The Discrete Gradient Evolutionary Strategy Method for Global Optimization

[†]H.A. Abbass, [‡]A.M. Bagirov and [‡]J. Zhang
†Artificial Life and Adaptive Robotics Lab, School of Information Technology and Electrical Engineering, Australian Defence Force Academy, University of New South Wales, Canberra, ACT 2600, Australia
‡Centre for Informatics and Applied Optimization, School of Information Technology and Mathematical Sciences, University of Ballarat, Victoria 3353, Australia

Abstract- Global optimization problems continue to be a challenge in computational mathematics. The field is progressing in two streams: deterministic and heuristic approaches. In this paper, we present a hybrid method that uses the discrete gradient method, which is a derivative free local search method, and evolutionary strategies. We show that the hybridization of the two methods is better than each of them in isolation.

1 Introduction

Optimization theory provides a compact set of techniques to handle different types of optimization problems. The general optimization problem can be stated as:

$$\text{(P1): Minimize } f(x) \tag{1}$$

$$\text{subject to: } x \in \mathbb{R}^n, \quad G(x) \leq 0 \tag{2}$$

where x is the set of decision variables, $f(x)$ is the objective function and $G(x)$ is a set of constraints. Let

$$\theta = \{x \in \mathbb{R}^n : \quad G(x) \leq 0\} \tag{3}$$

be a compact set representing the set of feasible solutions. Two important types of optimal solutions will be referred to in the rest of this paper, local and global optimal solutions. Let us define the open ball (*ie.* a neighborhood centered on $\bar{x}$ and defined by the Euclidean distance $\delta > 0$)

$$B_\delta(\bar{x}) = \{x \in \mathbb{R}^n : \ ||x - \bar{x}|| < \delta\} \tag{4}$$

Definition 1 A point $\bar{x} \in \theta$ is said to be a local minimum of the optimization problem iff $\exists\, \delta > 0$ such that $f(\bar{x}) \leq f(x), \forall x \in (B_\delta(\bar{x}) \cap \theta)$.

Definition 2 A point $\bar{x} \in \theta$ is said to be a global minimum of the optimization problem iff $f(\bar{x}) \leq f(x), \forall x \in \theta$.

In the general case, it is known that global optimization problems are NP–hard [3]; that is, there currently exists no polynomial time algorithm that can solve any global optimization problem on a Turing machine. Therefore, it is essential to identify the pros and cons of each method, in a hope to be able to establish a framework of when to use or not to use each method.

During the course of this paper, we will use some mathematical terminologies which we will clarify here.

Definition 3 A subset M of a linear space L is *convex* iff $x \in M$ and $y \in M$ implies that $\alpha \times x + (1 - \alpha) \times y \in M$ for any $\alpha \in [0, 1]$.

Definition 4 A function f is called *convex* iff its domain $\mathcal{D}$ is a convex set and $f(\alpha \times x + (1 - \alpha) \times y) \leq \alpha \times f(x) + (1 - \alpha) \times f(y)$ for any $x, y \in \mathcal{D}$ and $\alpha \in [0, 1]$.

Definition 5 Let f be a convex function with domain $\mathcal{D}$. Let x_0 be an interior point of $\mathcal{D}$. Define $g(x_0)$ such that $f(x) - f(x_0) \geq \langle g(x_0), x - x_0\rangle \ \ \forall x \in \mathcal{D}$. $g(x_0)$ is called a sub–gradient or a generalized gradient of f at x_0.

In the previous definition, $\langle g(x_0), x - x_0\rangle$ is a supporting hyperplane at x_0.

Theorem 1 The set of generalized gradients of a convex function f at any interior point x_0 of the domain $\mathcal{D}$ is nonempty, bounded, closed, and convex.

Definition 6 A function f is locally Lipschitz continuous on $\mathbb{R}^n$ if in any open bounded subset $S \in \mathbb{R}^n$ $\exists$ a constant $L > 0$ such that $\frac{f(x) - f(y)}{||x - y||} \leq L, \ \forall x, y \in S$.

The locally Lipschitz function f is differentiable almost everywhere and one can define for it a set of generalized gradients or a Clarke subdifferential (see [10]), by

$$\partial f(x) = \text{co}\,\{v \in \mathbb{R}^n : \exists (x^k \in D(f), x^k \to x, k \to +\infty) : v = \lim_{k \to +\infty} \nabla f(x^k)\},$$

here $D(f)$ denotes the set where f is differentiable, co denotes the convex hull of a set and $\nabla f(x)$ stands for a gradient of the function f at a point $x \in \mathbb{R}^n$.

Techniques for solving global optimization problems can be divided without any loss of generality into the following categories [13, 28]:

Covering methods: These methods guarantee certain level of accuracy. They use a global search strategy, such as quasi-Monte Carlo methods, to deterministically generate a sequence of points that uniformly search the space. The accuracy of these methods would usually depends on a measure of the uniformity of the sequence. Examples of this class of methods include [6, 9, 22, 31].

Statistical models of objective functions: These methods generate some local information to approximate the function landscape. The approximation can be done through statistical information methods, interpolation, Bayesian methods, or other multidimensional axiomatically based methods. This is a costly process but it can be useful for problems where the evaluation of the objective function is a computationally expensive task. Examples of this class of methods include [17, 20].

