IEEE TRANSACTIONS ON

INDUSTRIAL INFORMATICS

A PUBLICATION OF THE IEEE INDUSTRIAL ELECTRONICS SOCIETY

NOVEMBER 2011 VOLUME 7 NUMBER 4 ITIICH (ISSN 1551-3203)

STATE-OF-THE-ART PAPERS

REGULAR PAPERS

SPECIAL SECTION ON INFORMATION TECHNOLOGY IN AUTOMATION

SPECIAL SECTION PAPERS

IEEE

Comparative Study of Derivative Free Optimization Algorithms

Nam Pham, *Student Member, IEEE*, A. Malinowski, *Senior Member, IEEE*, and T. Bartczak

***Abstract*—Derivative free optimization algorithms are often used when it is difficult to find function derivatives, or if finding such derivatives are time consuming. The Nelder Mead's simplex method is one of the most popular derivative free optimization algorithms in the fields of engineering, statistics, and sciences. This algorithm is favored and widely used because of its fast convergence and simplicity. The simplex method converges really well with small scale problems of some variables. However, it does not have much success with large-scale problems of multiple variables. This factor has reduced its popularity in optimization sciences significantly. Two solutions of quasi-gradients are introduced to improve it in terms of the convergence rate and the convergence speed. The improved algorithm with higher success rate and faster convergence which still maintains the simplicity is the key feature of this paper. This algorithm will be compared on several benchmark functions with other popular optimization algorithms such as the genetic algorithm, the differential evolution algorithm, the particle swarm algorithm, and the original simplex method. Then, the comparing results will be reported and discussed.**

***Index Terms*—Simplex method, genetic algorithm (GA), differential evolution algorithm (DE), particle swarm optimization (PSO), quasi-gradient method.**

I. Introduction

THE Nelder Mead's simplex method [1] is a popular derivative free optimization algorithm and is a method of choice for many practitioners. It is the prime choice algorithm in the Matlab optimization toolbox. It converges relatively fast and can be implemented relatively easily compared with other classical algorithms relying on gradients or evolutionary computations, etc. Unlike gradient methods [2], [3], the simplex method can optimize a function without calculating its derivatives, which usually require a lot of computing power and are expensive in high-dimensional problems as well. This property makes it more advantageous than others.

Although the simplex method is simple and robust in small scale optimization, it easily fails with large-scale optimization. In order to become a reliable optimization tool, it has to overcome this shortcoming by improving its convergence rate and convergence speed. This literature will give some new insights on why the simplex method may become inefficient in high dimensional optimization because of its lack of gradient information. This approach explains the low convergence rate without concerning its descent property when the objective function is uniformly convex presented in other literature [4], [5]. This paper will particularly present how to improve the simplex method by combining with two different quasi-gradient methods. The improved algorithm without complex mathematic computations can optimize multidimensional problems with higher success rate and faster convergence speed.

The genetic algorithm (GA) [6], the differential evolution algorithm (DE) [7], [8] and the particle swarm algorithm [9], etc., are the other popular derivative free optimization tools which are widely applied and familiar by researchers and practitioners [10]–[12]. These algorithms can perform well in both a global and local search and have the ability to find the optimum solution without getting trapped in local minima. This capability is mostly lacked by local search algorithms such as the calculus-based algorithms or the simplex method. The big issue of global search algorithms is the computational cost which often makes their convergence speed much slower than local search algorithms. The particle swarm optimization (PSO) is a kind of global search technique. It is a probabilistic technique which is different from the deterministic and stochastic techniques. Compared with the GA and the differential evolution algorithm, the PSO is simpler in term of computations because its crossover and mutation operation are done simultaneously while the crossover and mutation operation of the GA and the differential evolution algorithm are done between each pair in the whole population. With improvements contributed in this paper, the simplex method can be considered as another optional optimization algorithm which can work much more efficiently than other well-known derivative free optimization algorithms in many different fields of engineering or sciences.

This paper is organized as follows. Section II reviews several well-known derivative free optimization algorithms and the simplex algorithm. Section III presents the improved simplex method with quasi-gradient search. Section IV presents optimization functions, experimental results, and discussions. Section V is the conclusion.

II. Overview of Algorithms

A. Genetic Algorithm (GA)

The GA is invented to mimic the natural behavior of evolution according to the Darwin principle of survival and reproduction [13]. Unlike calculus-based methods, GA does not require derivatives, and it also has the ability to do a parallel search in the solution space simultaneously. Therefore, it is less likely

Manuscript received August 05, 2011; revised August 13, 2011, August 13, 2011, August 16, 2011; accepted August 20, 2011. Date of publication September 06, 2011; date of current version November 09, 2011. Paper no. TII-11-393.

N. Pham is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849-5201 USA (e-mail: nguyehu@auburn.edu).

A. Malinowski is with the Department of Electrical and Computer Engineering, Bradley University, Peoria, IL 61525 USA.

T. Bartczak is with the Department of Electronics and Telecommunications, University of Information Technology and Management, Rzeszów 23456, Poland .

Digital Object Identifier 10.1109/TII.2011.2166799

1551-3203/$26.00 © 2011 IEEE

to get trapped in local minima. Like the particle swarm algorithm and the differential evolution algorithm, GA starts by its initial population, and each individual is called a chromosome to represent a solution. During each generation, chromosomes will be evaluated according to their fitness values and evolved to create new chromosomes for the next generation. New childish chromosomes can be produced in two different ways either by emerging from two parental chromosomes in current generation with the crossover operator or by modifying chromosomes with the mutation operator. In order to maintain the population size, all chromosomes have to go through the natural selecting process. The chromosomes with better genes or better fitness will have higher probability to go to the next generation and other ones with worse genes is more likely to be rejected. This procedure is repeated until the best chromosome close to the optimum solution can be obtained. Another big advantage of GA is that it can be applied in different domains, not just only in optimization problems. However, it still has the limitation of premature convergence and low local convergence speed. Therefore, GA is usually improved by research scholars [14], [15].

