Journal of Global Optimization **10:** 37–55, 1997.
© 1997 *Kluwer Academic Publishers. Printed in the Netherlands.*

The Hyperbell Algorithm for Global Optimization: A Random Walk Using Cauchy Densities

PIERRE COURRIEU
CREPCO (URA CNRS 182), Université de Provence 29 avenue Robert Schuman, F-13621 Aix-en-Provence Cedex 1, France

(Received: 16 May 1995; accepted: 28 May 1996)

Abstract. This article presents a new algorithm, called the "Hyperbell Algorithm", that searches for the global extrema of numerical functions of numerical variables. The algorithm relies on the principle of a monotone improving random walk whose steps are generated around the current position according to a gradually scaled down Cauchy distribution. The convergence of the algorithm is proven and its rate of convergence is discussed. Its performance is tested on some "hard" test functions and compared to that of other recent algorithms and possible variants. An experimental study of complexity is also provided, and simple tuning procedures for applications are proposed.

Key words: global optimization, random search, Cauchy distributions.

1. Introduction

Optimization problems vary in difficulty, depending on the properties of the function to be optimized. Uniextremal functions are generally the easiest to optimize because "local search" procedures can be used. These procedures, such as usual gradient based methods among others, are efficient and guarantee finding the solution. But many functions encountered in practice have multiple extrema, making it necessary to use "global search" methods, which can generally only guarantee finding the solution with a given degree of probability. Extensive efforts have been made within the past few years to solve hard optimization problems (see Horst & Pardalos, 1995). Certain recent methods are deterministic (Baritompa, 1993; Breiman & Cutler, 1993; Wood, 1992), but they will not be considered here. Most of the known approaches in global optimization are based on stochastic processes (see Zhigljavsky, 1991). Certain algorithms rely on the principle of a random walk converging to an extremum of the objective function (Dekker & Aarts, 1991; Romeijn & Smith, 1994; Solis & Wets, 1981; Zabinsky *et al.*, 1993). In other methods, global distributions of probabilities represented by samples of points ("populations") belonging to the search domain are made to converge (Courrieu, 1993; Goldberg, 1989; Holland, 1975). In still other methods, convergence is achieved by the global distribution control of a set of local searches (Boender, 1982; Rinnooy Kan & Timmer, 1987a, 1987b). Branch-and-Bound methods (Pintér, 1988; Zhigljavsky, 1991) construct a partition of the search domain and sequentially reject those subsets which have the lowest probability of containing a global optimum. Other approaches exist, although many of them appear to be more interesting from a theoretical than practical point of view.

Presented below is a new stochastic global optimization algorithm, which will be called the “Hyperbell Algorithm”. This algorithm relies on the principle of a monotone improving random walk whose steps are generated around the current position according to an n-dimensional Cauchy distribution which is gradually scaled down. The name “Hyperbell” refers to the shape of the probability density function. The algorithm is applicable to the search for the global extrema of a numerical function f defined on a bounded domain Ω of $\mathbb{R}^n$. It is of interest to situate this approach within the theoretical framework proposed by Solis and Wets (1981) because the Hyperbell Algorithm is compatible with their “conceptual algorithm”. The choice of Cauchy distributions is motivated by the search for a good compromise between the speed and the guarantee of convergence. As pointed out by Solis and Wets, guaranteeing convergence requires that the sampling strategy must not indefinitely ignore any subset (which has a positive Lebesgue mesure) of the search domain . Using a uniform distribution on the whole search domain would satisfy this requirement, but unfortunately this leads to very slow convergence. In order to improve the speed of convergence, a common strategy is to gradually concentrate the search on the most “promising” regions. This is the strategy of Branch-and-Bound methods, for example, but the definitive rejection of certain subsets of the search domain does not enable one to guarantee convergence in all cases. Hence, we must concentrate the search on certain subsets while not completely ignoring any subset of the search domain. This can be done by using a bell- shaped probability density of sampling centered on the current position in the search domain, and whose scale can be controlled. Certain common bell-shaped densities, like Gauss or logistic densities, decrease exponentially as the distance from their center increases (in scale units). Hence, the density rapidly tends to zero after a certain distance, resulting in a distribution whose support is “quasi-bounded”, and whose behavior for finite samplings is very similar to that of bounded support distributions. This results in a relatively high probability for the search to be trapped in local extrema basins of many functions, and there is poor guarantee of convergence in practice. Cauchy densities are more appropriate because the density decreases very slowly as the distance from the center of the distribution increases. Hence the probability of sampling is truly non-zero for any subset of the search domain which has a non-zero Lebesgue mesure. The position and the quartile deviations of Cauchy densities are easy to control in order to concentrate the search on appropriate regions. Note that the use of Cauchy distributions was previously shown to be more effective than the use of exponential distributions in the framework of Simulated Annealing algorithms (see Ingber, 1989; Ingber & Rosen, 1992). However, the Hyperbell algorithm is monotonically improving and does not use any concept equivalent to the “temperature” of Simulated Annealing.

