Communication papers of the Federated Conference on Computer Science and Information Systems, pp. 103–110 ISSN 2300-5963 ACSIS, Vol. 13
DOI: 10.15439/2017F119

The Revised Stochastic Simplex Bisection Algorithm and Particle Swarm Optimization

Christer Samuelsson
25A rue des Sports
57200 Sarreguemines, FRANCE
Email: csamuelsson42@yahoo.com

***Abstract*—The stochastic simplex bisection (SSB) algorithm is evaluated against particle swarm optimization (PSO) on a prominent test set. The original SSB algorithm performs on par with the PSO algorithm and a revised version of the SSB algorithm outperforms both of them. Detailed analysis of the performance on select objective functions brings to light key properties of the three algorithms. The core SSB algorithm is here viewed as a sampling tool for an outer loop that employs statistical pattern recognition. This opens the door for a host of other schemes.**

I. Introduction

STOCHASTIC optimization [1] involves introducing some degree of randomness, when searching for optima. Particularly successful approaches include genetic algorithms [2], particle swarm optimization [3], and ant-colony optimization [4]. Such schemes are often applied to continuous optimization problems, especially when the gradient and Hessian of the objective functions are not readily available. In addition to this they have proven effective when optimizing highly multimodal functions, i.e., function with a large number of optima.

We here compare the performance of the stochastic simplex bisection (SSB) algorithm—first proposed in [5] and first evaluated against other optimization algorithms in [6]—with a particle swarm optimizer (PSO). The former employs a common stochastic optimization scheme, but unlike other stochastic approaches, it applies the scheme to search space regions, rather than to individual points. The latter is a well-known global optimization scheme, which has spawned a number of related schemes, such as firefly optimization [7] and fish school search [8]. This is the first time the SSB algorithm is tested against another state-of-the-art global optimizer.

Two versions of the SSB algorithm were tested: the original one from [5], [6] and a new one, which uses the same core SSB algorithm and a modified outer loop that clusters a shrinking set of arguments with relatively low function values.

The rest of the article is organized as follows. Section II describes the new and old SSB algorithms. Section III details the PSO optimizer used. Section IV discusses the experimental setup, and reports and analyses the findings.

II. The Stochastic Simplex Bisection Algorithm

Consider the simple problem where we wish to minimize $f(x)$, which is strictly convex on $\mathbf{R}$. Assume further that we have found $x_1 < x_3 < x_5$, where $f(x_1) \geq f(x_3) \leq f(x_5)$, i.e., that we have found an interval that has an interior point x_3 with a smaller function value than its end points. [1]

Algorithm for Convex Functions

Given $x_1 < x_3 < x_5$ **with** $f(x_1) \geq f(x_3)$ **and** $f(x_3) \leq f(x_5)$

1: **Select** $x_2 \in]x_1, x_3[$ **and** $x_4 \in]x_3, x_5[$
2: **if** $f(x_2) \leq f(x_3)$ **then**
3: **Recurse on** x_1, x_2, x_3
4: **else if** $f(x_4) \leq f(x_3)$ **then**
5: **Recurse on** x_3, x_4, x_5
6: **else**
7: **Recurse on** x_2, x_3, x_4
8: **end if**

We thus recurse on the subinterval that has an interior point with a smaller function value than its end points. [2]

A. The Core SSB Algorithm

The SSB algorithm generalizes this to non-convex functions in n dimensions. An interval is generalized to a simplex, rather than to a hyperbox, The latter has 2^n corners, and bisecting it requires computing the function value in 2^{n-1} new corners. The former has only $n+1$ corners, and bisecting it only requires calculating the function value in one new point.

Core SSB Step

Given a set $\{T_{k'}\}$ of non-overlapping simplexes,

1: **Select** the next simplex T_k to bisect at random with probability $\frac{s_k}{\sum_{k'} s_{k'}}$.
2: **Select** a bisection point at random roughly in the middle of the longest edge of T_k.
3: Bisect T_k, yielding two new simplexes.
4: **Replace** T_k in $\{T_{k'}\}$ with its two offspring.

The simplex score s_k, which is defined in Section II-A1 below, requires the function value in the simplex midpoint. Thus only three new function values need be calculated for each bisection: that of the bisection point and those of the midpoints of the two new simplexes, see Section II-A2.

The core SSB algorithm starts with a simplex T_0 and maintains a partition of it by repeatedly performing the core SSB

[1] By strict convexity, one of the two end point values must be strictly larger, i.e., either $f(x_1) > f(x_3)$ or $f(x_3) < f(x_5)$, or both.