Generalized descent methods: These methods use similar search mechanisms as in local search methods while attempting to search globally. Two streams of methods exist under this category: trajectory and penalty methods. Trajectory methods correct the trajectories taken by the local search technique so that all local optima are discovered. For example, it may force an ascend trajectory after a local optima is found. Penalty methods penalize or forbid the search from encountering the same local optima again. For example, in the tunneling algorithm of Gomez and Levy [12], once a local minimum x with objective value $f(x)$ is encountered, the algorithm looks for a minima y, where $f(y) < f(x)$. Examples of this class of methods include [8, 11, 12].

Clustering methods: Clustering methods start the search by performing Monte Carlo sampling of the search space. The sample needs to be somehow proportional to the expected number of local optima in the problem; the more local optima the larger the sample needs to be. Clustering is then carried out to locate the local minima followed by the application of a local search technique within each cluster. A major drawback of these algorithms is their poor performance when the number of local optima increases. Examples of this class of methods include [7, 23, 27].

Random search methods: These techniques, sometimes called stochastic search methods, make random decisions during their search. We can distinguish between two classes of random search methods: adaptive and non–adaptive. Adaptive random search allows for the parameters or the initial distribution for generating solutions to change during the run, while in non–adaptive random search, this does not occur. Evolutionary computations and simulated annealing falls into this class of methods.

One approach that recently has drawn attention is to combine global and local search methods to design more efficient global optimization algorithms (see [5, 14, 15, 32].) In these hybrid methods some meta-heuristic methods like simulated annealing, tabu search and etc. can be used as a global search method. In this paper we develop a new hybrid discrete gradient evolutionary strategy method. This hybrid method uses the discrete gradient method, which is a derivative free local search method, and evolutionary strategies. We present the results of numerical experiments which demonstrate that the hybridization of these two methods is better than each of them in isolation.

The structure of the paper is as follows. Section 2 presents the methods to be used in this paper. Test problems are presented in Section 3 followed by the numerical experiments in Section 4. Conclusions are drawn in Section 5.

2 Methods

2.1 Discrete gradient method

In this section we will give a brief description of the discrete gradient method. The full description of this method can be found in [4]. The discrete gradient method can be considered as a version of the bundle method when subgradients are replaced by their approximations - discrete gradients (for the bundle method see, for example, [16]).

Let f be a locally Lipschitz continuous function defined on $\mathbb{R}^n$. Let

$$S_1 = \{g \in \mathbb{R}^n : \|g\| = 1\},$$
$$G = \{e \in \mathbb{R}^n : e = (e_1, \dots e_n), |e_j| = 1, \ j = 1, \dots, n\},$$
$$P = \{z(\lambda) : z(\lambda) \in \mathbb{R}^1, \ z(\lambda) > 0, \ \lambda > 0, \ \lambda^{-1}z(\lambda) \to 0,$$
$$\lambda \to 0\}, \quad I(g, \alpha) = \{i \in \{1, \dots, n\} : \ |g_i| \geq \alpha\},$$

where $\alpha \in (0, n^{-1/2}]$ is a fixed number. Here S_1 is the unit sphere, G is a set of vertices of the unit cube in $\mathbb{R}^n$ and P is a set of univariate positive infinitesimal functions.

We define operators $H_i^j : \mathbb{R}^n \to \mathbb{R}^n$ for $i = 1, \dots, n, \ j = 0, \dots, n$ by the formula

$$H_i^j g = \begin{cases} (g_1, \ldots, g_j, 0, \ldots, 0) & \text{if } j < i, \\ (g_1, \ldots, g_{i-1}, 0, g_{i+1}, \ldots, g_j, 0, \ldots, 0) & \text{if } j \geq i. \end{cases}$$

We can see that $H_i^0 g = 0 \in \mathbb{R}^n$ for all $i = 1, \ldots, n$. Let $e(\beta) = (\beta e_1, \beta^2 e_2, \ldots, \beta^n e_n)$, $\beta \in (0, 1]$. For $x \in \mathbb{R}^n$ we will consider vectors

$$x_i^j(g) \equiv x_i^j(g, e, z, \lambda, \beta) = x + \lambda g - z(\lambda) H_i^j e(\beta),$$

where $g \in S_1$, $e \in G$, $i \in I(g, \alpha)$, $z \in P$, $\lambda > 0$, $\beta \in (0, 1]$, $j = 0, \ldots, n$, $j \neq i$.

Definition 7 *([4]) The discrete gradient of the function f at the point $x \in \mathbb{R}^n$ is the vector*

$$\Gamma^i(x, g, e, z, \lambda, \beta) = (\Gamma_1^i, \ldots, \Gamma_n^i) \in \mathbb{R}^n, g \in S_1, i \in I(g, \alpha),$$

with the following coordinates:

$$\Gamma_j^i = [z(\lambda) e_j(\beta)]^{-1} \left[f(x_i^{j-1}(g)) - f(x_i^j(g)) \right],$$
$$j = 1, \ldots, n, j \neq i,$$
$$\Gamma_i^i = (\lambda g_i)^{-1} [f(x_i^n(g)) - f(x) - \sum_{j=1, j \neq i}^{n} \Gamma_j^i (\lambda g_j - z(\lambda) e_j(\beta))].$$

From the definition of the discrete gradient we can see that it is defined with respect to a given direction $g \in S_1$ and in order to calculate the discrete gradient we use step $\lambda > 0$ along this direction. The $n - 1$ coordinates of the discrete gradient are defined as finite difference estimates to a gradient in some neighborhood of the point $x + \lambda g$. The ith coordinate of the discrete gradient is defined so that to approximate a sub-gradient of the function f. Thus the discrete gradient contains some information about the behavior of the function f in some region around the point x.

