International Journal of Innovative Computing, Information and Control
Volume **8**, Number **2**, February **2012**

ICIC International ©2012 ISSN 1349-4198
pp. **1299–1312**

SOME APPLICATIONS OF S.T. (SHAFIABADY-TESHNEHLAB) EVOLUTIONARY OPTIMIZATION ALGORITHM

Niusha Shafiabady[1], Mohammad Teshnehlab[2] and Mohammad Ali Nekoui[2]

[1]Scientific Association of Mechatronics
Department of Mechatronics Engineering
Science and Research Branch
Islamic Azad University
P.O.Box 14515/755, Tehran 1477893855, Iran
nshafiabady@yahoo.com

[2]Department of Electrical Engineering
K.N.T University of Technology
470 Mirdamad Ave. West, P.O.Box 15875-4416, Tehran 1969764499, Iran
{teshnehlab; manekoui}@eetd.kntu.ac.ir

Received November 2010; revised March 2011

Abstract. *Considering that optimization methods firmly affect the error and the duration of solving a problem in science and engineering fields, applying a quick and efficient algorithm, suitable for the relevant problem is of much importance. This paper briefs a number of applications of a new evolutionary optimization algorithm, first presented by N. Shafiabady and M. Teshnehlab (referred hereinafter as "S.T." algorithm), which is efficient and quick in comparison with other popular conventional algorithms. The optimal solutions to a problem via S.T. algorithm, are found remarkably fast due to that the said algorithm is appropriate for online applications in compare with other conventional algorithms.*

Keywords: S.T. evolutionary optimization algorithm, Genetic algorithm, Particle swarm optimization

1. **Introduction.** Optimization algorithms have played important roles in vast areas of science, engineering and technology. The many different kinds of optimization algorithms have been categorized in various approaches. Evolutionary optimization algorithms – one of the above categories – are suitable for many applications. For example, aircraft traffic control has been modeled and handled by a constraint multi-objective optimization problem [10]. Evolutionary algorithms have also been widely used as an aid to neural networks and fuzzy systems in control applications [9,11,13].

PSO and GA are two of the most efficient and well-known optimization algorithms that numerous well-known optimization algorithms have been founded upon [1,12,14,15]. GA is an evolutionary optimization algorithm whereas PSO is a kind of swarm intelligence optimization algorithm. These two optimization algorithms have proven to be powerful tools applicable on nonlinear problems by which the optimal values and nonlinear parameters, which are quite complicated or even impossible to derivate via conventional methods, are derived. Since GA and PSO derive the solutions through large numbers of iterations, the long run-time makes it impossible for these algorithms to be applied on online applications whereas the optimization power of these algorithms is extensive and they are applicable to vast fields of science and engineering. The mentioned problem led the authors of this paper to a new evolutionary optimization algorithm which is capable of carrying out the optimization task as well as GA and PSO through much less number

of iterations. As indicated herein above some applications of S.T. evolutionary optimization algorithm, first initiated by N. Shafiabady, have been indicated and compared with common algorithms like PSO and GA. The results that have been derived using GA, PSO and S.T. through the problems that have been solved, show that S.T. has had a faster performance and has been able to do its task via less number of iterations in comparison with the other methods.

This paper is organized as follows. Section 2 represents S.T. algorithm, Section 3 shows the solved problems and the related simulation results and Section 4 represents the conclusion.

2. **S.T. Evolutionary Optimization Algorithm.** The initial idea of S.T. algorithm was originated from the classic perspective of propagation of sound waves through the medium of air molecules spreading randomly in atmosphere of the earth, although it is not its exact simulation. Consider a three dimensional closed system consisting of a sound producer and air as medium. The sinusoidal-spherical shaped sound waves are propagated in three dimensions of air medium when the molecules of the index element of air adjacent to the sound producer are vibrated respectively due to the vibration of the sound producer and the said element collides randomly with other layers of the medium. Henceforth the motion of the sound layers propagates and the energy of the sound producer is transmitted as a state wave function to all the directions in the system. This idea led to a new evolutionary optimization algorithm named S.T., although it is not its exact simulation. The structure of this algorithm is similar to the other evolutionary optimization algorithms. The pseudo code of S.T. algorithm is given below.

Algorithm 2.1. *Pseudo code of S.T. Algorithm*

- *Produce the initial population (called particles) with the initial velocities and positions.*
- *For* $i = 1$*: Maximum_iteration*
- $\{$*For* $j = 1$*: Number_of_population*
- $\{$*Calculate the FITNESS function's value*
- *Decide which particle has had the best performance*
- *Update the population's velocity and position* $\}$ $\}$

The initial population is produced giving each particle a weighting factor α. Figure 1 shows the initialization of the method. The particles have been produced with different positions randomly and each of the produced particles has a random velocity. The initial velocity of a sample particle is known as $v_{(t)}$ that is a combination of v_x and v_y.