[2] If $f(x_2) = f(x_3)$, the algorithm could be clever and instead recurse on x_2, x_{23}, x_3, with $x_{23} \in]x_2, x_3[$, where, by necessity $f(x_2) > f(x_{23}) < f(x_3)$, due to strict convexity. Similarly for $f(x_4) = f(x_3)$.

step, until some termination criterion is met. It then returns the best point found thus far. The algorithm is complete in the sense that no portion of the search space is ever discarded, and it avoids redundancy by using non-overlapping simplexes. These are two common pitfalls of stochastic optimization. The algorithm still reaps the benefits of stochastic search in exploring more promising regions earlier, in average, while granting also less promising regions a non-zero chance of being explored.

1) The Simplex Score: We define the simplex scores s_k as follows. Let $\{T_k = \langle \mathbf{x}_k^{(1)}, \ldots, \mathbf{x}_k^{(n+1)} \rangle\}$ be a collection of n-dimensional simplexes with (dropping the index k for clarity),

$$\bar{\mathbf{x}} \quad = \quad \frac{1}{n+1} \sum_{i=1}^{n+1} \mathbf{x}^{(i)} \quad ; \quad \bar{f} \quad = \quad \frac{f(\bar{\mathbf{x}}) + \sum_{i=1}^{n+1} f\left(\mathbf{x}^{(i)}\right)}{n+2}$$

where $\bar{\mathbf{x}}$ is the midpoint and $\bar{f}$ is the average function value over the corners and the midpoint. We then set

$$f^- \quad = \quad \min\left(f(\bar{\mathbf{x}}), \min_i f\left(\mathbf{x}^{(i)}\right)\right) \quad ; \quad \delta \quad = \quad \frac{\bar{f} - f^-}{4}$$

and introduce

$$f^\star \quad = \quad f^- - \delta$$

which is a combined measure of the lowest function value f^- and an estimate δ of how much it might potentially decrease, judging by the average function value $\bar{f}$ and this lowest value.

We next make $f^\star$ offset- and scale-invariant through min-max normalization, using the lowest function value f_{vb} in the very best point found this far, and some relatively high, fixed function value f_{w} found early in the search. [3]

$$f^\star \quad \leftarrow \quad \frac{f^\star - f_{\text{vb}}}{f_{\text{w}} - f_{\text{vb}}}$$

Thus, all simplexes must be rescored whenever a new very best point is found. As this happens rather seldom, it incurs very little overhead in practice.

The simplex score s is then defined as

$$s \quad = \quad l \cdot \exp\left(-\lambda_0 f^\star\right)$$

where l is the length of its longest edge:

$$l \quad = \quad \max_{ij} \left|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right|$$

The parameter λ_0, which defaults to one, controls the trade-off between exploration and exploitation. A small λ_0 will make the simplexes with different $f^\star$ more equiprobable, thus promoting exploration, a large one will do the opposite.

[3] In practice, we require that the $f^\star$ be non-negative and that the denominator $f_{\text{w}} - f_{\text{vb}}$ be at least 0.1 and at most one, yielding the expression:

$$f^\star \quad \leftarrow \quad \frac{\max\left(0, f^\star - f_{\text{vb}}\right)}{\max\left(0.1, \min\left(1, f_{\text{w}} - f_{\text{vb}}\right)\right)}$$

2) Simplex Bisection: Each round only creates two new simplexes: the longest edge of the selected simplex is bisected. All corners remain the same, save one of the two connected by this edge. Let these corners be $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. The edge bisection point $\bar{\mathbf{x}}'$—the new corner—is also randomized to counter-act any symmetries of the objective function $f(\mathbf{x})$ in its argument $\mathbf{x}$:

$$\bar{\mathbf{x}}' = (0.5 + \theta)\mathbf{x}^{(i)} + (0.5 - \theta)\mathbf{x}^{(j)} \quad \text{for} \quad \theta \sim \text{U}(-\alpha, \alpha)$$

An empirically good choice is $\alpha = 0.05$ (max 10% randomness). $\bar{\mathbf{x}}'$ replaces $\mathbf{x}^{(i)}$ in one offspring simplex and $\mathbf{x}^{(j)}$ in the other one. Thus only three new function values need be calculated for each bisection: $f(\bar{\mathbf{x}}')$ and the function values of the midpoints of the two new simplexes.

B. Outer Loop: Maintaining a Hyperbox

It is often desirable to start from a hyperbox. For example, constraints often take the form of bounds on the individual variables of each dimension. The tested SSB algorithms repeatedly restart an SSB algorithm from a hyperbox created in a previous iteration. They use an outer loop over epochs that maintains the hyperbox, and an inner loop over rounds, each of which performs the core SSB step. Note that although partitioning a hyperbox into a set of simplexes is trivial in two dimensions, it is a challenging and time-consuming task in higher ones, see [9] and [10].

The SSB algorithms thus consist of two nested loops. The outer loop over epochs maintains a hyperbox. Each *epoch* runs the inner loop over rounds, where each *round* contains one simplex bisection, which replaces one simplex with two new ones. Each epoch is divided into two phases, see below. In the first phase, the simplexes are processed as a first-in-first-out (FIFO) queue to create an initial grid. In the second phase, the next simplex to bisect is selected as in the core SSB step: at random, with probabilities proportional to the simplex scores.