Now we will consider the following unconstrained minimization problem:

$$\text{minimize } f(x) \text{ subject to } x \in \mathbb{R}^n \tag{5}$$

where the function f is assumed to be locally Lipschitz continuous. We consider the discrete gradient method for solving this problem. An important step in this method is the computation of a descent direction of the objective function f. So first, we describe an algorithm for the computation of the descent direction of the function f.

Let $z \in P$, $\lambda > 0$, $\beta \in (0, 1]$, the number $c \in (0, 1)$ and a small enough number $\delta > 0$ be given.

Algorithm 1 An algorithm for the computation of the descent direction.
Step 1. Choose any $g^1 \in S_1$, $e \in G$, $i \in I(g^1, \alpha)$ and compute a discrete gradient $v^1 = \Gamma^i(x, g^1, e, z, \lambda, \beta)$. Set $\overline{D}_1(x) = \{v^1\}$ and $k = 1$.
Step 2. Calculate the vector $\|w^k\| = \min\{\|w\| : w \in \overline{D}_k(x)\}$. If

$$\|w^k\| \leq \delta, \tag{6}$$

then stop. Otherwise go to Step 3.
Step 3. Calculate the search direction by $g^{k+1} = -\|w^k\|^{-1} w^k$.
Step 4. If

$$f(x + \lambda g^{k+1}) - f(x) \leq -c\lambda \|w^k\|, \tag{7}$$

then stop. Otherwise go to Step 5.
Step 5. Calculate a discrete gradient $v^{k+1} = \Gamma^i(x, g^{k+1}, e, z, \lambda, \beta)$, $i \in I(g^{k+1}, \alpha)$, construct the set $\overline{D}_{k+1}(x) = \text{co}\,\{\overline{D}_k(x) \bigcup \{v^{k+1}\}\}$, set $k = k + 1$ and go to Step 2.

The algorithm contains steps which deserve some explanations. In Step 1 we take any direction $g^1 \in S_1$ and calculate the first discrete gradient. In Step 2 we calculate least distance between the convex hull of the discrete gradients and the origin. This problem is reduced to a quadratic programming problem and can be effectively solved by Wolfe's terminating algorithm [30]. If this distance is less than some tolerance $\delta > 0$, the algorithm stops and we can consider this point as an approximated stationary point. Otherwise, in Step 3, a search direction is calculated. If this direction is a descent direction, the algorithm terminates, otherwise, in Step 5, we calculate a new discrete gradient with respect to this direction to improve the approximation of the set of generalized gradients. Since the discrete gradient contains some information about the behavior of the function f in some regions around the point x this algorithm allows to find descent directions in stationary points which are not local minima (descent directions in such stationary point always exist). This property makes the discrete gradient method attractive for design of hybrid methods in global optimization. It is proved that Algorithm 1 is a terminating (see [4]).

Now we can describe the discrete gradient method. Let numbers $c_1 \in (0, 1)$, $c_2 \in (0, c_1]$ be given.

Algorithm 2 Discrete gradient method
Step 1. Choose any starting point $x^0 \in \mathbb{R}^n$ and set $k = 0$.
Step 2. Set $s = 0$ and $x_s^k = x^k$.
Step 3. Apply Algorithm 1 for the calculation of the descent direction at $x = x_s^k$, $\delta = \delta_k$, $z = z_k$, $\lambda = \lambda_k$, $\beta = \beta_k$, $c = c_1$. After termination of this algorithm for some

finite $m > 0$ are computed an element $\|v_s^k\| = \min\{\|v\| : v \in \overline{D}_m(x_s^k)\}$ and a search direction $g_s^k = -\|v_s^k\|^{-1}v_s^k$ such that either

$$f(x_s^k + \lambda_k g_s^k) - f(x_s^k) \leq -c_1\lambda_k\|v_s^k\| \tag{8}$$

or $\|v_s^k\| \leq \delta_k$.

Step 4. If

$$\|v_s^k\| \leq \delta_k \tag{9}$$

then set $x^{k+1} = x_s^k$, $k = k+1$ and go to Step 2. Otherwise go to Step 5.

Step 5. Construct the following iteration $x_{s+1}^k = x_s^k + \sigma_s g_s^k$, where σ_s is defined as follows

$$\sigma_s = \arg\max\{\sigma \geq 0 : f(x_s^k + \sigma g_s^k) - f(x_s^k) \leq -c_2\sigma\|v_s^k\|\}.$$

Step 6. Set $s = s+1$ and go to Step 3.