If the acceleration of the particle (due to $\sum F$ exerted on it by the environment) during a time interval Δt is $a_{(t)}$, then the velocity function will be as mentioned in (1). This

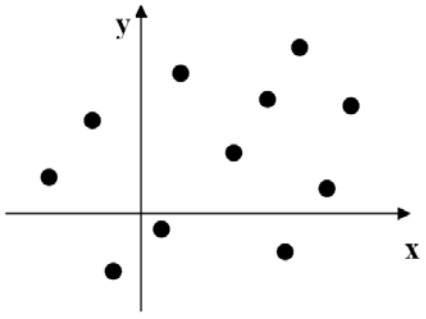

FIGURE 1. The initial particles with different positions

method can be symbolized by the following relations.

$$\begin{aligned} v_{ix}(t+1) &= \alpha_i v_{ix}(t) + a_{ix}\Delta t \\ a_{ix} &= random\_number \times \psi_{ix} \\ v_{iy}(t+1) &= \alpha_i v_{iy}(t) + a_{iy}\Delta t \\ a_{iy} &= random\_number \times \psi_{iy} \\ x_{ix}(t+1) &= x_{ix}(t) + v_{ix}(t+1) \times \beta\Delta t \\ x_{iy}(t+1) &= x_{iy}(t) + v_{iy}(t+1) \times \beta\Delta t \\ X_i &= real\left(\sqrt{x_{ix}(t+1) + x_{iy}(t+1)}\right) \end{aligned} \tag{1}$$

Here i denotes the particle number and there can be n genes in each of them denoting the number of the parameters of the target vector to be found. Each particle has a velocity in two directions that are shown by v_x and v_y. Then the new position of the particle is decided, again in two directions by x_x and x_y. The final new value of the particle is X that is decided according to its position in both directions. In the mentioned relations, $\beta\Delta t = 1$ and β is a constant value.

The weighting factor α_i that symbolizes the mass of each particle, which is a random uniform number that is produced only once, and represents the virtual weight given to each particle that remains unchanged during the run of the algorithm. Its proposed best bound is given in (2). This bound is found by 'try and error'.

$$-5 < \alpha_i < 5 \tag{2}$$

The other parameter is ψ that is another random uniform number applied to produce the random force. Its best bound is given in (3) that is found by 'try and error' respectively.

$$-25 < \psi < 25 \tag{3}$$

The last parameter is Δt that its best value is proposed to be $\Delta t = 0.1$ using 'try and error'. Here t symbolizes the time.

The parameters mentioned above are free from the optimization model and by changing them no improvements were shown in the results.

Like PSO algorithm, the calculated velocities are assumed to have a bound and if they exceed this bound that is $[-5, 5]$ in most of the problems, they have to be cut and put in the mentioned limits. This does not let our particles move wherever they wish without supervision. For most of the applications including the applications that have been elaborated in this article, the best value for this bound is $[-5, 5]$ that is achieved by 'try and error'. This bound is changeable according to the optimization problem. This works as a tuning device and if we want to let the particles move more freely it can be defined to have a bigger range.

It is to be mentioned that the X produced by $X = real\left(\sqrt{x_x(t+1) + x_y(t+1)}\right)$ can only be positive but the optimal values might be negative. In order to solve this problem this piece of code has to be used after determining X each time.

Algorithm 2.2. *Pseudo code for making the genes in each particle negative or positive*

- *Produce a Random_Number*
- *for* $Gen = 1$*: Max_Gen*
- $\{$*if Random_Number* > 0.5
- $\{a = -1;$
- *else*
- $a = 1\ \}$
- $X(Particle, Gen) = a * X(Particle, Gen)\}$

This makes the particles be able to cover the negative space too. This part can be omitted if the search space does not include negative numbers.

2.1. **Simulation results comparing S.T. with popular optimization algorithms.** To show the optimization ability of S.T. algorithm, it has been applied on some benchmark optimization problems and the results have been compared with GA and PSO. The simulations are done using MATLAB.

The benchmark optimization functions are shown in (4)-(7). The related figures of the test functions are shown in Figures 2-5.

Schwefel (f_1):

$$\begin{aligned} & f_1 = \sum_{i=1}^{n} \left[-x_i \sin\left(\sqrt{|x_i|}\right) \right] \\ & \vec{x} \in [-500, 500]^n \\ & \min f_1(\vec{x}^*) = f_1(420\vec{.}9687) = -418.9829n \end{aligned} \tag{4}$$

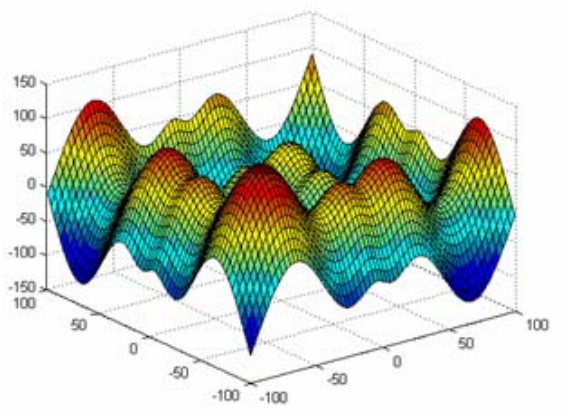

FIGURE 2. The shape of Schwefel function

Rastrigin (f_2):

$$\begin{aligned} & f_2 = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10) \\ & \vec{x} \in [-5.12, 5.12]^n \\ & \min f_2(\vec{x}^*) = f_2(\vec{0}) = 0 \end{aligned} \tag{5}$$