B. Differential Evolution Algorithm (DE)

The differential evolution algorithm (DE) was introduced by Storn and Price in 1997 [7], [8]. Today it becomes one of the most robust function minimizers with relatively simple self-adapting mutation and is able to solve a wide range of optimization problems. The whole idea of DE is generating a new scheme to compute trial parameter vectors. These new parameter vectors are computed by adding the weighted difference between two population members to a third one. If the resulting vector has a lower objective function value than a predefined population member, the newly generated vector will replace the vector with which it was compared. Through time, this algorithm has been adapted to increase its efficiency. In 2007, a new concept of multiple trial vectors [16] was introduced into this algorithm. This approach aims to make DE able to converge for a broader range of problems because one scheme of calculating trial vectors may work well with certain type of problems but may not work with other ones. Another approach was proposed where the choice of learning strategies and the two control parameters F (weighing factor) and CR (crossover constant) are dynamically adjusted and also made a significant improvement [17]. Recently, an adaptive differential evolution algorithm with multiple trial vectors can train artificial neural networks successfully and shows its competitive results with the error back propagation algorithm and the Lavenberg Marquardt algorithm [18].

C. Particle Swarm Optimization (PSO)

The PSO is a concept that simulates the social swarm behavior of a flock of birds or a school of fish in searching for food [19]. The main concept is to utilize the intercommunication between each individual swarm with the best one to update its position and velocity. This algorithm operates on a randomly created population of potential solutions and searches for the optimum value by creating the successive population of solutions. PSO sounds similar to the differential evolution algorithm or the GA in term of its selecting strategy of the best child (or the best swarm), but it is really different. In this algorithm, the potential solutions so called swarm particles are moving to the actual (dynamically changing) optimum in the solution space. Each swarm has its own location, best location, velocity, and fitness. In each generation, each swarm will contact with the best swarm and follow him to update its own information. During its motion, if some swarms find better positions by comparing with their own fitness, they will automatically update themselves. In case there is a swarm finding the new best position, that swarm will be considered immediately as the current best. Because of its global search ability and fast convergence speed compared with other global search algorithms, PSO is applied widespread in optimization.

D. Nelder Mead's Simplex Algorithm

The simplex method [1] is a direct downhill search method. It is a simple algorithm to search for local minima and applicable for multidimensional optimization applications. Unlike classical gradient methods, this algorithm does not have to calculate derivatives. Instead it creates a geometric simplex and uses this simplex's movement to guide its convergence. A simplex is defined as a geometrical figure which is formed by $(n+1)$ vertices, where n is the number of variables of an optimization function, and vertices are points selected to form a simplex. In each iteration, the simplex method will calculate a reflected vertex of the worst vertex through a centroid vertex. According to the function value at this new vertex, the algorithm will do all kinds of operations as reflection or extension, contraction, or shrink to form a new simplex. In other words, the function values at each vertex will be evaluated iteratively, and the worst vertex with the highest function value will be replaced by a new vertex which has just been found. Otherwise, a simplex will be shrunk around the best vertex, and this process will be continued until a desired minimum is met. Moreover, the convergence speed of this algorithm can also be influenced by three parameters α, β, γ (α is the reflection coefficient to define how far a reflected point should be from a centroid point; β is the contraction coefficient to define how far a contracted point should be when it is contracted from the worst point and the reflected point in case the function value of the reflected point is smaller than the function value of the worst point; γ is the expansion coefficient to define how far to expand from the reflected point in case a simplex moves on the right direction). Depending on these coefficients α, β, γ, the volume of a simplex will be changed by the operations of reflection, contraction, or expansion respectively. The Nelder Mead's simplex method can be summarized as follows and more details can be found in the original paper [1].

- *Step 1*: get α, β, γ, select an initial simplex with random vertices $x_0, x_1, \ldots, x_n$ and calculate their function values.
- *Step 2*: sort the vertices $x_0, x_1, \ldots, x_n$ of the current simplex so that $f_0, f_1, \ldots, f_n$ in the ascending order.
- *Step 3*: calculate the reflected point x_r, f_r
- *Step 4*: if $f_r < f_0$:
  a) calculate the extended point x_e, f_e;
  b) if $f_e < f_0$, replace the worst point by the extended point $x_n = x_e, f_n = f_e$;
  c) if $f_e > f_0$, replace the worst point by the reflected point $x_n = x_r, f_n = f_r$.
- *Step 5*: if $f_r > f_0$:

a) if $f_r < f_i$, replace the worst point by the reflected point $x_n = x_r, f_n = f_r$.
b) if $f_r > f_i$:
   (b_1) if $f_r > f_n$: calculate the contracted point x_c, f_c;
      (c_1) if $f_c > f_n$ then shrink the simplex;
      (c_2) if $f_c < f_n$ then replace the worst point by the contracted point $x_n = x_c, f_n = f_c$;
   (b_2) if $f_r < f_n$: replace the worst point by the reflected point $x_n = x_r, f_n = f_r$.

- *Step 6*: if the stopping conditions are not satisfied, the algorithm will continue at *Step 2*.

Compared with gradient methods, the simplex method is simpler in term of mathematic computation, which is normally more complicated to calculate derivatives and requires more computing cost as well. Unlike the GA or the differential evolution algorithm, there is no operation of mutation or crossover in this algorithm. In each iteration, only one new vertex is computed; therefore, it converges much faster. These advantages are key features which motivate authors of this paper to improve the simplex algorithm and make it become a useful optimization tool for engineers, scientists, etc., in many different types of applications [20], [21].