Inner Loop

Given A hyperbox H.
1: Partition H into a set $\{T_{k'}\}$ of simplexes.
2: Initialize other variables as described below.
3: Convert $\{T_{k'}\}$ into FIFO queue.
4: **while not** End-of-epoch **do**
5: **if** Phase 1 **then**
6: Run Core SSB step, but select as T_k the first simplex of the FIFO queue and add its offspring to the end of the queue.
7: **else**
8: Run Core SSB step. {Phase 2}
9: **end if**
{This yields three new objective function values $f(\mathbf{x}_a), f(\mathbf{x}_c), f(\mathbf{x}_c)$.}
10: **if** $f(\mathbf{x}_{a/b/c}) < f_{\text{vb}}$ **then**
11: $f_{\text{vb}} \leftarrow f(\mathbf{x}_{a/b/c})$; $\mathbf{x}_{\text{vb}} \leftarrow \mathbf{x}_{a/b/c}$
12: Rescore all simplexes.
13: **end if**
14: Update other variables as described below.
15: **end while**

For f_{W}, we use the highest function value in a corner of the bounding box. The manner in which the hyperbox is modified after each epoch depends on the sequence of points sampled.

1) Old Scheme: The sequence order matters in the old scheme, and a single hyperbox is created. To this end, we need some terminology. A *best point* is any point found during the second phase of an epoch that is the best this far in that epoch. The best points thus start over each epoch. The *very best point*, on the other hand, is the globally best point found in any round of any epoch. Its objective function value is f_{vb}. The *first phase* of each epoch consists of the first quarter of its rounds. The rest of its rounds constitute the *second phase.* Other choices than one quarter have been tested and found effective in other settings, see [5], [6].

Initializing other variables

Let f_b be the best function value found when scoring the simplexes partitioning the hyperbox.
BestPoints $\leftarrow \emptyset$;

Updating other variables

if $f(\mathbf{x}_{a/b/c}) < f_b$ **then**
  $f_b \leftarrow f(\mathbf{x}_{a/b/c})$;
  if Phase 2 **then**
    Add $\mathbf{x}_{a/b/c}$ to BestPoints.
  end if
end if

If the elapsed epoch had enough best points, the next hyperbox is one that contains all best points and the very best point as interior points. It turns out that the simple scheme of updating the lower and higher bounds of the hyperbox in each dimension, for each new best point, works well in practice. Otherwise, the previous hyperbox is increased in size and re-centered around the current very best point. In the tested SSB algorithm, the box is padded by 100 percent of the interval length in each dimension, when there are enough best points, and the old interval length is quadrupled, when there aren't.

Creating a new hyperbox

Let H be the smallest hyperbox that contains BestPoints $\cup \{\mathbf{x}_{\mathrm{vb}}\}$.
Retain the new midpoint of H.
if the set BestPoints is large enough **then**
  Multiply the length of each side of H by some fixed margin factor (e.g., 1.1).
else
  Set the length of each side of H to a fixed factor (e.g., 2) times that the old hyperbox.
end if
for all sides of the new hyperbox H **do**
  if the side length is zero **then**
    Set that side length of H to a fixed factor (e.g., 2) times that of the old hyperbox.
  end if
end for

The old scheme loop may seem somewhat ad hoc. It captures the idea, that if there has been non-trivial local improvements, search should focus on these improvements, yet also consider the globally best point found. We note that any new best point must be a bisection point, or the midpoint of a simplex, with a lower function value than its corner points. In one dimension, these cases coincide. For convex functions, such an interval must contain the minimum, which our introductory algorithm exploits. For non-convex functions, or in several dimensions, the region surrounding such a point merits further investigation.

2) New Scheme: The next hyperbox is in the new scheme created by drawing inferences from the set of sample points using statistical pattern recognition techniques. In its current incarnation, it selects all sample points with a function value below a certain threshold. We call these *low points*. To generate more low points, they are clustered using k-means clustering, see, e.g., [11], pp. 424–430. Points far away from the cluster means are pruned from the clusters. A hyperbox is then created for each cluster and the core SSB algorithm is run on this hyperbox. This will add new sample points and thus presumably additional low points, and is done to search a region containing this cluster for an optimum. In addition, the midpoints between clusters are evaluated and added as sample points. This is done to encourage merging clusters that in fact belong to the same optimum. Since sample points are retained between epochs, their number increases steadily.

The number of rounds in each epoch, where each round requires three function evaluations, is constant and distributed between running the core SSB algorithm on a hyperbox derived from the entire set of low points and on the hyperboxes derived from individual clusters. We found that using half of the rounds for each purpose, and thus one half divided by the number of viable clusters for each of the latter, was appropriate. Thus, compared to the old scheme, fewer rounds are used for global exploration.