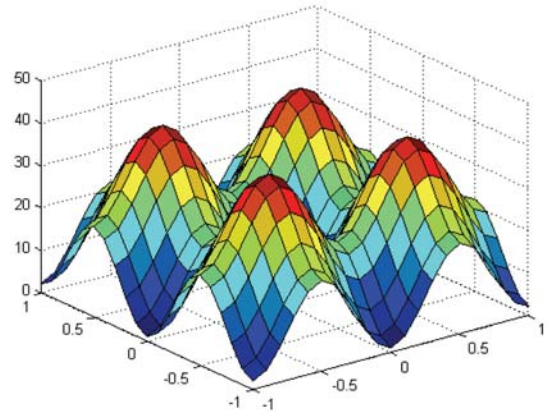

FIGURE 3. The shape of Rastrigin function

Rosenbrock (f_3):

$$\begin{aligned} & f_3 = \textstyle\sum_i^n (100(x_i + 1 - x_i^2)^2 + (x_i - 1)^2) \\ & \vec{x} \in [-2.048, 2.048]^n \\ & \min f_3(\vec{x}^*) \ = f_3(\vec{1}) = 0 \end{aligned} \tag{6}$$

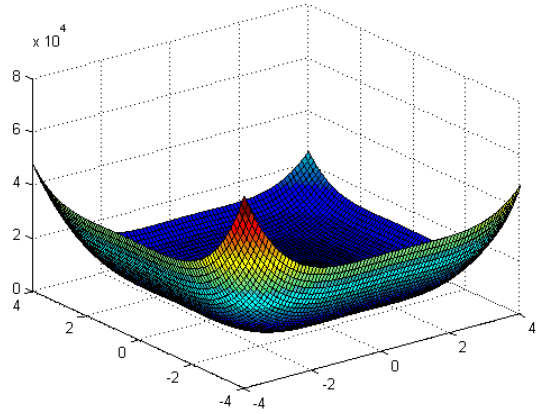

FIGURE 4. The shape of Rosenbrock function

Zakharov (f_4):

$$\begin{aligned} & f_4 = \textstyle\sum_i^n x_i^2 + (\sum_i^n 1/2ix_i)^2 + (\sum_i^n 1/2ix_i)^4 \\ & \vec{x} \in [-10, 10]^n \\ & \min f_4(\vec{x}^*) \ = f_4(\vec{0}) = 0 \end{aligned} \tag{7}$$

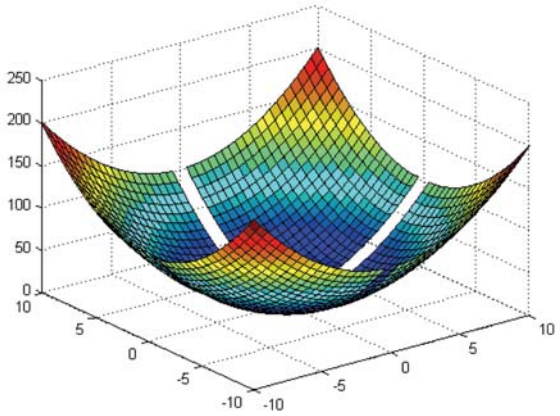

FIGURE 5. The shape of Zakharov function

The simulation results using two variables that represents $\vec{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}$ for the above-mentioned functions' minimum points are given in Table 1. This table shows the best, worst and the mean of the results in ten different runs for the three algorithms that have been compared with each other. Table 2 shows the number of iterations that are used for each run of the algorithms accordingly. The results show that S.T. has been able to derive the minimum point in apparently less number of iterations than the other methods and this shows that S.T. algorithm is clearly faster than the conventional methods.

In order to show the efficiency of the algorithm in higher search space dimensions the same functions have been simulated using ten variables and its related results are given

TABLE 1. The comparison of results of the algorithms' runs consisting of two variables

Test Function	**Best Results**			**Worst Results**			**Mean of the Results (10 Runs)**		
	GA	PSO	S.T.	GA	PSO	S.T.	GA	PSO	S.T.
Schwefel	−837.9658	−837.9658	−836.6151	−747.4487	−717.0748	−769.9438	−828.9139	−798.9427	−829.2813
Rastrigin	1.2403e-6	0	0	0.009	1.0100	0	0.0010	0.6000	0
Rosenbrock	0.7624	0.7624	0.7750	5.2256	0.7716	2.1232	1.8833	0.7634	1.4824
Zakharov	0.0202	1.2737e-8	0.0383	0.2263	0.0020	0.2284	0.1259	2.4139e-4	0.1060

TABLE 2. The comparison of the number of iterations of the algorithms' runs consisting of two variables

Test Function	**Number of Iterations for GA**	**Number of Iterations for PSO**	**Number of Iterations for S.T.**	Number of population
Schwefel	100	100	5	20
Rastrigin	100	100	2	20
Rosenbrock	100	100	10	20
Zakharov	100	100	5	20

TABLE 3. The comparison of results of the algorithms' runs consisting of ten variables