The main steps in this algorithm are Steps 3 and 5. In Step 3 we calculate a descent direction using Algorithm 1. The stepsize is calculated in Step 5. For the point $x^0 \in \mathbb{R}^n$ we consider the set

$$M(x^0) = \{x \in \mathbb{R}^n : f(x) \leq f(x^0)\}.$$

Theorem 2.1 *Assume that the set $M(x^0)$ is bounded for starting points $x^0 \in \mathbb{R}^n$. Then every accumulation point of $\{x^k\}$ belongs to the set $X^0 = \{x \in \mathbb{R}^n : 0 \in \partial f(x)\}$.*

2.2 Evolutionary strategies

Evolutionary strategies (ESs) [24, 25] were invented for numerical optimization. Let $\vec{x}$ be an n dimensional solution vector $(x_1, x_2, \ldots, x_n)$ for problem $P1$ and $\vec{\sigma}$ be the corresponding step–length $(\sigma_1, \sigma_2, \ldots, \sigma_n)$. Let μ be the number of parents, where each parent z_k is the pair $(\vec{x}_k, \vec{\sigma}_k)$. In the first generation, μ parents are generated at random. In each subsequent generation, λ children are generated from the μ parents through recombination and mutation as follows: let $y_j = (\vec{x}_j, \vec{\sigma}_j)$ be the j^{th} child to be generated from the two parents $z_k = (\vec{x}_k, \vec{\sigma}_k)$ and $z_l = (\vec{x}_l, \vec{\sigma}_l)$. The child is generated either by discrete recombination or arithmetic recombination as follows: for each variable x_{ji} in $\vec{x}_j$, do $x_{ji} = x_{ki}$ or x_{li} for discrete recombination or $x_{ji} = (x_{ki} + x_{li})/2$ for arithmetic recombination. The same recombination takes place for the step–size vectors σ. The child is then mutated as follows:

$$\vec{x'}_j = \vec{x}_k + \vec{R}_k \tag{10}$$

$$\vec{\sigma'}_j = \vec{\sigma}_k \tag{11}$$

where $\vec{R}_k$ is a random vector according to a Gaussian distribution with zero mean and standard deviation $\vec{\sigma}_k$; that is the probability, $Prob(R_{ki})$, of the random number $R_{ki} \in \vec{R}_k$ is

$$Prob(R_{ki}) = \frac{1}{\sqrt{2\pi}\sigma_{ki}} \exp^{-\frac{R_{ki}^2}{2\sigma_{ki}}} \tag{12}$$

Two variations of ESs exist based on the replacement mechanism. The first variation is $ES(\mu + \lambda)$, where λ children are generated from the μ parents then the parent in the next generation are the best solutions among the $\mu + \lambda$ solutions. The second variation is $ES(\mu, \lambda)$, where λ children are generated from the μ parents then the parent in the next generation are the best solutions among the λ solutions. In this second variation, $\lambda >> \mu$. A special case of each variation is usually used where both μ and λ equal each to 1.

The step–size σ can vary during the evolutionary process. In this case, the algorithm is called self–adaptive evolutionary strategy. The well–known one–fifth success rule is usually used, where the step–size increases if the ratio of successful mutations (mutations which produced children better than their parents) to all mutations is greater than 1/5. In [26], the lognormal self–adaptation is proposed, where a rotation angle is used to adapt the search towards coordinates which are likely to be correlated. Given τ and τ', the step size σ_j is perturbed as follows

$$\sigma'_{ji} = \sigma_{ji}\exp^{(\tau' N(0,1) + \tau N_j(0,1))} \tag{13}$$

where, the values of τ and τ' are suggested to be

$$\tau = \frac{1}{\sqrt{\left(2\sqrt{(n)}\right)}} \tag{14}$$

$$\tau' = \frac{1}{\sqrt{(2n)}} \tag{15}$$

The complete self–adaptive evolutionary strategy algorithms are depicted below.

Algorithm 3 *The self–adaptive evolutionary strategy ($\mu + \lambda$).*

Step 0. Randomly generate μ parents, where each parent $z_k = (\vec{x}_k, \vec{\sigma}_k)$.

Step 1. Set $\tau = \left(\sqrt{\left(2\sqrt{(n)}\right)}\right)^{-1}$ and $\tau' = \left(\sqrt{(2n)}\right)^{-1}$.

Step 2. Until λ children are generated, do

Step 3. Select two parents $z_k = (\vec{x}_k, \vec{\sigma}_k)$ and $z_l = (\vec{x}_l, \vec{\sigma}_l)$ at random to generate child $\vec{y}_j = (\vec{x}_j, \vec{\sigma}_j)$.

Step 4. Discrete recombination: for each variable x_{ji} and step size σ_{ji} in $\vec{y}_j$, do ($x_{ji} = x_{ki}$ and $\sigma_{ji} = \sigma_{ki}$) or ($x_{ji} = x_{li}$ and $\sigma_{ji} = \sigma_{li}$)

Step 5. Mutation: For each x_{ji} and step size σ_{ji} in $\vec{y}_j$

$$x'_{ji} = x_{ji} + \sigma_{ji} N_j(0,1) \tag{16}$$

$$\sigma'_{ji} = \sigma_{ji} \exp(\tau' N(0,1) + \tau N_j(0,1)) \tag{17}$$

Step 6. If the number of children is less than λ, go to 3.

Step 7. Select the best μ individuals among all the $\mu + \lambda$ parents and children.

Step 8. If the halting criteria are satisfied, stop, else go to step 1.

Algorithm 4 *The self-adaptive evolutionary strategy (μ, λ).*

Step 0. Randomly generate μ parents, where each parent $z_k = (\vec{x}_k, \vec{\sigma}_k)$.

Step 1. Set $\tau = \left(\sqrt{\left(2\sqrt{(n)}\right)}\right)^{-1}$ and $\tau' = \left(\sqrt{(2n)}\right)^{-1}$.

Step 2. Until λ children are generated, do

Step 3. Select two parents $z_k = (\vec{x}_k, \vec{\sigma}_k)$ and $z_l = (\vec{x}_l, \vec{\sigma}_l)$ at random to generate child $\vec{y}_j = (\vec{x}_j, \vec{\sigma}_j)$.