The final hyperbox used in the next epoch contains the set of low points. To gradually reduce it in size, the threshold is reduced from epoch to epoch to shrink the relative number of low points. In the first epoch, n_i points, say all points, are included; in the last one, only n_f, say three points, are. For intermediate epochs we calculate the rank of the highest function value to be included as

$$r(x) \;=\; \frac{n_i}{1 + c \cdot x^4} \text{ with } c \;=\; \frac{n_i}{n_f} - 1 \text{ and } x \;=\; \frac{\text{epoch}}{\text{maxEpoch}}$$

It is clear that $r(0) = n_i$ and that $r(1) = n_f$, independent of the power that x is raised to; four was found empirically to be a good choice. This successively reduces the number of low points, and hopefully forces the algorithm to eventually commit to one cluster containing the global optimum. Thus, the algorithm initially performs global exploration, but gradually resorts to searching a shrinking number of low points and clusters, and eventually focuses on exploiting a small number of low points and very few—if not a single—clusters, to converge on a potential global optimum.

This is an example of using a level curve—aka a contour curve, called a level (hyper-)surface in higher dimensions—of the function. In this case, we select all points inside it. Current

work includes applying statistical recognition of level surfaces to identify regions that warrant further investigation, as well as the use of more sophisticated clustering techniques.

C. Related Work

The SSB algorithm uses a typical stochastic optimization scheme. It maintains a set of elements, each with a positive score; randomly selects some elements based on the scores; uses these elements to explore the search space, often creating new elements in the process; and updates the set of elements and their scores according to the findings. The scheme is however here applied to regions of the search space, not to points in it, as in, e.g., PSO [12], shuffled complex evolution [13], covariance matrix adaptation evolution strategy [14], controlled random search [15], differential evolution [16], and firefly optimization [7].

Convergent optimization via most-promising-area stochastic search (COMPASS) [17] sounds similar, but it is a technique for discrete optimization via simulation, where "there is no explicit form of the objective function, and function evaluations are stochastic and computationally expensive." Predictably, it always samples the most promising area next, rather than select it probabilistically using scores, and stochastic search refers to uniform sampling within that area.

The simplex method [18], Chapter 9, doesn't actually use simplexes. To create new points, controlled random search [15] generates random simplexes. These may overlap and are not guaranteed to cover the search space. Nor are they subdivided. DIRECT [19] uses hyperboxes that partition the search space. It avoids the 2^n complexity by directional search from their midpoints, ignoring their corners. Each hyperbox potentially containing the global minimum is trisected, rather than bisected, along each dimension in turn. There is no randomized selection.

Whereas [20] uses stochastic optimization to improve k-means clustering by avoiding local optima, the new SSB scheme applies k-means clustering to data generated by the core SSB algorithm to distinguish different optima and coordinate disparate data points corresponding to the same one.

III. The Particle Swarm Optimizer

The employed PSO optimizer follows [1], p 121, with a few modifications.

The swarm consists of a number, typically 20–40, of boids—a word play on bots, birds, and droids—each possessing a position $\mathbf{x}$ and a velocity $\mathbf{v}$. The objective function value $f(\mathbf{x})$ is also recorded. In each iteration, the position and velocity of each boid are updated as follows.

$$\begin{aligned} \mathbf{v} &\leftarrow \theta\mathbf{v} + c_1\mathbf{u}_1 \circ (\mathbf{x}_1^b - \mathbf{x}) + c_2\mathbf{u}_2 \circ (\mathbf{x}_2^b - \mathbf{x}) \\ \mathbf{x} &\leftarrow \mathbf{x} + \mathbf{v} \end{aligned}$$

Here $\mathbf{x}_1^b$ is the best point historically of the boid in question, while $\mathbf{x}_2^b$ is the best current point of all boids. [4] Thus, $\mathbf{x}_i^b - \mathbf{x}$ is the vector from the current point $\mathbf{x}$ to the best point $\mathbf{x}_i^b$. The idea is that the velocity $\mathbf{v}$ is attracted to both these two best points, with acceleration, or attraction coefficients, $c_i\mathbf{u}_i$. The first of these two terms of the velocity update is called the cognitive component, depending only on the history of the individual boid, and the second one is called the social component, depending on the swarm as a whole.

In [1], p 121, $\mathbf{u}_i$ is in fact a random scalar u_i, drawn uniformly from $[0, 1]$, and the multiplication is scalar multiplication of the difference vector. This guarantees that this acceleration term is along the line connecting $\mathbf{x}$ and $\mathbf{x}_i^b$. We instead use the Hadamard product, i.e., the component-wise product, denoted $\circ$, and let $\mathbf{u}_1$ and $\mathbf{u}_2$ be vectors where each element is independently uniform on $[0, 1]$. This is the method used in [21], p 219. The acceleration term is then in a cone or pyramid centered around $\mathbf{x}_i^b - \mathbf{x}$. We set both the two scalars c_1 and c_2 to 2, which is standard practice and which means that each component of $c_i\mathbf{u}_i$ has expectation 1.