Test Function	**Best Results**			**Worst Results**			**Mean of the Results (10 Runs)**		
	GA	PSO	S.T.	GA	PSO	S.T.	GA	PSO	S.T.
Schwefel	−4.1898e3	−4.1411e3	−4.2636e3	−3.7161e3	−5.1143e3	−3.9153e3	−3.9767e3	−4.617e3	−4.2512e3
Rastrigin	1.6026e-4	3.3530	0	8.4816e-4	18.7894	0	3.5071e-4	11.3623	0
Rosenbrock	3.8120	10.5696	16.8615	6.0437	61.4309	38.0288	4.4816	34.9817	30.5677
Zakharov	0.3134	0.0581	0.4848	0.6007	0.4881	0.9731	0.4377	0.2300	0.7687

TABLE 4. The comparison of the number of iterations of the algorithms' runs consisting of ten variables

Test Function	**Number of Iterations for GA**	**Number of Iterations for PSO**	**Number of Iterations for S.T.**	Number of population
Schwefel	800	400	10	20
Rastrigin	400	600	10	20
Rosenbrock	700	800	300	20
Zakharov	100	100	10	20

in Table 3 for ten different runs. Table 4 shows the number of iterations that are applied to achieve the results. As the results show, S.T. is able to derive the minimum point in incomparably less number of iterations that is especially clear in the environments with higher dimensions in comparison with popular algorithms like GA and PSO. This represents S.T. as a fast optimization algorithm that can be used when computation time is an important issue.

As it was mentioned before the velocities' bounds work as a tuning device to control the velocities. For most of the applications including the applications that have been elaborated in this article, the best value for this bound is $[-5, 5]$ that is achieved by 'try and error'.

The part regarding generating negative numbers in Rosenbrock function has been deleted as it is not necessary.

3. **The Problems and Simulation Results.** Here a number of optimization problems are introduced and solved with GA, PSO and S.T. and their results have been compared with each other. The represented problems are goal programming, niching, TSP and multi objective optimization problems. All the programs are written using MATLAB software.

3.1. **Goal programming problem.** In order to compare S.T. with GA and PSO, these algorithms have been applied on another group of optimization problems that is goal programming problem. Goal programming is an approach to optimize one or a number of functions with the ability to define the desired priorities.

The general form of goal programming problem is given in (8) [3].

$$\begin{aligned} \text{Min} \quad & z_0 = \sum_{k=1}^{q} \sum_{i=1}^{m_0} P_k (w_{ki}^+ d_i^+ + w_{ki}^- d_i^-) \\ \text{s.t.} \quad & f_i(x) + d_i^- - d_i^+ = b_i \quad i = 1, 2, \ldots, m_0 \\ & g_i(x) \leq 0 \quad i = m_0 + 1, \ldots, m_1 \\ & h_i(x) = 0 \quad i = m_1 + 1, \ldots, m_2 \\ & d_i^-, d_i^+ \geq 0 \quad i = 1, 2, \ldots, m_0 \end{aligned} \tag{8}$$

Here P_k is the priority of the goals, d_i^-, d_i^+ are the deviations from the i^{th} goal, w_{ki}^-, w_{ki}^+ are the weights, f_i is the objective constraint that represents the main goal, g_i, h_i are equality and inequality constraints, b_i is the objective value and x is the result vector [3].

An example of the goal programming problem is shown in (9) [3].

$$\begin{aligned} \text{Min} \quad & \{d_3^+, 2d_1^- + d_2^+\} \\ \text{s.t.} \quad & x_1 x_2 + d_1^- - d_1^+ = 16 \\ & (x_1 - 3)^2 + x_2^2 + d_2^- - d_2^+ = 9 \\ & x_1 + x_2 + d_3^- - d_3^+ = 6 \end{aligned} \tag{9}$$

The three objective constraints have to be satisfied with respect to the mentioned priorities. The solutions to this problem using GA, PSO and S.T. are shown in Table 5. The last row of the table shows the desired value and as the given solutions show S.T.

TABLE 5. The comparison of different methods for goal programming problem

GA	Result vector $\vec{x} = [x_1 \;\; x_2]$	$\vec{x} = [3.1924 \;\; 3.1030]$
	Deviations from the 1st and the 2nd constraints $\{d_3^+, 2d_1^- + d_2^+\}$	$\{0, \; 30.4491\}$
	Number of iterations	300
PSO	Result vector $\vec{x} = [x_1 \;\; x_2]$	$\vec{x} = [3.0588 \;\; 3.0557]$
	Deviations from the 1st and the 2nd constraints $\{d_3^+, 2d_1^- + d_2^+\}$	$\{0, \; 30.3339\}$
	Number of iterations	300
S.T.	Result vector $\vec{x} = [x_1 \;\; x_2]$	$\vec{x} = \{3.026 \;\; 2.9708\}$
	Deviations from the 1st and the 2nd constraints $\{d_3^+, 2d_1^- + d_2^+\}$	$\{0, \; 32\}$
	Number of iterations	30
The Desired Value	**Result vector $\vec{x} = [x_1 \;\; x_2]$**	$\vec{x} = [\mathbf{3.01858 \;\; 2.98136}]$

has been able to achieve the goal better than GA and PSO in a friction of 0.1 numbers of iterations. The results represent that S.T. has been both fast and efficient.

3.2. **Niching problem.** Multi-solution problems are the problems with more than one optimum point. Niching is an algorithm for solving these kinds of problems. As niching points to the problems with multiple solutions as desired results, one of the parameters that are to be defined is the number of the optimal solutions or the number of niches that is q. Figure 6 shows the search space that is divided into five niches that are shown as $A_1,\ldots,A_5$ in the figure and five different niche radiuses that are labeled by $R_1,\ldots,R_5$ in the search space.