III. Improved Simplex Method With Quasi-Gradients

Although the simplex method was proposed a long time ago (1965) [1] and has not had much success in optimizing large-scale problems [22], it is still a method of choice because of its simplicity. As a matter of fact, its necessary improvement of convergence speed and convergence rate is still an attractive research topic in the area of computing and optimization. For this purpose, many authors have proposed different ideas to address this issue. Gao and Han [23] proposed an implementation of the simplex method in which the expansion, contraction, and shrinking parameters depend on the dimension of optimization problems. Another author, Torczon [24], suggested that this poor convergence may be due to the search as direction becomes increasingly orthogonal to the steepest descent direction, etc. Without any satisfactory convergence theory, there is no doubt that the effect of dimensionality should be extended and investigated more. Clearly, this is one of the main reasons restricting its convergence capability. This paper is another effort to improve the simplex algorithm with two simple solutions, which are different from other explanations in the literature. Furthermore, the simplicity is also the main goal of authors to keep this algorithm robust and different from other optimization algorithms.

As presented shortly in the overview, the simplex algorithm converges based on the formation of the geometric simplex and its movement to find local minima. During optimization, this algorithm assumes that the direction to local minima can be found by the operations of reflection, contraction, and expansion without caring about the gradient. In other words, the dynamic change of a geometric simplex through these operations is utilized to approximate better vertices along the gradient direction. However, this assumption is not always true in reality, and that explains why the simplex algorithm fails easily with high dimensional optimization problems. Instead of calculating the reflected vertex as the original algorithm proposed, a new way is presented in the next two paragraphs by combining it with two different quasi-gradient methods respectively. These two quasi-gradient methods can be assumed as two approaches to approximate gradients by using numerical analysis rather than analytical analysis. With this modification, the improved algorithm converges much faster and more reliably than the original one.

To maintain the simplicity, two quasi-gradient methods are presented to approximate gradients [25]. The first method uses an extra vertex in a simplex. Its accuracy depends on the linearity of a function in the vicinity of a simplex. However, its computing cost does not increase significantly when the size of optimization problems becomes larger. The second method uses a hyperplane equation formed from a simplex. This method can estimate gradients more accurately; therefore, it converges faster. However, its high computing cost of inverse matrixes does not have much advantage with the large size of optimization problems.

A. Quasi-Gradient Method Using an Extra Vertex

This method approximates gradients of a $(n+1)$ dimensional plane created from a geometrical simplex. First, it selects an extra vertex composed from $(n+1)$ vertices in a simplex and then combines this vertex with other n selected vertices in the same simplex to estimate gradients. Its steps are presented as follows.

Assume an optimized function $f : \mathbb{R}^n \rightarrow \mathbb{R}, x \in \mathbb{R}^n$.

- *Step 1*: Initialize a simplex with $(n+1)$ random vertices $x_1, x_2, \ldots, x_{n+1}$.
- *Step 2*: Select an extra vertex E with its coordinates composed from n out of $(n+1)$ vertices in a simplex. In other words, its coordinate is a diagonal of the matrix X.

$$x_E = \operatorname{diag} \begin{bmatrix} x_{1,1}\, x_{1,2} \cdots x_{1,n-1}\, x_{1,n} \\ x_{2,1}\, x_{2,2} \cdots x_{2,n-1}\, x_{2,n} \\ \cdots \\ x_{n,1}\, x_{n,2} \cdots x_{n,n-1}\, x_{n,n} \end{bmatrix}_{nxn}$$
$$\text{or} \quad x_E = \lfloor x_{1,1}, x_{2,2}, \ldots, x_{n,n} \rfloor. \tag{1}$$

- *Step 3*: Approximate gradients based on the extra vertex E with n vertices in the selected simplex.

For i =1: n,
  If mod (i, 2) = 0
    $$g_i = \frac{\partial f}{\partial x_i} = \frac{f(i-1) - f(x_E)}{x_{i-1,i} - x_{E,i}} \tag{2}$$
  Else
    $$g_i = \frac{\partial f}{\partial x_i} = \frac{f(i+1) - f(x_E)}{x_{i+1,i} - x_{E,i}} \tag{3}$$
  End
End

- *Step 4*: Calculate the new reflected vertex R' based on the best vertex B and the approximate gradient G. Parameter σ is the learning constant or step size

$$x_{R'} = x_B - \sigma * G. \tag{4}$$

- *Step 5*: If the function value at R' is smaller than the function value at B, it means that BR' points to the same direction as the gradient then R' can be expanded to E'

$$x_{E'} = (1-\gamma)x_B + \gamma x_{R'}. \tag{5}$$

B. Quasi-Gradient Method Using a Hyperplane Equation

This quasi-gradient method forms a $(n+1)$-dimensional plane from $(n+1)$ vertices in a simplex and then uses matrix calculations to approximate gradients. This method can be described as follows.

Assume an optimized function $f : \mathrm{IR^n} \to \mathrm{IR}, x\mathrm{IR^n}$.