2. Hyperbell Algorithm

The algorithm is presented in Pseudo-Pascal. The term “random”, followed by an interval, denotes a random value taken from a uniform distribution of probabilities on the specified interval. The algorithm shown here concerns the search for minima of a function defined on a bounded feasible region Ω of $\mathbb{R}^n$ (remember that the search for maxima of a function f is equivalent to the search for minima of $-f$). In order to initialize the Cauchy distribution scales $(s_1, s_2, \dots, s_n)$, it is useful to determine the upper and lower boundary of each variable such that $a_i \leqslant x_i \leqslant b_i$ and Ω is included in the hyperrectangle $\prod_{i=1}^{n}[a_i, b_i]$.

```
{**Parameters:**
$\alpha \in [0.8, 1)$  : scale reduction parameter;
$\varepsilon > 0$      : arbitrarily small constant;
$DLS$                  : variant selector}

{**Scale initialization (indicative example)**}
**for** $i := 1$ **to** $n$ **do** $s_i := \dfrac{b_i - a_i}{2tg(\pi(0.5)^{1/n}/2)}$;

{**Starting point**}
**repeat**
**for** $i := 1$ **to** $n$ **do** $x_i :=$ random $[a_i, b_i]$;
**until** $(X \in \Omega)$;

{**Search**}
**repeat**
    **repeat**
        **for** $i := 1$ **to** $n$ **do** $y_i := s_i$ tangent($\pi$ random$(-1/2, 1/2)) + x_i$ ;
    **until** $(Y \in \Omega)$;
    **if** $DLS$ **then** $Y := Y - r\nabla f(Y)$; {compute $r$ using a bisection method}
    **if** $f(Y) < f(X)$ **then** $X := Y$ **else for** $i := 1$ **to** $n$ **do** $s_i := \alpha(s_i - \varepsilon) + \varepsilon$;
**until** (stopping rule);
```

The Parameters

The parameter α is the reduction coefficient of Cauchy’s distribution scale, when this scale is reduced. Increasing α lowers the speed of convergence and the probability of being trapped in local minima of complex functions. Note that function complexity should not be assessed solely on the basis of the number of local extrema. We shall see in the experimental section below that there are functions with an infinite number of local minima which are easy to optimize with the Hyperbell Algorithm. The parameter ε has little effect in practice, but it guarantees that the s_i scales will not drop to zero and thereby prevents the process from freezing indefinitely. ε is arbitrarily small and it is generally chosen to be smaller than the desired precision for the result in Ω .

About Scale Initialization

The example of scale initialization given in the algorithm statement corresponds to a probability of 1/2 of generating a new point inside the critical hyperrectangle if the starting point is the center of this hyperrectangle. This choice may seem somewhat arbitrary, however it was empirically found to be usually appropriate. Note that ε must be chosen to be smaller than the initial scales.

Generation of Points

The coordinates of the sampled points are generated independently by means of the generating function: $y_i = s_i tg(\pi u_i) + x_i$, with u_i taken at random from a uniform distribution on $(-1/2, 1/2)$. The random variable defined as such obeys an n-dimensional Cauchy law with the density:

$$g_n(Y; X, S) = \prod_{i=1}^{n} \frac{1}{\pi s_i} \frac{1}{1 + \left(\frac{y_i - x_i}{s_i}\right)^2} .$$

The point X is the center of the distribution (the x_i's are the modes and medians of the marginal distributions), and corresponds to the current position of the random walk in the search domain. The scale parameters s_i are the quartile deviations. Cauchy's law has no moments, and in particular its variance is infinite, which is a reflection of the fact that the density decreases slowly and is never negligible. Given that points generated outside Ω are rejected, the density is in fact a conditional distribution, denoted h, given by:

$$\begin{aligned} h_n(Y; X, S) &= \frac{g_n(Y; X, S)}{\int_\Omega g_n(Z; X, S)\,\mathrm{d}Z} \geqslant g_n(Y; X, S), \quad \text{for } Y \in \Omega, \\ h_n(Y; X, S) &= 0, \quad \text{for } Y \notin \Omega. \end{aligned}$$

DLS Variant

Certain functions to be optimized have special properties, making the use of particular local transformations of the generated points interesting. One of these transformations, which we will refer to as the "Directional Local Search" (DLS) variant, consists simply of one standard local search step: $Y := Y - r\nabla f(Y)$, where the step length r is positive, and is determined using a standard bisection method (see Zhigljavsky, 1991, pp. 21–22) with the constraint that $Y \in \Omega$. The gradient calculation supposes that the function is C^1 continuous and is easily derivable, but an approximation can also be used by replacing the partial derivatives with the corresponding finite increase ratios.

Stopping Rules

Various stopping rules can be used, depending on the requirements for the application. One can simply limit the number of function calls (consumption criterion), or wait until the generation scales are close to ε (precision criterion on Ω).

Convergence

THEOREM 1. *Let f be a function defined on a bounded subset Ω of $\mathbb{R}^n$. Let $X_t \in \Omega$ be the point generated by the Hyperbell Algorithm at time t of the search for a minimum of f on Ω. If f is continuous at a global minimum then: $\forall \delta > 0, \lim_{k \to \infty}$ Prob$\{\exists t \leqslant k; |f(X_t) - \min_{X \in \Omega} f(X)| < \delta\} = 1$. (Replace "min" with "max" for a maximum search).*

Proof.

(1)- $\forall i \leqslant n; s_i(t+1) \geqslant \alpha(s_i(t) - \varepsilon) + \varepsilon \Rightarrow \forall i \leqslant n; \forall t; s_i(t) \geqslant \varepsilon > 0$.

(2)- If Ω is bounded then $\forall X, Y \in \Omega; \|Y - X\| < \infty$.

(3)- (1) and (2) $\Rightarrow \exists \lambda > 0; \forall t; \forall X, Y \in \Omega; h_n(Y; X, S(t)) \geqslant \lambda$.

(4)- Set $\Omega_\delta = \{Y \in \Omega; |f(Y) - \min_{Z \in \Omega} f(Z)| < \delta\}$,
let μ be the Lebesgue measure on the subsets of $\mathbb{R}^n$, if f is continuous at a global minimum then $\forall \delta > 0; \mu(\Omega_\delta) > 0$.

(5)- (3) and (4) $\Rightarrow \int_{\Omega_\delta} h_n(Y; X, S)\, \mathrm{d}Y \geqslant \lambda\mu(\Omega_\delta) > 0$.