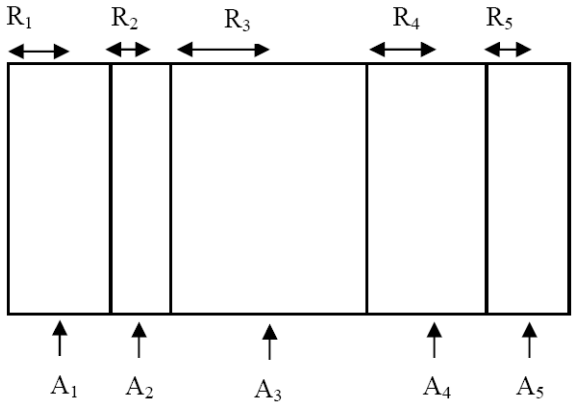

FIGURE 6. Different niches with different niche radiuses

The niche radius is defined in this way.

$$r = \frac{1}{2}\sqrt{\sum_{k=1}^{n}(x_{k,\max} - x_{k,\min})^2} \tag{10}$$

$$\rho = \frac{r}{\sqrt[n]{q}} \tag{11}$$

Here n denotes the problem's dimension. At first r is calculated and then ρ that is the niche radius is derived for each dimension. The niching algorithm using GA is implemented as mentioned below.

Algorithm 3.1. *Pseudo code of Niching Algorithm*

- *Produce every niche.*
- *For every niche* $[1,\ldots,q]$ *produce the next generation*:
  - *Choose the first parent as the best individual of the current niche.*
  - *Choose the second parent as the second best individual of the current niche.*
  - *Apply crossover and mutation.*
  - *Select niche_size of the population by grouping the population into q niches using DPS algorithm.*
  - *If the produced population is smaller than niche-size then produce the rest of the population randomly in the initial bounds.*

The Dynamic Peak Set (DPS) Algorithm is mentioned below and it is used for grouping the produced population [4].

Algorithm 3.2. *Pseudo code of DPS Algorithm*

- *Define ns = number_of_seen_peaks*

- *While all the population is not seen to*
  - *If chromosome(i) is within the radius of peak ns*
    - *Add this Chromosome to DPS(ns)*

The problems that are solved using GA, PSO and S.T. are four benchmark problems with five optimal solutions as mentioned here.

$$\begin{aligned} \text{Max} \quad & f_5(x) = \sin^6(5\pi x) \\ & x \in [0,1] \end{aligned} \tag{12}$$

The answer to this problem is the vector $\vec{x} = \begin{bmatrix} 0.1 & 0.3 & 0.5 & 0.7 & 0.9 \end{bmatrix}$.

Another problem is given in (13).

$$\begin{aligned} \text{Max} \quad & f_6(x) = e^{-2\log(2)\left(\frac{x-0.1}{0.8}\right)^2} \times \sin^6(5\pi x) \\ & x \in [0,1] \end{aligned} \tag{13}$$

The answer to this problem is the vector $\vec{x} = \begin{bmatrix} 0.08 & 0.25 & 0.45 & 0.68 & 0.93 \end{bmatrix}$.

The third problem is shown in (14).

$$\begin{aligned} \text{Max} \quad & f_7(x) = \sin^6\left(5\pi\left(x^{\frac{3}{4}} - 0.5\right)\right), \\ & x \in [0,1] \end{aligned} \tag{14}$$

The answer to this problem is the vector $\vec{x} = \begin{bmatrix} 0.1 & 0.3 & 0.5 & 0.7 & 0.9 \end{bmatrix}$.

The fourth problem is given in (15).

$$\begin{aligned} \text{Max} \quad & f_8(x) = e^{-2\log(2)\left(\frac{x-0.1}{0.8}\right)^2} \times \sin^6\left(5\pi\left(x^{\frac{3}{4}} - 0.5\right)\right), \\ & x \in [0,1] \end{aligned} \tag{15}$$

The answer to this problem is the vector $\vec{x} = \begin{bmatrix} 0.08 & 0.25 & 0.45 & 0.68 & 0.93 \end{bmatrix}$.

Table 6 shows the simulation results of the four test functions. The results show that all the algorithms specifically S.T. have derived the answer properly and S.T. has been able to achieve the results in distinctively less numbers of iterations.

In order to demonstrate the results completely, the related figures are also shown in the following Figures 7-14. Figure 7 shows the simulation result of the first function using GA and Figure 8 shows the same result using S.T.