The first term, $\theta\mathbf{v}$, where θ is called the inertia weight, is simply the previous velocity for $\theta = 1$. With $\theta > 1$, this term accelerates the boid in the direction it is currently travelling, preferring exploration over exploitation. With $\theta < 1$, it instead dampens the velocity, making it pay more heed to the cognitive and social acceleration components, thus preferring exploitation over exploration. It thus makes sense to start with an inertia weight above one, and then successively reduce it, ending the search with an inertia weight below one. To this end, we use dynamic inertia weighting to bring down θ geometrically from 1.4 in the first iteration to 0.3 in the last, cf. [1], pp 127–128.

As the velocities are initially self-accelerating, it is important to restrain them. Each velocity component is capped to have an absolute value that does not exceed a given maximum $v_{\max}$. This is another key parameter for balancing exploration and exploitation. The boids must stay within the feasibility domain, which is here a bounding hyperbox. Any boid that attempts to leave it has the offending $\mathbf{x}$ coordinate set to the boundary point. The corresponding $\mathbf{v}$ coordinate is set to the negative of its value. This leads to hard (elastic) reflection in the boundary, much like billiard balls.

IV. Experiments

A. Experimental Setup

We tested all *two-dimensional* objective functions of Figure 1 (penultimate page), most of which are from [22]. Function 0 comes from [5], Function 20 from [23], and Functions 21, 22, 24, and 25 are of our own design. All functions have unique global minima, except Function 0, due to symmetry in x and $x + y$, and Functions 12, 14, and 17, which have four global minima, due to symmetry in $\pm x, \pm y$. Functions 16 and 17 were corrected using [24].

We tested the optimizers on the three domains:

1) $[-80, 120] \times [-80, 120]$,
2) $[-800, 1200] \times [-800, 1200]$, and
3) $[-8000, 12000] \times [-8000, 12000]$.

These are larger than those of the test set, which are typically $[-10, 10] \times [-10, 10]$ or even $[-5, 5] \times [-5, 5]$. The domains

[4] Best here means "with the lowest objective function value."

were made asymmetric in x and y, since many test functions have their global minimum in $\mathbf{x} = \mathbf{0}$. In each trial, each optimizer was allowed $50\,000$ function evaluations. Success was defined as finding any argument with a function value within 10^{-13} of the known global minimal value.

The optimal value of $v_{\max}$ was determined for the PSO in each test domain. It used 20 boids and 2500 iterations to achieve the limit of $50\,000$ function evaluations. Fewer iterations result in failure to converge to within 10^{-13} of the minimum, whereas fewer boids fail to explore the search space effectively. Both SSB schemes used 40 epochs of 415 rounds each ($40 \times 415 \times 3 = 49\,800$), which deviates from the 60 epochs and 276 rounds ($60 \times 276 \times 3 = 49\,680$) used in [5]. This was done to accommodate the revised SSB scheme, where each epoch requires running the core SSB algorithm on a set of hyperboxes, one for each cluster, in addition to the hyperbox of the current epoch, see Section II-B, New Scheme.

B. Experimental Results

Table I shows their respective success rates in 1000 trials. We first and foremost note that the original SSB algorithm holds its own against the PSO algorithm and that the revised SSB scheme performs significantly better.

Ignoring ties, the revised SSB scheme is better than the PSO algorithm in 10 cases of 13 in the smallest domain and 10 of 14 cases in the other two domains. The pairwise sign test [25], which is not a very powerful test, yields p-values of 0.046 in the smallest domain, and 0.09 in the other two domains. Similarily, the revised SSB scheme is better than the original SSB scheme 18 times of 20 in the smallest domain and 15 times of 20 in two larger domains, yielding yields p-values of 0.0002 and 0.021, respectively. We used the sign test, rather than, say, the Wilcoxon signed-rank test [26], which is more powerful, since the objective functions vary greatly in difficulty. Thus, the difference pairs cannot be seen as drawn from the same distribution.

To further analyze these results, we need to consider the nature of each test objective function. These can be classified into:

- unimodal quadratic forms, Fcns 2, 6, 8;
- oligo-modal [5] polynomials, Fcns 3, 4, 5, 10, 18;
- multimodal damped trigonometrics, Fcns 0, 1, 12–17, 25;
- mixed trigonometrics and quadratics, Fcns 9, 11, 20, 24;
- other (unimodal) functions, Fcns 7, 21, 22.

The first group is simple there to see that the optimizers are sane; any deviation from a perfect score signals a fundamental problem. Functions 9, 10, 16, and 21 behave similarly. For these functions, the PSO and the new SSB perform flawlessly, whereas the old SSB fails in a small number of the trials. Inspection reveals that the reason for this is that the old SSB prematurely commits to a hyperbox that does not contain the global optimum. Subsequent epochs tend to repair this by increasing the hyperbox size, but this does not happen fast enough to allow convergence to within 10^{-13} of the optimum value. Conversely, this behavior pays dividends in the largest domain and for the hardest functions—0,11,and 14—where the old SSB algorithm prevails.