(6)- (5) $\Rightarrow \forall \delta > 0$, Prob$\{\exists t \leqslant k; |f(X_t) - \min_{X \in \Omega} f(X)| < \delta\} \geqslant 1 - (1 - \lambda\mu(\Omega_\delta))^k$.

(7)- Finally $\lim_{k \to \infty} 1 - (1 - \lambda\mu(\Omega_\delta))^k = 1$, which completes the proof. □

The situation is very similar using the DLS variant of the algorithm, since this variant is only an additional process which locally favours sampling the best points.

One can also use the global search convergence theorem of Solis and Wets (1981) since the Hyperbell Algorithm is clearly a case of their "conceptual algorithm" and it satisfies their conditions $H1$ and $H2$. The condition $H1$ is trivially satisfied by the acceptance rule of steps. Proving $H2$ is quite similar to the above proof, where one replaces Ω_δ by any subset of Ω with non-zero Lebesgue mesure.

Concerning the rate of convergence with Cauchy distributions, one can use equation 11 from Ingber (1989), or Ingber and Rosen (1992), and conclude that it is a sufficient condition, for obtaining stochastic convergence in any problem, that the scales decrease no faster than $s(0)/t^{1/n}$. However, this limit rate does not take into account properties of the problem other than the dimension (n), and faster convergence can almost always be obtained in practice. This usually requires the use of "free parameters", like α for the Hyperbell algorithm whose stochastic convergence was stated above. Let ε tend to 0, then the ith scale at time t is approximately equal to $s_i(0)\alpha^{t-w(t)}$, where $w(t)$ is the number of improving steps at time t. Clearly, the convergence rate has an exponential form, but it depends on the frequency of winning trials. The scales at a given stage of the process tend to stabilize as long as they are productive of improving steps, and they rapidly decrease when the frequency of winning trials decreases. The choice of α depends on the properties of the problem to be solved. Unfortunately, the author does not know any theory which would enable this choice to be generally optimized. To date, the most usual method consists of empirically determining an α-function appropriate to the class of problems to be solved in a given application. This is not generally difficult, but the cost of the tuning procedure has to be added to the implementation cost.

3. Experimental Study of Performance

Box-Constrained Test Functions

Presented below is an experimental study of the performance of the Hyperbell Algorithm on some hard minimization problems. Most test functions used in the current literature have only a small number of local extrema, which severely limits the scope of the tests (see, however, Floudas & Pardalos, 1990). We were nevertheless able to find two families of standard hard functions for the test: Csendes functions (Csendes, 1985; see also Zhigljavsky, 1991, p. 16) and Griewank functions (Griewank, 1981; see also Rinnooy Kan & Timmer, 1987b, p. 76). A third family of hard functions, previously used by Courrieu (1993), was also selected. This family will be called the W functions. The three families of functions were used with 2 and 10 variables in the comparative study. They were used with 10 variables in the study of densities. Only the W family was used in the complexity study.

Csendes test functions

$$Cn(X) = \sum_{i=1}^{n} x_i^6 \left(2 + \sin\frac{1}{x_i}\right), \quad -1 \leqslant x_i \leqslant 1.$$

Contrary to what might appear, this function is defined at $X = 0$, precisely where it has its unique global minimum (0). The function possesses a countable infinity of local minima on the search domain, and the oscillation frequency tends towards infinity on the neighborhood of the global solution. This property makes it practically impossible to minimize by applying local searches. However, Csendes functions have the following feature: they oscillate between two convex “hulls” which approach each other on the neighborhood of the solution.

Note: what we refer to as a “hull” of a function is a hypersurface which joins all the extrema of the same type (maxima or minima) of the function, and which itself has the minimum number of extrema.

W functions

$$Wn,k(X) = \frac{1}{n}\sum_{i=1}^{n} 1 - \cos(kx_i)\exp(-x_i^2/2), \quad -\pi \leqslant x_i \leqslant \pi.$$

W functions have their unique global minimum (0) at $X = 0$. The number of local minima on the search domain is k^n (for k odd) or $(k+1)^n$ (for k even). Only $k = 10$ was used in the comparative study, which gave 121 local minima for $n = 2$, and more than 2.59×10^{10} local minima for $n = 10$. The function oscillates between two hulls of constant mean $(= 1)$ whose distance from each other is maximal on the neighborhood of the solution.

Griewank test functions

$$Gn(X) = 1 + \sum_{i=1}^{n} x_i^2/d - \prod_{i=1}^{n}\cos(x_i/\sqrt{i}),$$

For $n = 2 : d = 200, -100 \leqslant x_i \leqslant 100$. For $n = 10 : d = 4000, -600 \leqslant x_i \leqslant 600$. These functions have their unique global minimum (0) at $X = 0$, and have a large number of local minima. They have many “trap” basins on a large area around the solution. However, in the neighborhood of certain points, Griewank functions have some special directional local properties. To get an idea of these properties, assign one of the variables a value like $x_i = (2k+1)\pi\sqrt{i}/2, k \in Z$. In this case, the partial derivatives for all other variables point to the solution, which is a stable attractor for the variables that reach its neighborhood.

Comparative Study for Box-Constrained Problems

Reference algorithms

Because the experimental results for problems like these are rare in the literature, three reference algorithms were used.

Simulated Annealing: We chose the Dekker and Aarts (1991) algorithm, which is a recent version of the Simulated Annealing algorithm adapted to numerical spaces. It is like a sequential Multistart algorithm in that it uses multiple directional local search steps, whose global distribution is governed by a simulated annealing rule. The algorithm was implemented in compliance with the indications provided by the authors, except that the stopping rule was simplified in an experimentally adapted way. This rule was replaced by a “temperature” thresholding, involving a complete local search starting from the best point found whenever the temperature fell below

10^{-8}. For the test functions used, this order of magnitude for the temperature was indeed found to correspond to a “freezing” point in the process, after which no progress was observed. We used the tuning values proposed by the authors (namely: $\chi_0 = 0.9, L_0 = 10, t = 0.75$), with $m_0 = 100$, except for the speed parameter that usually had to be reduced since a δ of 0.1 proved to be unsuitable to problems this difficult. To determine δ, a trial and error procedure was used to obtain 10 successive runs with no error in the result for each problem. This search succeeded for three out of six problems (W2, G2, G10), but it was impossible to obtain an exact result for the remaining problems, even once. The value used for δ, then, was simply the minimum value found in those problems which were solved (0.005).