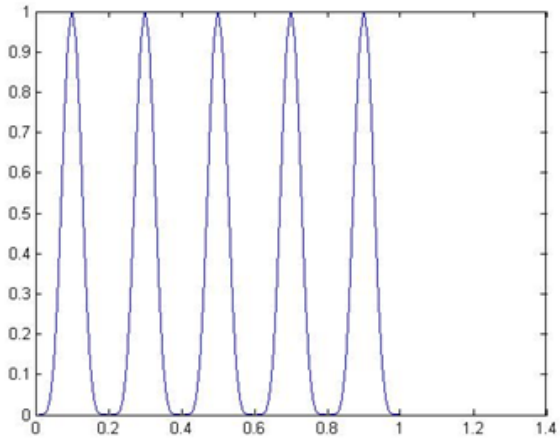

FIGURE 7. The result of $f_5(x)$ with GA

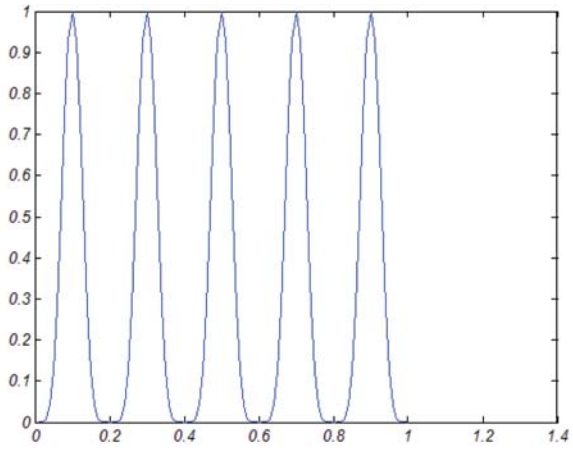

FIGURE 8. The result of $f_5(x)$ with S.T.

TABLE 6. The comparison of different methods for niching problem

$f_5(x)$	GA	Output vector	$\vec{x} = [0.1120 \;\; 0.2843 \;\; 0.4991 \;\; 0.6980 \;\; 0.8990]$
		Iterations	100
	S.T.	Output vector	$\vec{x} = [0.1024 \;\; 0.2982 \;\; 0.5058 \;\; 0.7009 \;\; 0.9015]$
		Iterations	3
	Desired output vector		$\vec{\boldsymbol{x}} = [\mathbf{0.1 \;\; 0.3 \;\; 0.5 \;\; 0.7 \;\; 0.9}]$
$f_6(x)$	GA	Output vector	$\vec{x} = [0.1104 \;\; 0.3015 \;\; 0.4948 \;\; 0.6968 \;\; 0.8977]$
		Iterations	100
	S.T.	Output vector	$\vec{x} = [0.1068 \;\; 0.3046 \;\; 0.5022 \;\; 0.6991 \;\; 0.9065]$
		Iterations	4
	Desired output vector		$\vec{\boldsymbol{x}} = [\mathbf{0.08 \;\; 0.25 \;\; 0.45 \;\; 0.68 \;\; 0.93}]$
$f_7(x)$	GA	Output vector	$\vec{x} = [0.0981 \;\; 0.2592 \;\; 0.4554 \;\; 0.6962 \;\; 0.9332]$
		Iterations	100
	S.T.	Output vector	$\vec{x} = [0.0792 \;\; 0.2423 \;\; 0.4474 \;\; 0.6877 \;\; 0.9371]$
		Iterations	6
	Desired output vector		$\vec{\boldsymbol{x}} = [\mathbf{0.1 \;\; 0.3 \;\; 0.5 \;\; 0.7 \;\; 0.9}]$
$f_8(x)$	GA	Output vector	$\vec{x} = [0.1047 \;\; 0.2443 \;\; 0.4525 \;\; 0.6680 \;\; 0.9371]$
		Iterations	100
	S.T.	Output vector	$\vec{x} = [0.0778 \;\; 0.2520 \;\; 0.4445 \;\; 0.6963 \;\; 0.9403]$
		Iterations	9
	Desired output vector		$\vec{\boldsymbol{x}} = [\mathbf{0.08 \;\; 0.25 \;\; 0.45 \;\; 0.68 \;\; 0.93}]$

The simulation results regarding the second problem using GA and S.T. are given in Figures 9 and 10.

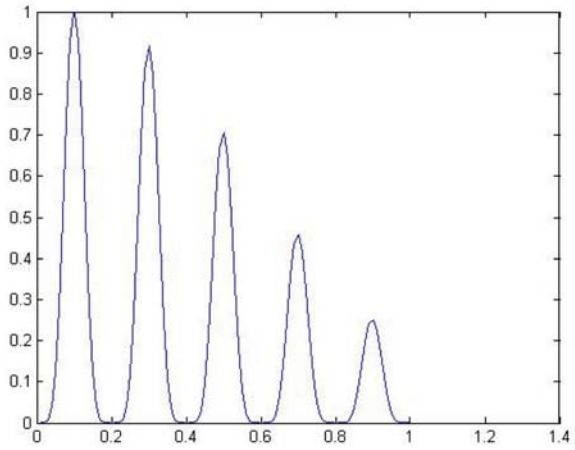

FIGURE 9. The result of $f_6(x)$ with GA

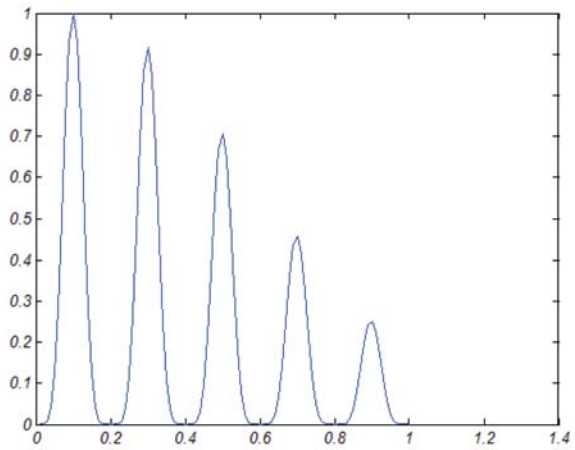

FIGURE 10. The result of $f_6(x)$ with S.T.