Step 4. Discrete recombination: for each variable x_{ji} and step size σ_{ji} in $\vec{y}_j$, do ($x_{ji} = x_{ki}$ and $\sigma_{ji} = \sigma_{ki}$) or ($x_{ji} = x_{li}$ and $\sigma_{ji} = \sigma_{li}$)

Step 5. Mutation: For each x_{ji} and step size σ_{ji} in $\vec{y}_j$

$$x'_{ji} = x_{ji} + \sigma_{ji} N_j(0,1) \tag{18}$$

$$\sigma'_{ji} = \sigma_{ji} \exp(\tau' N(0,1) + \tau N_j(0,1)) \tag{19}$$

Step 6. If the number of children is less than λ, go to 3.

Step 7. Select the best μ individuals among the λ children.

Step 8. If the halting criteria are satisfied, stop, else go to step 1.

2.3 Hybridization of evolutionary strategies and the discrete gradient method

In this method, we use discrete gradient as a local search operator once for $(\mu + \lambda)$ ES and another time for (μ, λ) ES. The algorithm simply works by applying the discrete gradient on all individuals in the population of the initial generation. In subsequent generations, discrete gradient is applied only for the best solutions found so far. We will call $(\mu + \lambda)$ with DG for local search $Algorithm 5$ and (μ, λ) with DG for local search $Algorithm 6$. The details of the algorithm for discrete gradient with $(\mu + \lambda)$ are as follows:

Algorithm 5 *DG($\mu + \lambda$).*

Step 0. Randomly generate μ parents, where each parent $z_k = (\vec{x}_k, \vec{\sigma}_k)$.

Step 1. Apply discrete gradient on each parent.

Step 2. Set $\tau = \left(\sqrt{\left(2\sqrt{(n)}\right)}\right)^{-1}$ and $\tau' = \left(\sqrt{(2n)}\right)^{-1}$.

Step 3. Until λ children are generated, do

Step 4. Select two parents $z_k = (\vec{x}_k, \vec{\sigma}_k)$ and $z_l = (\vec{x}_l, \vec{\sigma}_l)$ at random to generate child $\vec{y}_j = (\vec{x}_j, \vec{\sigma}_j)$.

Step 5. Discrete recombination: for each variable x_{ji} and step size σ_{ji} in $\vec{y}_j$, do ($x_{ji} = x_{ki}$ and $\sigma_{ji} = \sigma_{ki}$) or ($x_{ji} = x_{li}$ and $\sigma_{ji} = \sigma_{li}$)

Step 6. Mutation: For each x_{ji} and step size σ_{ji} in $\vec{y}_j$

$$x'_{ji} = x_{ji} + \sigma_{ji} N_j(0,1) \tag{20}$$

$$\sigma'_{ji} = \sigma_{ji} \exp(\tau' N(0,1) + \tau N_j(0,1)) \tag{21}$$

Step 7. If the number of children is less than λ, go to 4.

Step 8. Select the best μ individuals among all the $\mu + \lambda$ parents and children.

Step 9. Apply discrete gradient on the best individual among the selected μ individuals.

Step 10. If the halting criteria are satisfied, stop, else go to step 2.

Algorithm 6 *DG(μ, λ).*

Step 0. Randomly generate μ parents, where each parent $z_k = (\vec{x}_k, \vec{\sigma}_k)$.

Step 1. Apply discrete gradient on each parent $\vec{x}_k$.

Step 2. Set $\tau = \left(\sqrt{\left(2\sqrt{(n)}\right)}\right)^{-1}$ and $\tau' = \left(\sqrt{(2n)}\right)^{-1}$.

Step 3. Until λ children are generated, do

Step 4. Select two parents $z_k = (\vec{x}_k, \vec{\sigma}_k)$ and $z_l = (\vec{x}_l, \vec{\sigma}_l)$ at random to generate child $\vec{y}_j = (\vec{x}_j, \vec{\sigma}_j)$.

Step 5. Discrete recombination: for each variable x_{ji} and step size σ_{ji} in $\vec{y}_j$, do ($x_{ji} = x_{ki}$ and $\sigma_{ji} = \sigma_{ki}$) or ($x_{ji} = x_{li}$ and $\sigma_{ji} = \sigma_{li}$)

Step 6. Mutation: For each x_{ji} and step size σ_{ji} in $\vec{y}_j$

$$x'_{ji} = x_{ji} + \sigma_{ji} N_j(0, 1) \quad (22)$$

$$\sigma'_{ji} = \sigma_{ji} \exp(\tau' N(0, 1) + \tau N_j(0, 1)) \quad (23)$$

Step 7. If the number of children is less than λ, go to 4.

Step 8. Select the best μ individuals among the λ children.

Step 9. Apply discrete gradient on the best individual among the selected μ individuals.

Step 10. If the halting criteria are satisfied, stop, else go to step 2.

3 Test problems

Table 1 lists the 32 test problems we used in this paper, their corresponding number of variables, and the best known minimum for each of them. Each problem was run for a maximum of a million objective evaluations and population size of 100. Each run was repeated a 100 times with different initial seed. The standard deviation for the ES methods was initialized to 3. Local search for the hybrid methods was carried out for each solution for 50 objective evaluations (these 50 was included in the calculations of the maximum number of objective evaluations per run). All variables were uniformly initialized within their boxing constraints. All runs were carried out on a PC running WindowsXP with 512 memory and a Pentium 4 CPU.