Improving Hit-and-Run: we chose the algorithm proposed by Zabinsky *et al.* (1993) (see also Romeijn & Smith, 1994). It is a converging random walk algorithm which generates an asymptotically uniform distribution on the feasible region, and it was implemented as it is described by the authors (with $H = I$). No tuning is necessary for the Improving Hit-and-Run algorithm, and the stopping rule was fixed at 500000 evaluations of the function.

Distributed Search: this algorithm proposed by Courrieu (1993) was chosen because it uses Cauchy distributions like the Hyperbell Algorithm, but it is not a random walk. The Distributed Search uses a population of M points in the feasible region, each of them being the center of a Cauchy distribution. The convergence is obtained by estimating the most appropriate positions and scales of the M distributions at successive stages of the search. The speed of convergence is controlled by a parameter (α), and the algorithm has a DLS variant which is quite similar to that of the Hyperbell Algorithm. To determine M, α and the eventual necessity of applying the DLS variant, a trial and error procedure was used to obtain 10 successive runs with no error in the result for each problem.

Tuning the Hyperbell Algorithm

The Hyperbell Algorithm parameter α was also estimated by trial and error until we obtained 10 successive runs with no error in the result of any of the problems. This criterion obviously did not imply that the probability of error was equal to zero, which *a priori* is impossible in finite time. The results were considered exact (for all algorithms) when the process found a function minimum equal to 0, within the precision range of the compiler (TURBO-PASCAL 5.0 on a COMPAQ Deskpro 486 computer). The parameter ε was set at 10^{-20}. The DLS variant was applied only if the tuning procedure failed without it (this occured only for the function G10).

Results

The results are presented in Table I. For each algorithm and each problem, the table gives the values of the tuning parameters and the mean number of function calls

Table I. Tuning, and mean performance (with standard deviation) on 10 successive runs for the four algorithms and six test functions.

			TESTED	ALGORITHMS	
		Dekker & Aarts (1991)	Zabinsky *et al.* (1993)	Courrieu (1993)	Hyperbell Algorithm
C2	tuning	δ = 0.005	–	M=100,α =1.00	α = 0.93
	f calls	46549 (13149)	287292 (100493)	7028 (949)	663 (21)
	f error	3.38E–15 (1.01E–14)	5.49E–30 (1.11E–29)	No error	No error
	X error	2.42E–03 (1.65E–03)	8.14E–06 (5.60E–06)		
C10	tuning	δ = 0.005	–	M=200,α =1.00	α =0.993
	f calls	430301 (41406)	465682 (31070)	89453 (2304)	6621 (88)
	f error	1.30E–10 (1.18E–10)	2.03E–23 (1.79E–23)	No error	No error
	X error	3.76E–02 (4.76E–03)	2.47E–04 (5.21E–05)		
W2	tuning	δ = 0.005	–	M=100,α =0.75	α =0.99
	f calls	29485 (11534)	363364 (108840)	4161 (371)	2313 (41)
	f error	No error	2.71E–08 (3.09E–08)	No error	No error
	X error		2.75E–05 (1.88E–05)		
W10	tuning	δ = 0.005	–	M=250,α =0.75	α =0.99978
	f calls	221752 (29660)	496511 (4753)	119799 (2617)	105723 (899)
	f error	0.210 (0.028)	0.372 (0.058)	No error	No error
	X error	2.927 (0.494)	4.187 (0.936)		
G2	tuning	δ = 0.005	–	M=150,α =0.80	α =0.995
	f calls	52793 (4741)	402978 (98100)	5712 (393)	4687 (93)
	f error	No error	6.38E–07 (6.92E–07)	No error with DLS	No error
	X error		1.21E–03 (6.27E–04)		
G10	tuning	δ = 0.01	–	M=300,α =0.60	α =0.995, DLS
	f calls	251870 (10208)	471011 (31388)	205584 (4084)	106799 (10583)
	f error	No error	0.161 (0.106)	No error with DLS	No error
	X error		23.98 (8.12)		

for the 10 runs (f calls). Each gradient calculation was counted as a calculation of the function. This is particularly justified in the case of separable functions where the calculation cost of the gradient is very close to the calculation cost of the function. Also given are the mean error on the function value (f error) and the mean Euclidean distance between the best point found and the global minimizer (X error). Standard deviations (with 9 degrees of freedom) are shown in parentheses to the right of the corresponding means.

Table II. Performance of the random walk using Gauss and logistic densities for three test functions (ten successive runs).

f		Gauss density	Logistic density
C10	tuning	α = 0.989	α = 0.99
	f calls	4305 (66)	4705 (58)
		No error	No error
W10	tuning	α = 0.9999	α = 0.9999
	f calls	250781 (10306)	247011 (11470)
	f error	0.293 (0.082)	0.268 (0.063)
	X error	3.492 (0.951)	3.397 (0.773)
G10	tuning	α = 0.99, DLS	α = 0.99, DLS
	f calls	59094 (1667)	58922 (2187)
		No error	No error

The results are quite clear. The Hyperbell Algorithm solved all the problems with the smallest number of function evaluations. The most difficult function for the algorithm was G10 whose optimization clearly required the DLS variant. Without the DLS variant, the process frequently became trapped by local minima. Note that the minimization of Griewank functions is relatively easy for the Dekker and Aarts (1991) algorithm; this is logical given its strong directional local component. However, note also that Rinnooy Kan and Timmer (1987b) reported some disappointing results concerning the minimization of the same Griewank functions by a “Multi Level Single Linkage” algorithm. The Zabinsky *et al.* (1993) algorithm found good approximations of the minimum for functions C2, C10, W2 and G2, but the convergence was slow. The Distributed Search method found the exact solution for all the problems, but as one can see in Table I, that algorithm converges much slower than the Hyperbell Algorithm.