- *Step 1*: Initialize a simplex with $(n+1)$ random vertices $x_1, x_2, \ldots, x_{n+1}$.
- *Step 2*: A $(n+1)$-dimensional hyperplane formed from this simplex is assumed to have the approximate equation

$$V = a_0 + a_1x_1 + a_2x_2 + \cdots + a_{n-1}x_{n-1} + a_nx_n. \tag{6}$$

- *Step 3*: Substitute each vertex into the hyperplane equation, so there will be $(n+1)$ equations, as shown in (7) at the bottom of the page, where $V_i = f(x_{i,1}, x_{i,2}, \ldots, x_{i,n}), i = 1 : n+1$.
- *Step 4*: Calculate the approximate gradient matrix by writing the above multi-equations in the matrix formula $G = X^{-1} * V$

$$X^{-1} = \begin{bmatrix} 1 & 1 \cdots 1 \\ x_{1,1} & x_{2,1} \cdots x_{n+1,1} \\ x_{1,2} & x_{2,2} \cdots x_{n+1,2} \\ \cdots & \\ x_{1,n} & x_{2,n} \cdots x_{n+1,n} \end{bmatrix}^{-1}_{n+1\times n+1} \tag{8}$$

$$V = \begin{bmatrix} V_1 \\ V_2 \\ \cdots \\ V_{n+1} \end{bmatrix}_{n+1\times 1} \tag{9}$$

$$G = [a_0\ a_1\ a_2 \ldots a_{n-1}\ a_n]_{1\times n+1}, \frac{\partial V}{\partial x_i} = a_i. \tag{10}$$

- *Step 5*: Calculate the new reflected vertex R'

$$x_{R'} = x_B - \sigma * G. \tag{11}$$

- *Step 6*: Calculate the new expanded vertex E'

$$x_{E'} = (1-\gamma)x_B + \gamma x_{R'}. \tag{12}$$

The improved algorithm with quasi-gradient search is similar to the original simplex method except that it has to approximate gradients to search for its reflected vertex. In other words, its convergence will rely on the gradient direction through the new reflected vertex R' rather than the reflected vertex R calculated through the centroid vertex proposed by the original simplex algorithm. The improved simplex method with quasi-gradient search can be applied successfully in synthesizing lossy filters, and training artificial neural networks, etc. [26].

IV. Optimization Functions, Experimental Results and Discussions

All algorithms are tested on a set of functions with different levels of difficulty. These functions are well-known unconstrained optimization functions in literature. A large number of problems are relatively adequate to prove the reliability and robustness of these algorithms. It also warrants that the improved algorithm is much better than the original algorithm overall, not just better in a small set of problems. Algorithms are tested on a wide range not close to solutions to address on their convergence capability. Therefore, it is much more satisfactory when starting points are generated randomly. Unlike other publications in literature, we do not use the standard starting points for a certain function to test algorithms because it is fairly hard to measure their reliability and robustness or to differentiate between similar algorithms in this case. The use of initial points farther away from solutions frequently reveals dramatic differences of algorithms such as success rate, computing time, etc. These measurements will reflect the efficiency of each algorithm.

To evaluate the ability of these algorithms to solve problems, we measure their success rate and computing time with a list of benchmark functions from (1)–(16). These functions are typical ones usually used in testing local minimum optimization algorithms and have minima at "zero."

1) *De Jong function* [1], [27]–[29]

$$F_1(x) = \sum_{i=1}^{n} x_i^2$$

2) *De Jong function with moved axis* [30]

$$F_2(x) = \sum_{i=1}^{n} (x_i - a_i)^2$$

.

3) *Quadruple function* [31]

$$F_3(x) = \sum_{i=1}^{n} \left(\frac{x_i}{4}\right)^4.$$

4) *Powell function* [1], [28]

$$F_4(x) = \sum_{i=1}^{n} [\,(x_i + 10x_{i+1})^2 + 5(x_{i+2} - x_{i+3})^2 + (x_{i+1} - 2x_{i+2})^4 + 10(x_i - x_{i+3})^4].$$

$$\begin{aligned} V_1 &= a_0 + a_1x_{1,1} + a_2x_{1,2} + \cdots + a_{n-1}x_{1,n-1} + a_nx_{1,n} \\ V_2 &= a_0 + a_1x_{2,1} + a_2x_{2,2} + \cdots + a_{n-1}x_{2,n-1} + a_nx_{2,n} \\ &\cdots \\ V_{n+1} &= a_0 + a_1x_{n+1,1} + a_2x_{n+1,2} + \cdots + a_{n-1}x_{n+1,n-1} + a_nx_{n+1,n} \end{aligned} \tag{7}$$

5) *Moved axis Parallel hyper-ellipsoid function* [30]

$$F_5(x) = \sum_{i=1}^{n} i x_i^2.$$

6) *Zarakov function* [32]

$$F_6(x) = \sum_{i=1}^{n} \left[x_i^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^2 + \left(\sum_{i=1}^{n} 0.5 i x_i \right)^4 \right].$$

7) *Schwefel function* [32]

$$F_7(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2.$$

8) *Sum of different power function* [30]

$$F_8(x) = \sum_{i=1}^{n} |x_i|^{i+1}.$$

9) *Step function* [27]

$$F_9(x) = \sum_{i=1}^{n} |x_i + 0.5|^2.$$

10) *Box function* [29]

$$F_{10}(x) = \sum_{i=1}^{n} [e^{-ax_i} - e^{-ax_{i+1}} - x_{i+2}(e^{-a} - e^{-10a})]^2$$

where $a = t * i, t$ *is a const.*

11) *Rosenbrock function* [1], [27], [28], [33]

$$F_{11}(x) = \sum_{i=1}^{n} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2 \right].$$

12) *Biggs Exp6 function* [4], [28]

$$F_{12}(x) = \sum_{i=1}^{n} [x_{i+2} e^{t_i x_i} - x_{i+3} e^{-t_i x_{i+1}} + x_{i+5} e^{t_i x_{i+4}} - y_i]^2$$

where

$$t_i = 0.5i$$
$$y_i = e^{-t_i} - 5e^{-10t_i} + 3e^{-4t_i}.$$

13) *Kowalik and Osborne function* [27], [28], [33]

$$F_{13}(x) = \sum_{i=1}^{n} \left[a_i - \frac{x_i \left(b_i^2 + b_i x_{i+1} \right)}{b_i^2 + b_i x_{i+2} + x_{i+3}} \right]^2$$

where a, b *are vectors in* [27].