4 Numerical Experiments

In this section, we present the results obtained for the 32 global optimization problems presented in the previous section. In this paper, we will restrict out discussion to the quality of solutions without considering the speed for considerations of space. Table 2 presents the results for ES(μ, λ) and ES(μ, λ) with local search. Table 3 presents the results for ES($\mu + \lambda$) and ES($\mu + \lambda$) with local search. Lastly, Table 4 lists the results for discrete gradient and identifies the best performing algorithm(s) on each problem.

It is interesting to note that the discrete gradient method on its own was quite competitive on some problems and consistently reached the global optimum. More interestingly, on problem F12, the discrete gradient method was the winner and solutions obtained by the other methods were inferior for F12. It is worth noting that the hybrid methods were better on F12 than when using the evolutionary strategies in isolation.

Table 1: The 32 test problems used in this paper.

Function	# of variables	known minimum
F1 ([21]: problem3)	2	-186.7309
F2 ([21, 15]: Griewanks Function)	10	0
F3 ([2]: Ackleys)	10	0
F4 ([15]: or Bohachevsky)	2	0
F5 ([15]: or Bohachevsky)	2	0
F6 ([15]: or Bohachevsky)	2	0
F7 ([2]: Branin)	2	0.399
F9 ([2]: Easom)	2	-1
F10 ([1]: Goldstein and Price)	2	3
F11 ([2]: Hartman with $n = 3$)	3	-3.86278
F12 ([2]: Hartman with $n = 6$)	6	-3.32237
F13 ([14]: Hump)	2	0
F15 ([32]: Levy No 1)	10	0
F16 ([32]: Levy No 2)	10	0
F17 ([15]: Michalewicz)	2	-1.8013
F18 ([2]: Neumaier 2)	4	0
F20 ([2]: Rastringins)	10	0
F22 ([2]: Schaffer 1)	2	0
F23 ([2]: Schaffer 2)	2	0
F24 ([2]: Shekel-5)	4	-10.15320
F25 ([2]: Shekel-7)	4	-10.40294
F26 ([2]: Shekel-10)	4	-10.53641
F27 ([2]: Shubert 1)	2	-186.7309
F28 ([32]: Shubert 2)	2	-186.7309
F29 ([2]: Step)	10	0
F30 ([15]: Zakharov)	10	0
F32 ([2]: f2)	10	0
F33 ([2]: f3)	10	0
F35 ([2]: f6)	10	0
F36 ([2]: f7)	10	0
F37 ([2]: f8)	10	-
F38 ([2]: f13)	10	0

Table 2: The performance of Alg4 and Alg6 on the 32 problems. The best average overall algorithms is underlined.

Function	Alg6		Alg4	
	Best	Avg ± Std	Best	Avg ± Std
F1	-186.73	$\underline{-186.42 \pm 2.81}$	-185.39	-81.82 ± 41.21
F2	0.06	1.91 ± 3.27	0	$\underline{0 \pm 0}$
F3	0.22	0.69 ± 0.28	0	$\underline{0 \pm 0}$
F4	0.12	$\underline{0.12 \pm 0}$	0.14	0.77 ± 0.12
F5	0	$\underline{0 \pm 0}$	0	$\underline{0 \pm 0}$
F6	0	$\underline{0 \pm 0}$	0	$\underline{0 \pm 0}$
F7	0.4	$\underline{0.4 \pm 0}$	0.4	1.07 ± 0.64
F9	-1	-0.98 ± 0.14	-0.15	0 ± 0
F10	3	$\underline{3 \pm 0}$	3	6.77 ± 6.94
F11	-3.86	$\underline{-3.86 \pm 0}$	-3.86	-3.62 ± 0.18
F12	-3.32	$\underline{-3.27 \pm 0}$	-3.04	-2.08 ± 0.43
F13	0	$\underline{0 \pm 0}$	0.02	0.6 ± 0.36
F15	0	$\underline{0 \pm 0}$	2.65	2.65 ± 0
F16	0	0.05 ± 0.13	2.65	2.65 ± 0
F17	-1.8	$\underline{-1.8 \pm 0}$	-1.8	-1.38 ± 0.28
F18	0	0.05 ± 0	15320	15320 ± 0
F20	0	6.34 ± 4.9	0	$\underline{0 \pm 0}$
F22	0	$\underline{0 \pm 0}$	0	$\underline{0 \pm 0}$
F23	0	$\underline{0 \pm 0}$	0	$\underline{0 \pm 0}$
F24	-40.61	-14.82 ± 9.84	-2.05	-1.31 ± 0.14
F25	-41.55	-17.73 ± 10.85	-2.22	-1.35 ± 0.17
F26	-10.54	-10.42 ± 0.41	-2.35	-1.4 ± 0.2
F27	-186.73	$\underline{-186.73 \pm 0}$	-185.39	-81.82 ± 41.21
F28	-186.73	$\underline{-186.72 \pm 0}$	-170.47	-64.09 ± 37.55
F29	30	$\underline{30 \pm 0}$	31	39.58 ± 2.63
F30	0.09	3.04 ± 3.37	0	$\underline{0 \pm 0}$
F32	0	0.01 ± 0	0	$\underline{0 \pm 0}$
F33	0	0.09 ± 0.49	0	$\underline{0 \pm 0}$
F35	1	106.33 ± 185.36	0	$\underline{0 \pm 0}$
F36	0	0.02 ± 0	0	$\underline{0 \pm 0}$
F37	-7139.76	$\underline{-6256.67 \pm 531.96}$	-2275.75	-1401.88 ± 243.62
F38	0	$\underline{0 \pm 0}$	1	1 ± 0

When counting the number of times an algorithm

Table 3: The performance of Alg3 and Alg5 on the 32 problems. The best average overall algorithms is underlined.