Study of Densities: Using Other Bells

The functions C10, W10 (with $k = 10$) and G10 were used for testing the performance of the random walk when one replaces the Cauchy density with gradually scaled down Gauss and logistic densities. Approximations of Gaussian variables were generated using a sum of 12 independent uniform random variables. Each of the 10 standard deviations was initialized at $s_i(0) = (b_i - a_i)/3.664$, given that Prob$\{|z| < 3.664/2\} = 0.5^{1/10}$. Logistic random variables were generated using the formula $y_i = -s_i\ \mathrm{Ln}(1/u_i - 1) + x_i$, with u_i taken at random from a uniform distribution on (0,1), and $s_i(0) = (b_i - a_i)/2(-\mathrm{Ln}(2/(1 + 0.5^{1/n}) - 1)), n = 10$. The tuning procedure was similar to that employed for the Hyperbell Algorithm with Cauchy densities. Means and standard deviations of performance on ten suc-

Table III. Tuning of the Hyperbell Algorithm and mean number of function calls (with standard deviation) as a function of the number of variables (ten successive runs without error).

Number of variables	Tuning (α)	Function calls		Number of variables	Tuning (α)	Function calls	
3	0.9500	504	(18)	18	0.9949	5045	(205)
6	0.9810	1339	(66)	21	0.9958	6002	(146)
9	0.9870	1982	(73)	24	0.9965	7247	(262)
12	0.9910	2875	(113)	27	0.9971	8786	(350)
15	0.9938	4127	(133)	30	0.9973	9529	(376)

cessive runs are presented in Table II. As one can see, Gauss and logistic densities provided good performance on function C10, and also on function G10 with the DLS variant. However, it was impossible to solve the W10 problem, even once, using Gauss or logistic densities. Clearly, the use of exponential densities does not lead to algorithms as robust as the Hyperbell Algorithm with Cauchy densities. The behaviour of exponential densities looks like the behaviour of bounded support densities, they provide fast but uncertain convergence. In contrast, approximations of uniform densities (e.g. Improving Hit-and-Run) provide slow, truly guaranteed convergence. The use of Cauchy densities allows for a good compromise between speed and accuracy requirements.

Complexity Study

The behaviour of the Hyperbell Algorithm was studied varying two usual factors of complexity: the number of variables and the number of minima. The W test function family was used since these complexity factors are easy to control for these functions.

Effect of the number of variables: For this study, the W test function was used with $k = 0$ and a number of variables varying from 3 to 30 (step 3). This is a set of ten uniminimal functions. The tuning procedure of the Hyperbell Algorithm was similar to that employed in the preceding studies. Results are presented in Table III. The number of function calls (t) increased as a function of the number of variables (n). This function was close to linear. The best regression equation is $t = 343.4n^{1.3} - 922$, with a very high correlation coefficient $r = 0.997$. A simple linear regression gave $r = 0.994$. The tuning was very well predicted by the relation $\alpha = 0.2064(1 - n^{-1.3}) + 0.7934, r = 0.999$.

Effect of the number of minima: For this study, the W test function with $n = 1$ and 13 even values for k was used, giving a range of 1 to 101 minima. Results are presented in Table IV. As one can see, the number of function calls increased monotonically as a function of the number of minima (m). The best simple relation which was found is $t = 47.83m^{0.87} + 156.97, r = 0.968$. However, this type of

Table IV. Tuning of the Hyperbell Algorithm and mean number of function calls (with standard deviation) as a function of the number of minima (ten successive runs without error).

Number of minima	Tuning (α)	Function calls		Number of minima	Tuning (α)	Function calls	
1	0.820	134	(15)	15	0.9720	764	(64)
3	0.830	149	(17)	17	0.9725	823	(28)
5	0.860	178	(10)	19	0.9730	826	(16)
7	0.940	372	(30)	21	0.9736	867	(43)
9	0.965	629	(36)	51	0.9800	1173	(40)
11	0.969	672	(73)	101	0.9923	2956	(126)
13	0.971	758	(47)				

relation cannot be generalized without caution since we have seen that it is easy for the Hyperbell Algorithm to minimize Csendes functions, which have theoretically an infinity of minima. In fact, we have to suspect that the number of minima *per se* is not relevent, but that it is linked to more relevent caracteristics in the case of W functions.

Comparative Study for Nonlinearly Constrained Problems

The Hyperbell algorithm was initialy designed for searching global extrema of multiextremal functions defined on simple box-constrained feasible regions. However, certain recent global optimization algorithms are designed for solving optimization problems on feasible regions with relatively complex constraints (Zabinsky *et al.*, 1993; Romeijn & Smith, 1994). Hence, it appeared interesting to complete this experimental study with standard nonlinearly constrained problems. The two most difficult nonlinearly constrained problems used by Romeijn & Smith (1994) were selected (problem 1 and 2, pp. 119–120). These authors reported the best performance which was obtained with Hide-and-Seek type algorithms (including Improving Hit-and-Run) for solving these problems with a precision of 1%. For comparison, the same precision criterion (stopping rule) was used in the present study. Table V presents the mean number of constraint evaluations necessary for finding an initial point inside the feasible region (i.c.e), the mean number of function evaluations (f.e.), and the mean number of constraint evaluations (c.e.). These means were obtained with 20 successive runs, and the corresponding standard deviations are given in parentheses for the Hyperbell algorithm. Results concerning the Hide- and-Seek algorithm are taken from Romeijn & Smith (1994). As one can see, the Hide-and-Seek algorithm found an initial feasible point faster than the Hyperbell algorithm for problem 2. Hide-and-Seek algorithm has a special procedure for finding an initial feasible point. When the constraint function defines a subset with relatively low Lebesgue mesure, this procedure is more effective than

Table V. Mean performance (with standard deviations) for 20 successive runs on two nonlinearly constrained problems solved with 1% precision.