Figures 11 and 12 show the simulation results using GA and S.T. for the third problem.

Figures 13 and 14 show the simulation results using GA and S.T. for the fourth problem.

Finally, we can conclude that S.T. algorithm has been able to achieve the results in less number of iterations and with a good accuracy.

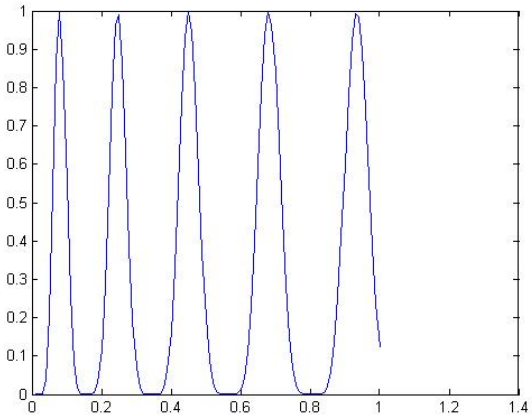

FIGURE 11. The result of $f_7(x)$ with GA

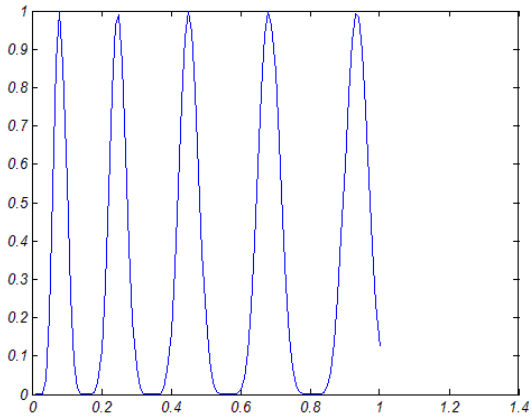

FIGURE 12. The result of $f_7(x)$ with S.T.

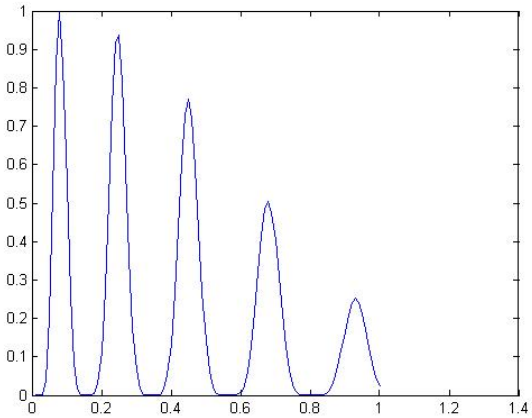

FIGURE 13. The result of $f_8(x)$ with GA

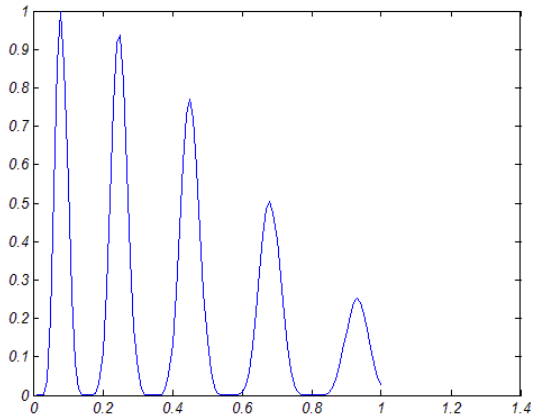

FIGURE 14. The result of $f_8(x)$ with S.T.

3.3. **Traveler's salesman problem.** Traveler's Salesman Problem (TSP) is an example of discrete form of optimization problems. TSP is an NP-complete problem and has been solved by different varieties of optimization algorithms and its aim is to find a minimum cost Hamiltonian cycle in a complete undirected graph [5-8]. Hamiltonian cycle is a cycle that traverses all the nodes with the minimum cost.

In order to compare GA, PSO and S.T. in solving a discrete optimization problem with each other, an example of TSP problem has been solved with all algorithms and the results have been compared with each other. The complete undirected graph is shown in Figure 15.

Table 7 shows that all the algorithms have achieved the Hamiltonian cycle that traverses the nodes (1,3,4,2,1). Twenty population members have been used for all the algorithms. As it is seen here, S.T. has been able to perform well in less numbers of iterations.

The demonstrated results show that S.T. has been able to derive the solution in a friction of 0.03 numbers of iterations in compare with GA and PSO for solving this discrete problem.