Function 7 is very hard, and defeats all optimizers. It has a concave valley, kinks, and a very anisotropic variable coupling. The gradient is ill-defined and unbounded in the valley bottom, and especially ill-behaved in the optimum. It is included as a reminder that our successes are relative. *Memento mori.*

Functions 1,11,14 are essentially constant at some distance to the optimum. This makes it harder for the exploration aspect of the algorithms to locate the target optimum. As the domain size increases, performance drops from full score, for at least some optimizer, to essentially total failure for all of them.

Functions 12, 14, and 17 have multiple global optima due to symmetries. This profits the old SSB scheme and leaves the PSO scheme in the dust. In conjunction with the very localized variation of function 14, this defeats the new SSB scheme, as it does not have enough rounds for exploration.

The only functions for which the old SSB scheme does better than the new one are Functions 0, 4, 12, 14, and 17. We have just discussed the latter three, which have multiple global optima due to symmetries. Function 0 has very many local optima and the only real remedy is exploration. Function 4 is interesting in that it is a case where the clustering fails to some extent. Better clustering methods raise the success rate from 0.435 to 0.864. This is however beyond the scope of the current article.

V. Summary and Conclusions

We evaluated the stochastic simplex bisection (SSB) algorithm against a particle swarm optimizer (PSO) on a prominent test set. The former employs a common stochastic optimization scheme, but unlike other stochastic approaches, it applies the scheme to search space regions, rather than to individual points. The latter is a well-known workhorse for stochastic optimization. This is the first evaluation of the SSB algorithm against a state-of-the-art global optimizer.

The original SSB scheme holds its own against the PSO. The revised SSB scheme is better at exploitation than the old one, allowing it to significantly outperform both the PSO and old SSB schemes in all three domains.

The key difference between the new and original SSB schemes is that the new one applies statistical pattern recognition to the data points sampled using the core SSB algorithm. This opens the door for a host of other schemes that view stochastic optimization not as a random walk, but as statistical inference. Current work includes using more sophisticated statistical pattern recognition techniques to identify regions that warrant further investigation.

The PSO algorithm has been in extensive use since 1995, and it comes with a large body of experience and know-how. The SSB algorithm was first published in 2015 and it is still very much evolving. It already outperforms the PSO algorithm. There is every reason to expect rapid progress in its performance.

[5] Oligo-modal means "with a few modes." (Apologies for mixing Greek and Latin roots...)

TABLE I
SUCCESS RATE IN 1000 TRIALS OF THE PSO THE SSB ALGORITHMS.

Domain	$[-80, 120]^2$			$[-800, 1200]^2$			$[-8000, 12000]^2$		
Fcn	PSO	Old SSB	New SSB	PSO	Old SSB	New SSB	PSO	Old SSB	New SSB
0	0.025	0.124	0.066	0.002	0.107	0.036	0.000	0.086	0.023
1	0.999	0.925	1.000	0.567	0.331	0.827	0.044	0.002	0.004
2	1.000	0.999	1.000	1.000	0.999	1.000	1.000	0.999	1.000
3	0.007	0.832	0.999	0.001	0.693	0.992	0.002	0.507	0.963
4	0.997	0.986	1.000	0.971	0.978	0.939	0.852	0.905	0.435
5	0.994	0.763	1.000	0.865	0.787	1.000	0.681	0.758	0.988
6	1.000	0.998	1.000	1.000	0.987	1.000	1.000	0.929	1.000
7	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
8	1.000	1.000	1.000	1.000	0.993	1.000	1.000	0.954	1.000
9	1.000	0.979	1.000	1.000	0.939	1.000	1.000	0.923	0.999
10	1.000	0.930	1.000	1.000	0.877	1.000	1.000	0.897	1.000
11	0.173	0.984	0.999	0.001	0.060	0.006	0.000	0.000	0.000
12	0.953	0.975	0.997	0.255	0.966	0.981	0.003	0.861	0.593
14	0.064	0.977	0.951	0.000	0.922	0.670	0.000	0.103	0.004
16	1.000	0.920	1.000	1.000	0.874	1.000	0.998	0.860	1.000
17	0.999	0.928	0.934	0.793	0.870	0.863	0.194	0.819	0.817
18	0.917	0.999	1.000	0.992	1.000	1.000	1.000	0.994	1.000
20	1.000	0.667	1.000	1.000	0.715	1.000	0.858	0.669	0.976
21	1.000	0.848	1.000	1.000	0.813	0.999	1.000	0.749	1.000
22	1.000	0.082	0.667	1.000	0.073	0.692	1.000	0.067	0.620
24	0.998	0.365	0.966	0.960	0.337	0.854	0.169	0.312	0.752
25	0.948	0.995	1.000	0.658	0.954	1.000	0.348	0.956	0.982
Ave	0.776	0.785	0.890	0.685	0.694	0.812	0.552	0.607	0.689

ACKNOWLEDGMENT

This work was funded by the BAULT network and the METALOGUE project. BAULT, building and using language technology, is a multidisciplinary research community at the University of Helsinki. METALOGUE is a Seventh Framework Programme collaborative project funded by the European Commission, grant agreement number 611073 (http://www.metalogue.eu).