Function	Alg5		Alg3	
	Best	Avg±Std	Best	Avg±Std
F1	-186.73	-186.73 ± 41.21	-185.39	-81.82 ± 0
F2	0	0.01 ± 0	0	0 ± 0
F3	0	0.05 ± 0	0	0 ± 0
F4	0.12	0.12 ± 0.12	0.14	0.77 ± 0
F5	0	0 ± 0	0	0 ± 0
F6	0	0 ± 0	0	0 ± 0
F7	0.4	0.4 ± 0.64	0.4	1.07 ± 0
F9	-1	-1 ± 0	-0.15	0 ± 0
F10	3	3 ± 6.94	3	6.77 ± 0
F11	-3.86	-3.86 ± 0.18	-3.86	-3.62 ± 0
F12	-3.32	-3.31 ± 0.43	-3.04	-2.08 ± 0
F13	0	0 ± 0.36	0.02	0.6 ± 0
F15	0	0 ± 0	2.65	2.65 ± 0
F16	0	0 ± 0	2.65	2.65 ± 0
F17	-1.8	-1.8 ± 0.28	-1.8	-1.38 ± 0
F18	0	0.04 ± 0	15320	15320 ± 0
F20	0	0.24 ± 0	0	0 ± 0.47
F22	0	0 ± 0	0	0 ± 0
F23	0	0 ± 0	0	0 ± 0
F24	-40.61	-27.06 ± 0.14	-2.05	-1.31 ± 11.36
F25	-41.6	-36.23 ± 0.17	-2.22	-1.35 ± 10.41
F26	-10.54	-10.54 ± 0.2	-2.35	-1.4 ± 0
F27	-186.73	-186.73 ± 41.21	-185.39	-81.82 ± 0
F28	-186.73	-186.71 ± 37.55	-170.47	-64.07 ± 0
F29	30	30 ± 2.63	31	39.58 ± 0
F30	0	0 ± 0	0	0 ± 0
F32	0	0 ± 0	0	0 ± 0
F33	0	0 ± 0	0	0 ± 0
F35	0	0 ± 0	0	0 ± 0
F36	0	0 ± 0	0	0 ± 0
F37	-4485.78	-4196.58 ± 243.62	-2275.75	-1401.88 ± 42.35
F38	0	0 ± 0	1	1 ± 0

Table 4: The performance of Alg2 on the 32 problems. The best average overall algorithms is underlined. The second column lists the best algorithm(s) for each problem.

Function	Alg2		Best Algorithms
	Best	Avg±Std	
F1	-3.34	-3.34 ± 0	Alg5, Alg6
F2	47.79	47.79 ± 0	Alg3
F3	7.01	7.01 ± 0	Alg3
F4	50.8	50.8 ± 0	Alg6
F5	0.22	0.22 ± 0	Alg3, Alg5, Alg6
F6	0	0 ± 0	Alg2, Alg3, Alg5, Alg6
F7	0.4	0.4 ± 0	Alg6
F9	0	0 ± 0	Alg5
F10	84	84 ± 0	Alg6
F11	-1	-1 ± 0	Alg6
F12	-3.32	-3.32 ± 0	Alg2
F13	22.45	22.45 ± 0	Alg6
F15	5.92	5.92 ± 0	Alg5
F16	0	0 ± 0	Alg2, Alg5, Alg6
F17	-1	-1 ± 0	Alg6
F18	15320	15320 ± 0	Alg5
F20	173.07	173.07 ± 0	Alg4
F22	0.49	0.49 ± 0	Alg3, Alg5, Alg6
F23	0.09	0.09 ± 0	Alg3, Alg5, Alg6
F24	-0.23	-0.23 ± 0	Alg5
F25	-0.31	-0.31 ± 0	Alg5
F26	-0.42	-0.42 ± 0	Alg5
F27	-36.6	-36.6 ± 0	Alg6
F28	-0.79	-0.79 ± 0	Alg6
F29	67	67 ± 0	Alg6
F30	55894100	55894100 ± 0	Alg3, Alg4, Alg5
F32	0	0 ± 0	Alg2, Alg3, Alg4, Alg5
F33	0	0 ± 0	Alg2, Alg3, Alg4, Alg5
F35	43750	43750 ± 0	Alg3, Alg4, Alg5
F36	36.28	36.28 ± 0	Alg3 Alg5
F37	-2141.72	-2141.72 ± 0	Alg6
F38	0	0 ± 0	Alg2, Alg5

appeared to be the best as shown in Table 4, we find that Algorithm5 is the winner, being the best 18 times followed by Algorithm6 (16 times), Algorithm3 (11 times), Algorithm2 (6 times), and lastly Algorithm 4 (5 times). It is also interesting to see that on 28 out of the 32 problems, the hybrid algorithms consistently performed better than their non-hybrid counterparts with exceptions in four cases (F2,F3, F12, and F20).