14) *Colville function* [27]

$$\begin{aligned} F_{14}(x) = \sum_{i=1}^{n} & [100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2 + (x_{i+2} - 1)^2 \\ & + 10.1((x_{i+1} - 1)^2 + (x_{i+3} - 1)^2) \\ & + 90(x_{i+2}^2 - x_{i+3})^2 + 19.8(x_{i+1} - 1)(x_{i+3} - 1)]. \end{aligned}$$

15) *Wood function* [4], [28], [33]

$$\begin{aligned} F_{15}(x) = \sum_{i=1}^{n} & [100 \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \\ & + 90 \left(x_{i+3} - x_{i+2}^2 \right)^2 + (1 - x_{i+2})^2 \\ & + 10.1((1 - x_{i+1})^2 + (1 - x_{i+3})^2) \\ & + 19.8(1 - x_{i+1})(1 - x_{i+3})]. \end{aligned}$$

16) *Bard function* [4], [28]

$$F_{16}(x) = \sum_{i=1}^{n} \left[y_i - x_i + \frac{u_i}{v_i x_{i+1} + w_i x_{i+2}} \right]^2$$

where y, u, v, w are vectors in [28].

With this significant improvement, the improved simplex method can be a useful optimization tool to replace for other popular algorithms as the GA or the particle swarm algorithm, etc. All evaluated algorithms are written in Matlab, and all experiments are tested on a PC with Intel Quad. In order to compare performances of these algorithms, some assumptions are set: algorithms start with a random initial variables in the range of $[-100, 100]$; dimension of all benchmark problems is 20; maximum iteration is equal to 100 000; desired error predefined to terminate algorithms is equal to 0.001; coefficients of the simplex method $\alpha = 1, \beta = 0.5, \gamma = 2$, learning constant $\sigma = 1$. In addition, the GA, the differential evolution algorithm, and the PSO each has 20 members in its population. These algorithms use the same default values as the ones written in the standard Matlab toolboxes. Because of timing cost to test the GA and the differential evolution algorithm, all results in Table I are the average values calculated over 25 random running times.

In this simulation, it is unnecessary to verify how the dimensionality affects the simplex algorithm. Therefore, one experiment with 20 dimensions is conducted, and only the algorithm using an extra vertex (SIM1) is selected to compare because of its simple computations relative to other derivative free optimization algorithms.

- *The Genetic Algorithm (GA)*: There are only two cases the GA can obtain 100% success rates. Although the GA cannot obtain high success rate as the improved simplex method, but it still shows its convergence ability in 5 out of 13 problems. However, this algorithm cannot converge as fast as the improved simplex method or the PSO does. There are 8 out of 13 problems, the GA cannot converge. Even the data is not displayed in Table I, but the GA shows its convergence trend in this experiment if a number of generations is increased. Among four derivative free optimization algorithms discussed here, the GA is the slowest one. This can be explained by its complex mutation and crossover operations.
- *The Differential Evolution Algorithm (DE)*: This algorithm does not converge really well with these optimization functions. There are five cases, it converges with high success rates. And there are two cases, it converges with low success rates. It cannot converge nearly half of problems. In order to optimize these functions, this algorithm needs more iterations. Therefore, it will take longer to solve the

TABLE I
EVALUATION OF AVERAGE ERROR, ERROR STANDARD DEVIATION, AND COMPUTING TIME OF 20-DIMENSIONAL FUNCTION

Optimization Function	GA		DE		PSO		SIM1	
	Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)
De Jong	44%	153.435	100%	111.45	96%	5.5137	100%	0.3450
De Jong with moved axis	60%	152.184	100%	122.30	Failure		100%	0.4450
Quadruple	100%	46.6543	80%	117.85	100%	0.0840	100%	0.2541
Powell	Failure		10%	173.76	100%	4.5735	100%	1.1455
Moved axis Parallel hyper-ellipsoid	Failure		85%	123.50	96%	9.7553	100%	0.7254
Zarakov	Failure		Failure		96%	5.2152	100%	1.1444
Schwefel	Failure		Failure		32%	89.9445	100%	0.9587
Sum of different power	100%	141.422	Failure		100%	0.06425	100%	1.7978
Step	35%	166.335	95%	121.09	80%	23.1161	100%	0.3992
Rosenbrock	Failure		Failure		Failure		54%	10.107
Biggs Exp6	Failure		10%	160.05	4%	126.276	27%	9.0644
Colville	Failure		Failure		Failure		40%	5.6746
Wood	Failure		Failure		Failure		44%	7.7333

TABLE II
EVALUATION OF SUCCESS RATE AND COMPUTING TIME OF TEN-DIMENSIONAL FUNCTIONS

Optimization Function	Nelder Mead's simplex algorithm		Improved simplex algorithm using an extra vertex		Improved simplex algorithm using a hyper plane equation	
	Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)
De Jong	100%	0.0907	100%	0.0806	100%	0.0474
De Jong with moved axis	100%	0.0932	100%	0.0824	100%	0.0477
Quadruple	100%	0.2183	100%	0.0539	100%	0.0350
Powell	100%	0.1037	100%	0.1346	100%	0.1444
Moved axis Parallel hyper-ellipsoid	100%	0.1102	100%	0.0841	100%	0.0648
Zarakov	99%	0.3195	100%	0.1879	100%	0.1766
Schwefel	100%	0.1666	100%	0.1400	100%	0.1420
Sum of different power	26%	1.1012	100%	0.1150	100%	0.1232
Step	100%	0.0863	100%	0.0825	100%	0.0475
Box	65%	0.3321	81%	0.3636	81%	0.5761
Rosenbrock	55%	0.8812	76%	1.2094	82%	1.5916
Biggs Exp6	52%	0.1118	60%	0.1244	20%	0.3509
Kowalik and Osborne	48%	0.2828	76%	0.4913	48%	5.0087
Colville	46%	0.2026	50%	0.2081	60%	0.2077
Wood	41%	0.2042	52%	0.2038	58%	0.2155
Bard	16%	0.7240	48%	0.9095	13%	2.2250

same problems. The DE converges faster than the GA, but it is much slower than the improved simplex algorithm.