REFERENCES

[1] M. Wahde, *Biologically Inspired Optimization Algorithms*. WIT Press, 2008.

[2] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine Learning*, vol. 3, no. 2-3, pp. 95–99, 1988. doi: 10.1023/A:1022602019183

[3] J. Kennedy and R. Eberhart, "Particle swarm optimization," 1995. doi: 10.1109/ICNN.1995.488968

[4] M. Dorigo, V. Maniezzo, and A. Colorni, "The ant system: Optimization by a colony of cooperating agents," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 26, no. 1, pp. 29–41, 1996. doi: 10.1109/3477.484436

[5] C. Samuelsson, "The stochastic simplex bisection algorithm," in *Procs. 15th Int.'l Conf. Computational Science*, ser. ICCS/15. Elsevier, 2015. doi: 10.1016/j.procs.2015.05.215 pp. 855–864. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915010236

[6] ——, "Comparative evaluation of the stochastic simplex bisection algorithm and the scipy.optimize module," in *Procs. 8th Workshop on Computational Optimization*, ser. WCO/15, 2015, pp. 573–578.

[7] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Procs. 5th Int.'l Conf. Stochastic Algorithms: Foundations and Applications*, ser. SAGA'09. Springer-Verlag, 2009. doi: 10.1007/978-3-642-04944-6_14. ISBN 3-642-04943-5, 978-3-642-04943-9 pp. 169–178. [Online]. Available: http://dl.acm.org/citation.cfm?id=1814087.1814105

[8] C. J. A. B. Filho, F. B. de Lima Neto, A. J. da C. C. Lins, A. I. S. Nascimento, and M. P. Lima, "A novel search algorithm based on fish school behavior," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Singapore, 12-15 October 2008*, 2008. doi: 10.1109/ICSMC.2008.4811695 pp. 2646–2651. [Online]. Available: http://dx.doi.org/10.1109/ICSMC.2008.4811695

[9] M. Haiman, "A simple and relatively efficient triangulation of the n-cube," *Discrete & Computational Geometry*, vol. 6, no. 1, pp. 287–289, 1991. doi: 10.1007/BF02574690. [Online]. Available: http://dx.doi.org/10.1007/BF02574690

[10] R. B. Hughes and M. R. Anderson, "Simplexity of the cube," *Discrete Mathematics*, vol. 158, no. 1âĂŞ3, pp. 99 – 150, 1996. doi: http://dx.doi.org/10.1016/0012-365X(95)00075-8. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0012365X95000758

[11] C. M. Bishop, *Machine Learning and Pattern Recognition*. Springer Verlag, 2006.

[12] A. I. Vaz and L. N. Vicente, "A particle swarm pattern search method for bound constrained global optimization," *J. of Global Optimization*, vol. 39, no. 2, pp. 197–219, Oct. 2007. doi: 10.1007/s10898-007-9133-5. [Online]. Available: http://dx.doi.org/10.1007/s10898-007-9133-5

[13] Q. Duan, S. Sorooshian, and V. Gupta, "Effective and efficient global optimization for conceptual rainfall-runoff models," *Water resources research*, vol. 28, no. 4, pp. 1015–1031, 1992. doi: 10.1029/91WR02985

[14] N. Hansen and S. Kern, "Evaluating the CMA evolution strategy on multimodal test functions," in *Parallel Problem Solving from Nature VIII*, X. Yao *et al.*, Eds. Springer, 2004. doi: 10.1007/978-3-540-30217-9_29 pp. 282–291.

[15] P. Kaelo and M. M. Ali, "Some variants of the controlled random search algorithm for global optimization," *J. Optim. Theory Appl*, vol. 130, no. 2, pp. 253–264, 2006. doi: 10.1007/s10957-006-9101-0

[16] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*, ser. Natural Computing. Springer-Verlag, 2006, iSBN 540209506.

Objective Functions

Fcns 21, 22, 24, 25 are novel; Fcn 0 from [5]; Fcn 20 from [23]; remainder from [22]. Fcns 16 and 17 corrected using [24].

