldsymbol{x}$:

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

The search occurs by generating points in a stochastic manner with uniform probability in the search region. To speed up operation, a single displacement $\boldsymbol{\Delta}$ from the current point is generated (line 4) and the two points $\boldsymbol{x}^{(t)} + \boldsymbol{\Delta}$ (lines 5–7) and $\boldsymbol{x}^{(t)} - \boldsymbol{\Delta}$ (lines 8–10), specular with respect to $\boldsymbol{x}^{(t)}$, are considered in the region for evaluation (*double shot*). If the value of f at one of the two points is better than the value at $\boldsymbol{x}^{(t)}$, then the better point shall be chosen as the new point in the trajectory, $\boldsymbol{x}^{(t+1)}$ (lines 6, 9); otherwise, $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)}$.

The main feature of the RASH optimization technique, however, is its ability to modify the size and the shape of the search region $\mathcal{R}$ according to the local function landscape, for example when the search trajectory is moving down a narrow "valley", such as in point C' of Figure 1. Therefore, while the search area is obviously enlarged if the search is successful and reduced if

Table 1: Experimental results on 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	—	$1.54 \cdot 10^{-3}$

unsuccessful, it is also stretched along the successful direction (leading, e.g., to the diamond-shaped areas around B and C'), or shortened if neither evaluation leads to an improvement. This is achieved by applying an affine transformation to the vectors defining the area (line 13) whose matrix is determined according to the outcome of the last evaluation (lines 7 and 10 in case of success, line 12 if the current displacement does not lead to an improvement).

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. The scheme described above is devised in order to maintain the search region size as large as possible while still ensuring that the probability of a success per evaluation does not become too small.

3 Experimental results

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].

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

$$f(x) - f_{\min} < \varepsilon_{\text{rel}}|f_{\min}| + \varepsilon_{\text{abs}} \tag{2}$$

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.

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].

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 for comparison Table I of [8]. However, by design RASH has no mechanism to escape local minima after they are identified. Therefore the percentage of success is lower for other functions (e.g., Shekel). Very ill-conditioned problems, such as the Rosenbrock function, are solved in a satisfactory way only for a small number of dimensions. The effects of high dimensionality are

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	$2.53 \cdot 10^{-3}$

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

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 [8] should be considered.

A second test has been performed 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 (2) or the overall maximum number of function evaluations is reached.

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.

3.1 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 [8] and Enhanced Continuous Tabu Search [4], which, like RASH, aim at minimizing functions of continuous variables. Another classical proposal, the Improved Simulated Annealing algorithm [10] 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, 8], 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.

4 Conclusions

In this paper the Reactive Affine Shaker adaptive random search algorithm has been described.

Experiments show a performance which is in some cases comparable or better with respect to 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. We argue that these good results are due to a rapid and effective adaptation of the search region based on feedback from function evaluations at random points. Therefore we believe that this component can be considered 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] Mauro Brunato and Roberto Battiti. Technical Report DIT-06-012, Dipartimento di Informatica e Telecomunicazioni, Università di Trento, March 2006.

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

[8] 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.

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

[10] 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.

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