- *The Particle Swarm Algorithm*: The particle swarm algorithm converges relatively fast; however, it does not have enough consistency. There are four cases, it fails to converge with no matter of a number of generations. There is one case that it converges with a really low success rate. Even it converges more than half of cases, but it is still relatively slower and has lower success rate than the improved simplex method. However, the particle swarm algorithm is much faster than the differential evolution algorithm and the GA in term of convergence speed.
- *The Improved Simplex Method*: From the experimental results in Table I, we can conclude that the improved simplex method converges much faster and more efficiently than the GA, the differential evolution algorithm, and the particle swarm algorithm in local minimum optimization. This is reflected through its higher success rate and less computing time for each testing function. It can get optimum solutions more than 75% of problems with 100% success rate. In other 25% of problems, its success rates are around 50%, but it is still much better than other algorithms. In term of computing time, this algorithm particularly outclasses the others. Its convergence speed is at least from ten to 100 times faster.

Obviously, the improved simplex algorithm with this significant modification can be an alternative tool to replace efficiently for other optimization tools. This algorithm is a direct search method which is free of derivative calculation. Therefore, it can converge much faster and higher success rate than algorithms based on evolutionary computations such as the GA, etc.

The next experiments are conducted with the same assumptions as the last one: algorithms start with a random initial simplex in the range of $[-100, 100]$; dimensions of all benchmark problems are equal to 10, 15, and 20, respectively; maximum iteration is equal to 100 000; target error predefined to terminate algorithms is equal to 0.001; coefficients

TABLE III
EVALUATION OF SUCCESS RATE AND COMPUTING TIME OF 15-DIMENSIONAL FUNCTIONS (NOTE: "-": NOT TESTED)

Optimization Function	Nelder Mead's simplex algorithm		Improved simplex algorithm using an extra vertex		Improved simplex algorithm using a hyper plane equation	
	Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)
De Jong	9%	0.2539	100%	0.2292	100%	0.0888
De Jong with moved axis	7%	1.0124	100%	0.2263	100%	0.0899
Quadruple	21%	0.9097	100%	0.1778	100%	0.0638
Powell	93%	1.5418	100%	0.4222	100%	0.3303
Moved axis Parallel hyper-ellipsoid	2%	0.6789	100%	0.2358	100%	0.1540
Zarakov	Failure		100%	0.5598	100%	0.5123
Schwefel	2%	1.1187	100%	0.3942	100%	0.3948
Sum of different power	Failure		100%	0.3303	100%	0.3300
Step	13%	0.4213	100%	0.1927	100%	0.0910
Box	Failure		19%	1.8882	4%	4.2749
Rosenbrock	Failure		55%	3.9881	80%	3.3474
Biggs Exp6	4%	0.7417	60%	1.4860	3%	2.5735
Kowalik and Osborne	-		-		-	
Colville	10%	1.9298	53%	0.5028	52%	0.5123
Wood	12%	2.3841	52%	0.5078	61%	0.5003
Bard	Failure	0.7240	11%	3.6462	Failure	2.2250

TABLE IV
EVALUATION OF SUCCESS RATE AND COMPUTING TIME OF 20-DIMENSIONAL FUNCTIONS

Optimization Function	Nelder Mead's simplex algorithm		Improved simplex algorithm using an extra vertex		Improved simplex algorithm using a hyper plane equation	
	Success rate	Comp. time (s)	Success rate	Comp. time (s)	Success rate	Comp. time (s)
De Jong	Failure		100%	0.4529	100%	0.1538
De Jong with moved axis	Failure		100%	0.4188	100%	0.1565
Quadruple	3%	1.7213	100%	0.3212	100%	0.1124
Powell	Failure		100%	0.8969	100%	0.6518
Moved axis Parallel hyper-ellipsoid	Failure		100%	0.4340	100%	0.3289
Zarakov	Failure		100%	1.3628	100%	1.2251
Schwefel	Failure		100%	0.8041	100%	0.9506
Sum of different power	Failure		100%	0.7094	100%	0.6802
Step	Failure		100%	0.3207	100%	0.1541
Box	Failure		5%	2.1718	3%	8.8745
Rosenbrock	Failure		54%	6.0026	75%	6.4713
Biggs Exp6	Failure		27%	3.2283	Failure	
Kowalik and Osborne	-		-		-	
Colville	Failure		40%	1.1572	44%	1.1259
Wood	Failure		44%	1.1708	50%	1.0224
Bard	-		-		-	0.1538

$\alpha = 1, \beta = 0.5, \gamma = 2$, learning constant $\sigma = 1$. All results in Tables II–IV are average values calculated over 100 random running times.