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \prod_{k=1}^{n} f_0\left(\sum_{i=1}^{k} x_i\right) \quad \text{with} \tag{0}$$

$$f_0(x) = \sin(\sqrt{1+x^2})\cdot\cos(2x(x+1))\cdot\frac{\ln(1+x^2)}{\sqrt{1+x^2}}$$

$$f(\mathbf{x}) = 20\left(1-\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)\right)+e-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) \tag{1}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \sum_{i=1}^{n} x_i^2 \tag{2}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \sum_{i=1}^{n-1}\left(100\cdot(x_{i+1}-x_i^2)^2+(x_i-1)^2\right) \tag{3}$$

$$f(x,y) = (1.5-x+xy)^2+(2.25-x+xy^2)^2+(2.625-x+xy^3)^2 \tag{4}$$

$$f(x,y) = \left(1+(x+y+1)^2(19-14x+3x^2-14y+6xy+3y^2)\right)\cdot \tag{5}$$
$$\cdot\left(30+(2x-3y)^2(18-32x+12x^2+48y-36xy+27y^2)\right)$$

$$f(x,y) = (x+2y-7)^2+(2x+y-5)^2 \tag{6}$$

$$f(x,y) = 100\sqrt{|y-0.01x^2|}+0.01|x+10| \tag{7}$$

$$f(x,y) = 0.26(x^2+y^2)-0.48xy \tag{8}$$

$$f(x,y) = \sin^2(3\pi x)+(x-1)^2\left(1+\sin^2(3\pi y)\right)+(y-1)^2\left(1+\sin^2(2\pi x)\right) \tag{9}$$

$$f(x,y) = 2x^2-1.05x^4+\frac{x^6}{6}+xy+y^2 \tag{10}$$

$$f(x,y) = -\cos(x)\cos(y)\exp\left(-(x-\pi)^2-(y-\pi)^2\right) \tag{11}$$

$$f(x,y) = -0.0001\left(|\sin(x)\sin(y)|\exp\left(\left|100-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right)+1\right)^{0.1} \tag{12}$$

$$f(x,y) = -|\sin(x)\cos(y)|\exp\left(\left|1-\frac{\sqrt{x^2+y^2}}{\pi}\right|\right) \tag{14}$$

$$f(x,y) = 0.5+\frac{\sin^2(x^2-y^2)-0.5}{1+0.001(x^2+y^2)^2} \tag{16}$$

$$f(x,y) = 0.5+\frac{\cos^2(\sin(x^2-y^2))-0.5}{1+0.001(x^2+y^2)^2} \tag{17}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \frac{1}{2}\sum_{i=1}^{n}\left(x_i^4-16x_i^2+5x_i\right) \tag{18}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \sum_{i=1}^{n}\left(x_i^2+10\cdot(1-\cos(2\pi x_i))\right) \tag{20}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \sum_{i=1}^{n}|x_i| \tag{21}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \sum_{i=1}^{n}\sqrt{|x_i|} \tag{22}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \sum_{i=1}^{n}|x_i|+\left(10+x_i^2\right)\cdot(1-\cos(2\pi x_i)) \tag{24}$$

$$f(\mathbf{x}) = f(x_1,\ldots,x_n) = \sum_{i=1}^{n}1-\mathrm{sink}(x_i) \quad \text{with} \quad \mathrm{sink}(x) = \frac{\sin(x)}{x} \tag{25}$$

Fig. 1. Fig. 1 List of objective functions

[17] L. J. Hong and B. L. Nelson, "Discrete optimization via simulation using compass," *Operations Research*, vol. 54, no. 1, pp. 115–129, 2006.
[18] N. Andréassson, A. Egrafov, and M. Patriksson, *An Introduction to Continuous Optimization*. Studentlitteratur, 2005.
[19] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, no. 1, pp. 157–181, Oct. 1993. doi: 10.1007/BF00941892. [Online]. Available: http://dx.doi.org/10.1007/BF00941892
[20] R. Tang, S. Fong, X.-S. Yang, and S. Deb, "Integrating nature-inspired optimization algorithms to k-means clustering." in *ICDIM*, S. Fong, P. Pichappan, S. Mohammed, P. Hung, and S. Asghar, Eds. IEEE, 2012. ISBN 978-1-4673-2428-1 pp. 116–123.
[21] X.-S. Yang, *Nature-Inspired Optimization Algorithms*. Elsevier, 2014. ISBN 9780124167452, 9780124167438
[22] Wikipedia, "Test functions for optimization," http://en.wikipedia.org/wiki/Test_functions_for_optimization.html, 2014, accessed: 2014-12-14.
[23] K. Mullen, D. Ardia, D. Gil, D. Windover, and J. Cline, "DEoptim: An R package for global optimization by differential evolution," *J. of Stat. Software*, vol. 40, no. 6, pp. 1–26, 2011. [Online]. Available: http://www.jstatsoft.org/v40/i06/
[24] A. Gavana, "Global optimization benchmarks and AMPGO," http://infinity77.net/global_optimization, 2015, accessed: 2015-01-09.
[25] Wikipedia, "Sign test," https://en.wikipedia.org/wiki/Sign_test, 2017, accessed: 2017-06-19.
[26] ——, "Wilcoxon signed-rank test," https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test, 2017, accessed: 2017-06-19.