	Hyperbell	Romeijn & Smith (1994)
Problem 1	α = 0.992	
i.c.e	6.55 (6.39)	10.1
f.e.	329.75 (131.70)	530.6
c.e.	744.90 (276.10)	1876.3
Problem 2	α = 0.990	
i.c.e.	835.85 (770.82)	158.1
f.e.	168.95 (51.30)	281.9
c.e.	8323.45 (2730.87)	13893.1

the very simple initialization procedure of the Hyperbell algorithm. However, this was the only advantage of Hide-and-Seek in the present experiment, and globally, the Hyperbell algorithm always solved the problems faster.

4. Tuning Procedures for Applications

Tuning the algorithm for standard test problems is quite easy because the exact solution of these problems is known *a priori*. However, this is not the case, in general, for realistic applications. Moreover, certain applications require strict control of computational effort, and more or less precision (or reliability) for the result. Usually, the tuning is performed at the time of implementation ("off-line tuning"). In this case, one must determine an α value or an α-function (of complexity variables) that properly generalizes to the (infinite) set of problems which the application should be able to solve. However, in certain cases, it is not possible to find an appropriate α-function for the entire set of potential problems. Such a situation requires the use of an "on-line tuning" procedure. To date, we do not know of any general tuning method with low computational cost. In this section, we first examine the behavior of the algorithm as a function of its tuning. Based on the resulting observations, examples of simple tuning procedures are defined and experimental results are presented.

Test Problem

For this study, we selected a family of problems which has practical applications and whose exact solution is not known to date. This is the so called "elliptic Fekete points' problem (of order d)" (Pardalos, 1995; Shub & Smale, 1993):

$$\begin{array}{ll} \text{global max} & f_d(x) = \prod_{1\leqslant i\leqslant j\leqslant d} \|x_i - x_j\|, \quad x \in \mathbb{R}^{3d}, \\ \text{subject to} & \|x_i\| = 1, i = 1, ..., d. \end{array}$$

Table VI. Behavior of the Hyperbell Algorithm (means and standard deviations for 10 runs) as a function of α for a 12 Fekete points' problem.

α	Maximum of f_{12} found	Solving time
0.8	1.78405E+8 (3.46344E+8)	392 (15)
0.9	4.62357E+8 (4.25513E+8)	810 (34)
0.95	1.55778E+9 (2.98085E+8)	1607 (44)
0.975	2.05888E+9 (1.12839E+8)	3166 (61)
0.9875	2.19609E+9 (1.00874E+8)	6192 (132)
0.99375	2.35477E+9 (6.96366E+7)	12275 (225)
0.996875	2.40710E+9 (2.66601E+7)	22209 (2480)
0.9984375	2.41785E+9 (0)	25802 (5062)
0.99921875	2.41785E+9 (0)	54298 (6732)
0.999609375	2.41785E+9 (0)	91316 (21832)
α	Frequency of winning trials	Time to stop
0.8	0.491 (0.020)	395 (15)
0.9	0.477 (0.021)	816 (32)
0.95	0.459 (0.014)	1619 (42)
0.975	0.446 (0.009)	3198 (50)
0.9875	0.430 (0.010)	6262 (108)
0.99375	0.425 (0.008)	12434 (179)
0.996875	0.370 (0.055)	22880 (1902)
0.9984375	0.031 (0.006)	29599 (192)
0.99921875	0.011 (0.001)	58010 (85)
0.999609375	0.004 (0.001)	115301 (81)

The constraint defines a set of zero mesure in $\mathbb{R}^3$ (the unit sphere surface). Hence we have to use a projection of $\mathbb{R}^3$ points on S^2:

$$x_i := z_i/\|z_i\|, z_i \in [-1,1]^3\backslash 0.$$

In doing this, all points z_i generated along a given radial direction are equivalent, the number of independent variables theoretically reduces to $2d$, and one can easily verify that the convergence properties of the Hyperbell Algorithm are not modified.

Effects of Tuning

In order to illustrate a typical behavior of the Hyperbell Algorithm as a function of α , a 12 Fekete points' problem was approximately solved with 10 different values of α according to: $\alpha_{k+1} = (\alpha_k + 1)/2, \alpha_0 = 0.8, k = 0, ..., 9$. The stopping criterion was $s(t) \leqslant 1.10\varepsilon, \varepsilon = 10^{-20}$, and 10 runs were performed for each α value. Table VI reports the means (and standard deviations) of the maximum found for the objective function, the solving time (number of function calls for finding the result), the frequency of winning trials at stopping criterion, and the time to stop (number of function calls for reaching the stopping criterion).

One can see in Table VI that the approximation of the function global maximum is improved as α increases, and that one obtains a stable result (with variance close to 0) after a critical value of α is reached. The computational cost monotonically increases, and the frequency of winning trials decreases as a function of α . Note also that the mean frequency of winning trials is lower than 1/2 for usual values of $\alpha(\geqslant 0.8)$. A stable result can be considered as the best solution which can be provided by the algorithm in the limit of a given computational effort. However, if no validity test of the result is available, one can only guess that this is a global optimum.