The comparisons between the simplex algorithm and its improved versions are summarized above. In these simulations all algorithms are still compared in terms of the success rate and computing time. While the success rate is used to describe their reliability, computing time reflects how fast these algorithms can converge. Three different sizes of functions 10, 15, and 20 are also tested, respectively, for the purpose of comparing their robustness, and it is also used to verify how the simplex method is affected by its dimensionality. From Tables II–IV, we can draw a conclusion that the improved algorithm shows its better performance than the original simplex method in terms of both success rate and computing time. The experimental results also tell that the improved simplex method, using a hyperplane equation, converges faster than the one using an extra vertex in most cases. It even requires more computations to approximate the gradient matrix. There are only two cases (*Box* function and *Biggs Exp6* function), the method of a hyperplane equation shows its worse results than the method of an extra vertex. When the problem size increases, the simplex method starts getting worse and is unable to converge. In Table II of ten-dimensional functions, the simplex algorithm converges relatively well although it does not have a high success rate in several cases. However, these numbers are still good enough and acceptable because of its fast convergence. When the size increases to 15 in Table III, its convergence rate suddenly drops down dramatically, and it totally fails in 20-dimensional problems or higher (Table IV); whereas, the improved algorithm still converges consistently well. It has 100% success in 9 out of 14 problems and over 40% success rate in 3 out of 14 problems.

There is only one case of *Box* function that the improved algorithm cannot obtain a good success rate. Even with 20-dimensional problems, this algorithm is still able to converge very fast when its minimum and maximum computing time is less than 1 (s) and 10 (s), respectively. Comparing these two algorithms, we can conclude that the improved algorithm using quasi-gradients can define its moving direction more precisely. That is the reason why the improved algorithm converges much better. With the same random choice of initial vertices, the improved simplex method usually gets a higher convergence rate and less computing time than the original simplex method. Even this algorithm is combined with the quasi-gradients, it does not face any difficulty to find function derivatives, and particularly its finding such derivatives is not time consuming as classical algorithms based on gradients.

V. Conclusion

Upon the comparative study of derivative free optimizations algorithms, it seems like the improved simplex method has more advantages than other algorithms based on evolutionary computations. Their comparison was summarized in Table I. The improved simplex algorithm with quasi-gradient search is presented with details in this paper. It is a derivative free optimization algorithm with two simple approaches of gradient search described. This algorithm can be used when it is difficult to find function derivatives, or if finding such derivatives are time consuming. This algorithm was tested over several benchmark problems of local minimum optimization, and shows its better performance than the original simplex method in terms of both convergence rate and computing time (Tables II–IV). Therefore, it shows a great deal of large-scale optimization problems and has been applied successfully in synthesizing filters and training neural networks. This algorithm also shows very promising results compared with other well-known evolutionary algorithms independent of gradients as the GA, the particle swarm algorithm, or the differential evolution algorithm, etc. it outperforms these algorithms with much higher success rate and at least ten to 100 times faster. The experiments tell that this algorithm is an effective alternative for other optimization algorithms. However, the modified algorithm presented in this paper can be improved by using other numerical techniques to calculate more accurate gradients. By using the analytical gradient instead of the quasi-gradient for some optimization problems, the improved simplex method can converge at least ten times faster. These types of improvements can be a good topic of future research.

References

[1] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Comput. J.*, vol. 7, pp. 308–313, 1965.

[2] K. Bredies, D. A. Lorenz, and P. Mass, "A generalized conditional gradient method and its connection to an iterative shrinkage method," *Comput. Optim. Appl.*, vol. 42, no. 2, pp. 173–193, 2009.

[3] K. Levenberg, "A method for the solution of certain problems in least squares," *Quart. Appl. Mach.*, vol. 2, pp. 164–168, 1944.

[4] A. Burmen, J. Puhan, and T. Tuma, "Grid restrained Nelder-Mead algorithm," *Comput. Optim. Appl.*, vol. 34, no. 3, pp. 359–375, 2006.

[5] C. T. Kelly, "Detection and remediation of stagnation in the Nelder-Mead algorithm using a sufficient decrease condition," *SIAM J. Opt.*, vol. 10, pp. 43–55, 2000.

[6] W. Lenwari, M. Sumner, and P. Zanchetta, "The use of genetic algorithms for the design of resonant compensators for active filter," *IEEE Trans. Ind. Electron.*, vol. 56, no. 8, pp. 2852–2861, Aug. 2009.

[7] R. Storn and K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, Dec. 1997.

[8] K. Price, "An Introduction to Differential Evolution," in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, Eds. London, U.K.: McGraw-Hill, 1999, pp. 79–108.

[9] B. Biswal, P. K. Dash, and B. K. Panigrahi, "Power quality disturbance classification using fuzzy C-means algorithm and adaptive particle swarm optimization," *IEEE Trans. Ind. Electron.*, vol. 56, no. 1, pp. 212–220, Jan. 2009.

[10] S.-H. Hur, R. Katebi, and A. Taylor, "Modeling and control of a plastic film manufacturing web process," *IEEE Trans. Ind. Inform.*, vol. 7, no. 2, pp. 171–178, May 2011.

[11] C. H. Lo, E. H. K. Fung, and Y. K. Wong, "Intelligent automatic fault detection for actuator failures in aircraft," *IEEE Trans. Ind. Inform.*, vol. 5, no. 1, pp. 50–55, Feb. 2009.

[12] F. Tao, D. Zhao, Y. Hu, and Z. Zhou, "Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system," *IEEE Trans. Ind. Inform.*, vol. 4, no. 4, pp. 315–327, Nov. 2008.

[13] P. Zanchetta, P. W. Wheeler, J. C. Clare, M. Bland, L. Empringham, and D. Katsis, "Control design of a three-phase matrix-converter-based AC-AC mobile utility power supply," *IEEE Trans. Ind. Electron.*, vol. 55, no. 1, pp. 209–217, 2008.