5 Conclusions

In this paper we have introduced a new type of evolutionary strategies with local search. The discrete gradient method, a derivative free method, was integrated into the evolutionary strategies. The hybrid approach seems to perform better than the non-hybrid approach on the majority of the problems being presented here. The discrete gradient method has a major advantage over traditional gradient-based local search techniques, in the sense that it does not require an explicit gradient and it can work in certain cases, even when the true gradient does not exist.

For future work, we are planning to provide a detailed analysis to the performance of these methods and compare them with other global optimization techniques as well as hybrid techniques from the evolutionary computation literature such as the local evolutionary search enhancement by random memorizing [29] and landscape approximation and local search [19, 18].

Bibliography

[1] Online access. http://www.geatbx.com/docu/fcngold.html.

[2] Test functions for global optimization. http://www.mat.univie.ac.at/~neum/glopt/janka/funcs.html.

[3] N. Ansari and E. Hou. *Computational intelligence for optimization*. Kluwer Academic Publisher, 1997.

[4] A.M. Bagirov. Derivative-free methods for unconstrainted nonsmooth optimization and its numerical analysis. *Investigacao Operacional*, 19:75–93, 1999.

[5] A.M. Bagirov and A.M. Rubinov. Cutting angle method and a local search. *Journal of Global Optimization, to appear*.

[6] P. Basso. Iterative methods for the localization of the global optimum. *SIAM Journal on Numerical Analysis*, 19:781–792, 1982.

[7] R.W. Becker and G.V. Lago. A global optimization algorithm. In *Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, pages 3–12, 1970.

[8] F.H. Branin. Widely convergent methods for finding multiple solutions of simultaneous nonlinear equations. *IBM Journal of Research Developments*, pages 504–522, 1972.

[9] R.P. Brent. *Algorithms for minimization without derivatives*. Prentice–Hall, 1973.

[10] F. Clarke. *Optimization and Nonsmooth Analysis*. John Wiley & Sons, New York, 1983.

[11] A.A. Goldstein and J.F. Price. On descent from local minima. *Mathematics of Computation*, 25:569–574, 1971.

[12] S. Gomez and A.V. Levy. The tunneling method for solving the constrained global optimization problem with several non-connected feasible regions. In *Lecturer notes in mathematics 909*, number 3, pages 34–47. Springer-Verlag, 1982.

[13] W.E. Hart. *Adaptive global optimization with local search*. PhD thesis, University of California, San Diago, 1994.

[14] A.-R. Hedar and M. Fukushima. Hybrid simulated annealing and direct search method for nonlinear global optimization. Technical Report 2001-013, Department of Applied Mathematics and Physics, Kyoto University, December 2001.

[15] A.-R. Hedar and M. Fukushima. Simplex coding genetic algorithm for the global optimization of nonlinear functions. Technical Report 2002-006, Department of Applied Mathematics and Physics, Kyoto University, March 2002.

[16] J.-B. Hiriart-Urruty and C. Lemarechal. *Convex Analysis and Minimization Algorithms*. Springer-Verlag, Berlin, New York, 1993.

[17] M.J. Kushner. A new method of locating the maximum point of arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.

[18] K.H. Liang. *Evolutionary Optimization with Self-Adapatation and Landscape Approximation*. PhD thesis, School of Computer Science, University of New South Wales at ADFA, 2000.

[19] K.H. Liang, X. Yao, and C. Newton. Combining landscape approximation and local search in global optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 1514–1520, Mayflower Hotel, Washington D.C., USA, 6-9 1999. IEEE Press.

[20] J. Mockus. The Bayesian approach to global optimization. Technical Report Report 176, Freie Universität Berlin, 1984.

[21] A. Neumaier. Optimization test problems. Publicly available from: solon.cma.univie.ac.at/ñeum/glopt.html.

[22] S.A. Pijavskij. An algorithm for finding the global extremum of a function. *Optimal Decision Theory 2*, pages 13–24, 1967.

[23] W.L. Price. A controlled random search procedure for global optimization. In L.C.W. Dixon and G.P. Szegö, editors, *Towards global optimization 2*, pages 71–84. North-Holland, Amsterdam, 1978.

[24] I. Rechenberg. *Cybernetic solution path of an experimental problem*. Royal Aircraft Establishment, Library Translation 1122, 1965.

[25] I. Rechenberg. *Evolutionsstrategie - optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann–Holzboog, 1973.

[26] H.P. Schwefel. *Numerical optimization of computer models*. Wiler, 1981.

[27] A. Törn. Probabilistic global optimization, a cluster analysis approach. In M. Roubens, editor, *Proceedings of the second European congress on operations research*, pages 521–527. North-Holland, N.Y., 1976.

[28] A. Törn and A. Žilinskas. Global optimization. In G. Goos and J. Hartmanis, editors, *Lecturer Notes in Computer Science*. Springer-Verlag, 1987.

[29] H.M. Voigt and J.M. Lange. Local evolutionary search enhancement by random memorizing. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, pages 547–552, Piscataway, NJ., 1989. IEEE Press.

[30] P. Wolfe. Finding the nearest point in a polytope. *Mathematical Programming*, 11(2):128–149, 1976.

[31] G.R. Wood. Multidimensional bisection and global minimization. Technical report, University of Canterburry, 1985.

[32] K.F.C. Yiu, Y. Liu, and K.L. Teo. A hybrid descent method for global optimization. *Journal of Global Optimization, to appear.*