Cost Constrained Tuning

This is an on-line tuning method which enables one to choose the range of the computational cost allowed for the search. Choose a large integer T, a small real $\omega > \varepsilon$, and a stopping criterion of the form $s(t_\omega) \leqslant \omega$, with the (approximative) constraint that $t_\omega \leqslant T$. If the scales of the different variables are not equal, consider only the largest one, or eventually choose an ω value for each variable in such a way that $\omega_i = c(s_i(0) - \varepsilon) + \varepsilon, i = 1...n$, where c is a small positive constant. Clearly T is an approximation of the maximum number of function evaluations allowed for the search, with a precision criterion ω in Ω . After the algorithm statement, we have:

$$\begin{aligned} &s(t_\omega) = (s(0) - \varepsilon)\alpha^{(1-w_\omega)t_\omega} + \varepsilon = \omega \Rightarrow \\ &t_\omega = \frac{1}{(1 - w_\omega)\ln\alpha} \ln\left(\frac{\omega - \varepsilon}{s(0) - \varepsilon}\right), \end{aligned}$$

where w_ω is the frequency of winning trials at stopping criterion time. w_ω is a random variable whose law depends on the problem and on the tuning, however one can estimate that $0 < w_\omega \leqslant 1/2$ in most cases. Assuming this, we obtain:

$$\ln\left(\frac{\omega - \varepsilon}{s(0) - \varepsilon}\right) / \ln\alpha < t_\omega \leqslant 2ln\left(\frac{\omega - \varepsilon}{s(0) - \varepsilon}\right) / \ln\alpha \simeq T,$$

then we take:

$$\alpha = \exp\left(2\ln\left(\frac{\omega - \varepsilon}{s(0) - \varepsilon}\right) / T\right) = \left(\frac{\omega - \varepsilon}{s(0) - \varepsilon}\right)^{2/T}.$$

If T is sufficiently large, one can obtain a stable result (or a global extremum), but this cannot be verified using only one run. If no validity test of the result is available, one can use two different costs (say T1 and T2) and perform the two corresponding runs. Then the global cost will be comprised between (T1+T2)/2 and T1+T2. In general, there is little chance of obtaining two similar results if these are not stable solutions.

Off-Line Tuning

Performing an off-line tuning, one must determine an α value or an α-function (of complexity variables) which will provide reliable results with minimal computational cost for a large class of problems. Given a particular problem, we first must find a minimal α value which provides stable results. Given the convergence properties illustrated in Table VI, this reduces to a one variable uniextremal optimization. Hence, variants of usual local search methods, such as bisection type procedures, can be used, and the convergence towards an optimal tuning can be guaranteed. Now, for the purpose of an application, it is not necessarily relevent to find an exact optimal tuning for a particular problem, and one has to take into account the computational cost of the tuning procedure itself. In particular, we have to take into account the fact that the computational cost increases quickly as α tends to 1, since this cost is approximately proportional to $|1/\ln\alpha|$ (see previous section). Hence, we must define a tuning procedure which avoids large overestimating of α and repeated runs using overestimated α values. As an example, the following procedure provides quite good results:

```
{parameters: $\alpha_0, \alpha_1$: two initial low values of $\alpha$;
N: critical number of repeated runs; }
{first approximation} $f_0$ := Hyperbell$(f, \alpha_0)$;
BEST := $f_0$; $f_1$ := Hyperbell$(f, \alpha_1)$;
if ($f_1$ better than BEST) then BEST := $f_1$;
$k := 1$;
while ($f_{k-1} \neq$ BEST) or ($f_k \neq$ BEST) do
    begin
    $k := k + 1$;
    $\beta_k := \left( \frac{|f_{k-1} - f_{k-2}|}{|f_{k-1}| + |f_{k-2}|} \right)^{0.1}$;
    $\alpha_k := (\alpha_{k-1} + \beta_k)/(1 + \beta_k)$;
    $f_k$ := Hyperbell$(f, \alpha_k)$;
    if ($f_k$ better than BEST) then BEST := $f_k$;
    end;

{final tuning}
$k := k - 1$;
$r := 1$;
repeat
    repeat
        $f_k$ := Hyperbell$(f, \alpha_k)$;
        $r := r + 1$;
    until $(r = N)$ or ($f_k \neq$ BEST);
```

```
if ($f_k \neq$ BEST) then begin
        $r := 0$;
        $k := k + 1$;
        $\beta_k := \left( \frac{|f_{k-1} - \text{BEST}|}{|f_{k-1}| + |\text{BEST}|} \right)^{0.1}$;
        $\alpha_k := (\alpha_{k-1} + \beta_k)/(1 + \beta_k)$;
        if ($f_{k-1}$ better than BEST) then BEST $:= f_{k-1}$;
        end;
until ($r = N$);
```

Note that the differences ($\neq$) have to be tested with finite precision (e.g. 15 significant decimal digits). The expression of β_k can be modified. For example, using $\beta_k = 1$ provides an α sequence similar to the one reported in Table VI (with α_0 = 0.8 and α_1 = 0.9). The choice of N (critical number of repeated runs) depends on the desired reliability of the tuning. The "first approximation" part of the procedure can also be used as an on-line tuning procedure. This can be interesting because the procedure ends when a result stability criterion is satisfied, however the computational cost is not controlled for.

The tuning procedure was applied to elliptic Fekete points' problems of various orders. Main results concerning orders 10, 11 and 12 are reported in Table VII. Given the α-function (of the number of variables) previously obtained for uniextremal functions (see the "complexity study" section), it was assumed that $\alpha \geqslant \alpha_1 = 0.8 + 0.2(1 - 1/d)$. Tuning costs are total numbers of function evaluations. The first approximation tuning cost can be viewed as an on-line tuning cost with stability criterion. The estimated maximum of f was always obtained in the first approximation procedure with 15 significant digits. The final tuning procedure only minimized the solving time for the specified reliability criterion ($N = 10$).