[14] K. Hangyu, J. Jing, and S. Yong, "Improving crossover and mutation for adaptive genetic algorithm," *Comput. Eng. Appl.*, vol. 12, pp. 93–96, 2006.

[15] L. H. Cheng and W. Y. Ping, "Genetic algorithm with a hybrid crossover operator and its convergence," *Comput. Eng. Appl.*, vol. 16, pp. 22–24, 2006.

[16] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, Apr. 2009.

[17] E. M. Montes, C. A. Coello, J. V. Ryes, and L. M. Davila, "Multiple trial vectors in differential evolution for engineering design," *Eng. Optim.*, vol. 39, no. 5, pp. 567–589, Jul. 2007.

[18] A. Slowik, "Application of an adaptive differential evolution algorithm with multiple trial vectors to artificial," *IEEE Trans. Ind. Electron.*, vol. 58, no. 8, pp. 3160–3167, Aug. 2011.

[19] S. Agrawal, Y. Dashora, M. K. Tiwari, and Y. J. Son, "Interactive particle swarm: A pareto-adaptive metaheuristic to multiobjective optimization," *IEEE Trans. System, Man, Cybern,*, vol. 38, no. 2, pp. 258–277, 2008.

[20] S. Wang, J. Watada, and W. Pedrycz, "Value-at-risk-based two-stage fuzzy facility location problems," *IEEE Trans. Ind. Inform.*, vol. 5, no. 4, pp. 465–482, Nov. 2009.

[21] G. Guo and Y. Shouyi, "Evolutionary parallel local search for function optimization," *IEEE Trans. Syst., Man, Cybern.*, vol. 33, no. 6, pp. 864–876, 2003.

[22] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright, "Convergence properties of the Nelder-Mead simplex method in low dimensions," *SIAM J. Opt.*, vol. 9, no. 1, pp. 112–147, 1998.

[23] F. Gao and L. Han, "Implementing the Nelder-Mead simplex algorithm with adaptive parameters," *Comput. Optim. Appl.*, May 4, 2010.

[24] Torczon, "V.: Multi-directional search: A direct search algorithm for parallel machines," Ph.D. dissertation, Rice Univ., Houston, TX, 1989.

[25] M. Manic and B. M. Wilamowski, "Random weights search in compressed neural networks using overdetermined pseudoinverse," in *Proc. IEEE Int. Symp. Ind. Electron. 2003*, 2003, vol. 2, pp. 678–683.

[26] N. Pham and B. M. Wilamowski, "Improved Nedler Mead's simplex method and applications," *J. Comput.*, vol. 3, no. 3, pp. 55–63, Mar. 2011, 2011.

[27] C. J. Chung and R. G. Reynolds, "Function optimization using evolutionary programming with self-adaptive cultural algorithms," in *Proc. SEAL'96 Selected Papers from the 1st Asia-Pacific Conf. Simulated Evol. Learning*.

[28] J. J. More, B. S. Garbow, and K. E. Hillstrom, "Testing unconstrained optimization software," *ACM Trans. Math. Softw.*, vol. 7, no. 1, pp. 136–140, 1981.

[29] J. T. Betts, "Solving the nonlinear least square problem: Application of a general method," *J. Opt. Theory Appl.*, vol. 18, no. 4, pp. 469–483, 1976.

[30] K. M. Bryden, D. A. Asklock, S. Corn, and S. J. Wilson, "Graph-based evolutionary algorithms," *IEEE Trans. Evol. Comput.*, vol. 10, no. 5, pp. 550–567, Oct. 2006.

[31] J. D. Hewlett, B. M. Wilamowski, and G. Dundar, "Optimization using a modified second-order approach with evolutionary enhancement," *IEEE Trans. Ind. Electron.*, vol. 55, no. 9, pp. 3374–3380, Sep. 2008.
[32] C. J. Price, I. D. Coope, and D. Byatt, "A convergent variant of the Nelder-Mead algorithm," *J. Opt. Theory Appl.*, vol. 11, no. 3, pp. 5–19, 2002.
[33] L. Nazareth and P. Tseng, "Gilding the Lily: A variant of the Nelder-Mead algorithm based on golden-section search," *Comput. Optim. Appl*, vol. 22, no. 1, pp. 133–144, 2002.

Aleksander Malinowski (M'93–SM'00) received the M.S. degree in electronics from the Gdansk University of Technology, Gdansk, Poland, in 1990 and the Ph.D. degree with highest honors in computer science and engineering (also received the Binford Memorial Award) from the University of Louisville, Louisville, KY, in 1996.

He is an Associate Professor at the Department of Electrical and Computer Engineering, Bradley University, Peoria, IL. Before he was visiting with the University of Wyoming, and also briefly taught at the Gdansk University of Technology. He is the author of six journal papers, six book chapters, oen solution manual, and 47 other refereed publications. The areas of his main interests include networked embedded systems, microcontroller Linux, autonomous mobile navigation, and computational intelligence.

Nam D. Pham (S'08) received the M.S. degree in electrical engineering from Auburn University, Auburn, AL. Currently, he is working towards the Ph.D. degree in electrical engineering at the Auburn University.

He is a Research Assistant with the Department of Electrical and Computer Engineering, Auburn University. His main interests include numerical optimization, neural networks, database systems, and network security.

Tomasz Bartczak received the M.Sc. degree from the Faculty of Electrical and Computer Engineering, University of Technology, Rzeszów, Poland, in 1998.

In 1998, he joined the Department of Electronics and Telecommunications, University of Information Technology and Management, Rzeszów, Poland. His teaching and research interests includes operating systems, computer networks, and architectural systems development. He currently works on the improvement of the computer network.