The remaining problem concerns the generalization of tuning to a large class of problems. Depending on the application, the problems can differ only by a random set of data, or (also) by systematic complexity factors (e.g. number of variables). In the first case, a quite evident method consists of selecting a sample of problems and performing the tuning for each problem, which provides a sample of tunings. Then one can determine an upper boundary of α using a simple confidence interval method. However, the distribution of α, considered as a random variable, is very asymetrical. Hence, it is better to compute the confidence interval for the t_ω variable (by usual statistical methods), and then to compute the corresponding upper bound of α using the relations stated in the "cost constrained tuning" section. The second case is the most frequent in practice. Unfortunately, it is also the most problematic case because available empirical complexity factors are not always the relevent ones, or the α-function is not a simple monotonic function of these factors. Sometimes, it is easy to find an appropriate approximation of the α-function, as we did in the "complexity study" section. However, if one considers for example

Table VII. Results provided by the tuning procedure for Fekete points' problems of order 10, 11, and 12. $\alpha_0 = 0.8, \alpha_1 = 0.8 + 0.2(1 - 1/d), N = 10$.

	f_{10}	f_{11}	f_{12}
α	0.999609313189144	0.999654352379111	0.998184163562489
maximum of f_d	5.74088185070187E+6	9.99798997082430E+7	2.41785163922926E+9
mean solving time	90807	122889	21645
first approx. cost	712488	1051620	121779
total tuning cost	1798733	2591413	352330

the set of elliptic Fekete points' problems, the α-function of d seems hard to predict. In such a situation, a practical solution would probably be to implement the tuning values for the most frequent problems, and to implement an on-line tuning procedure for the remaining cases.

5. Conclusion

The Hyperbell Algorithm clearly exhibits high performance levels on difficult global optimization problems. The use of Cauchy distributions in this type of framework is visibly more appropriate than the use of more usual distributions. Complexity effects which were experimentally tested are close to linear, however not all the relevent complexity factors are identified. Simple tuning procedures are available, however verifying reliability always requires a non negligible amount of computational effort. Further research efforts are needed to develop an efficient way of determining the best algorithm tuning for each problem or application. This requires a theory of problem complexity which is not available to date. Despite these remaining questions, the Hyperbell algorithm is a useful tool in practice. It is very easy to implement and, clearly, it provides reliable results with fast convergence.

References

Baritompa, W. (1993), Customizing methods for global optimization – a geometric viewpoint. *Journal of Global Optimization* **3**, 193–212.

Boender, C.G.E., Rinnooy Kan, A.H.G., Stougie, L., Timmer, G.T. (1982), A stochastic method for global optimization, *Mathematical Programming* **22**, 125–140.

Breiman, L., Cutler, A. (1993), A deterministic algorithm for global optimization, *Mathematical Programming* **58**, 179–199.

Courrieu, P. (1993), A distributed search algorithm for global optimization on numerical spaces. *RAIRO: Recherche Opérationnelle / Operations Research,* **27(3)**, 281–292.

Csendes, T. (1985), A simple but hard-to-solve global optimization test problem, *IIASA Workshop on Global Optimization,* Sopron (Hungary).

Dekker, A., Aarts, E. (1991), Global optimization and simulated annealing, *Mathematical Programming* **50**, 367–393.

Floudas, C.A., Pardalos, P.M. (1990), *Collection of Test Problems for Constrained Global Optimization Algorithms,* Springer-Verlag, Lecture Notes in Computer Science 455.

Goldberg, D.E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts.

Griewank, A.O. (1981), Generalized descent for global optimization, *Journal of Optimization Techniques and Application* **34**, 11– 39.

Holland, J.H. (1975), *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor.

Horst, R., Pardalos, P.M. (1995), *Handbook of Global Optimization,* Kluwer Academic Publishers, Dordrecht.

Ingber, L. (1989), Very fast simulated re-annealing, *Mathl. Comput. Modeling* **12(8)**, 967–973.

Ingber, L., Rosen, B. (1992), Genetic algorithms and very fast simulated reannealing: a comparison. *Mathl. Comput. Modelling* **16(11)**, 87–100.

Pardalos, P.M. (1995), An open global optimization problem on the unit sphere, *Journal of Global Optimization* **6**, 213.

Pintér, J. (1988), Branch-and-Bound methods for solving global optimization problems with Lipschitzian structure, *Optimization* **19(1)**, 101–110.

Rinnooy Kan, A.H.G., Timmer, G.T. (1987a), Stochastic global optimization methods. Part I: clustering methods, *Mathematical Programming* **39**, 27–56.

Rinnooy Kan, A.H.G., Timmer, G.T. (1987b), Stochastic global optimization methods. Part II: multi level methods, *Mathematical Programming* **39**, 57–78.

Romeijn, H.E., Smith, R.L. (1994), Simulated Annealing for constrained global optimization, *Journal of Global Optimization* **5**, 101–126.

Shub, M., Smale, S. (1993), Complexity of Bezout's theorem III. Condition number and packing, *Journal of Complexity* **9**, 4– 14.

Solis, F.J., Wets, R.J-B. (1981), Minimization by random search techniques, *Mathematics of Operations Research* **6(1)**, 19– 30.

Wood, G.R. (1992), The bisection method in higher dimensions, *Mathematical Programming* **55**, 319–337.

Zabinsky, Z.B., Smith, R.L., McDonald, J.F., Romeijn, H.E., Kaufman, D.E. (1993), Improving Hit-and-Run for global optimization. *Journal of Global Optimization* **3**, 171–192.

Zhigljavsky, A.A. (1991), *Theory of Global Random Search,* Kluwer Academic Publishers, Dordrecht.