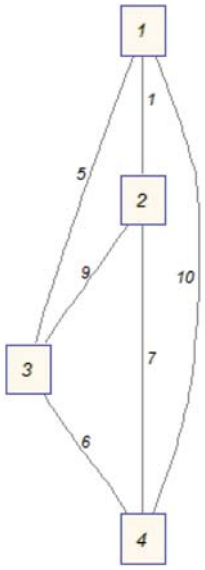

FIGURE 15. An example of a graph for solving TSP problem

TABLE 7. The comparison of different methods for solving TSP problem

GA			PSO			S.T.		
Minimum Cost	**Hamiltonian Cycle**	**Iterations**	**Minimum Cost**	**Hamiltonian Cycle**	**Iterations**	**Minimum Cost**	**Hamiltonian Cycle**	**Iterations**
18	(1,3,4,2,1)	100	18	(1,3,4,2,1)	100	18	(1,3,4,2,1)	3

3.4. **Multi-objective problem.** Multi-objective optimization problems are the problems with more than one goal and these goals are in a way contradictory with each other. This means that one cannot add all the goals and extract a single goal from them. This is a challenge in optimal decision making that can be mentioned as one of the human's concerns in everyday life.

The general form of multi-objective optimization problems is shown in (16).

$$\begin{aligned} \text{Max} \quad & \{z_1 = f_1(\vec{x}),\ z_2 = f_2(\vec{x}), \ldots,\ z_q = f_q(\vec{x})\} \\ \text{s.t.} \quad & g_i(\vec{x}) \leq b_i \quad i = 1, \ldots, m \end{aligned} \tag{16}$$

Here q objectives with contradictions are present and with m arbitrary constraints [3]. The aim is finding a pareto solution that is a set of the best solution that best satisfies all the conditions and constraints in compare with other sets of solutions.

One of the test beds of these kinds of optimization problems is the problem mentioned in (17) [3].

$$\begin{aligned} \text{Min} \quad & f_1 = x^2, \quad f_2 = (x-2)^2 \\ \text{s.t.} \quad & x \in R \end{aligned} \tag{17}$$

GA, PSO and S.T. have been applied on this problem to find the pareto solution. The simulations are done using 200 iterations for all the algorithms. For GA the crossover rate is assumed to be 0.3 and the mutation rate is assumed to be 0.1. For S.T. algorithm the velocities' bound is assumed to be $[-4, 4]$ and this is achieved by 'try and error'. The three algorithms' pareto solutions are shown in Figure 16 that is the same as the results given in [3]. This figure shows that all three algorithms have been able to find the pareto solution efficiently all with identical performances.

Now in order to compare the diversity of the solutions, the resulted pareto solution by each algorithm is shown in a separate figure respectively. Figure 17 shows the pareto

solution found by GA, Figure 18 shows the solutions found by PSO and Figure 19 shows the pareto solution found by S.T. If all the solutions are drawn together in one figure the resulting pareto solutions will be overlapped by the others exactly as demonstrated in Figure 16.

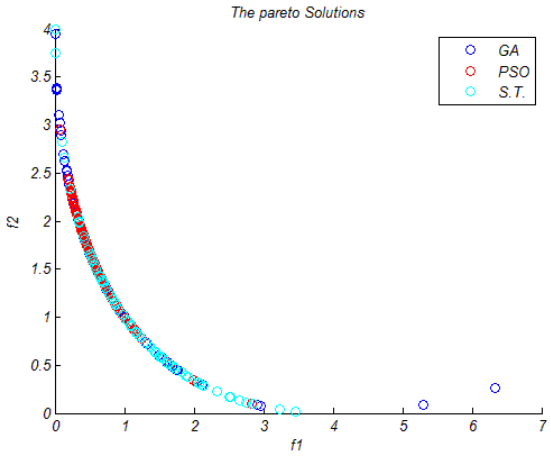

FIGURE 16. The pareto solution for three optimization algorithms

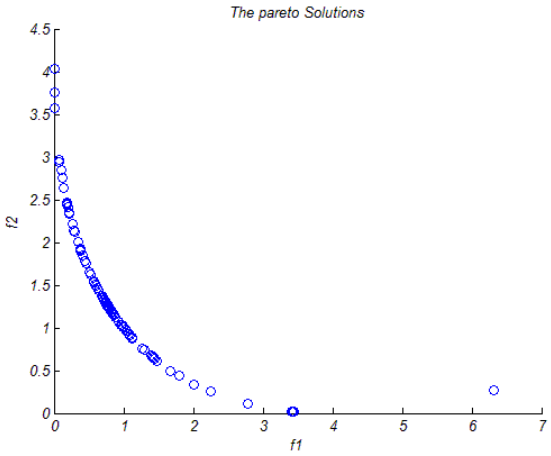

FIGURE 17. The pareto solution for GA

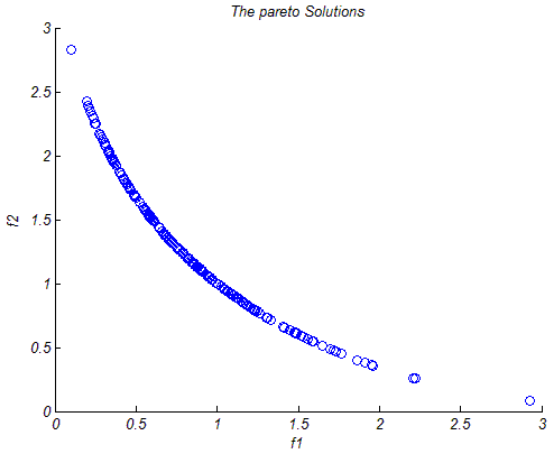

FIGURE 18. The pareto solution for PSO

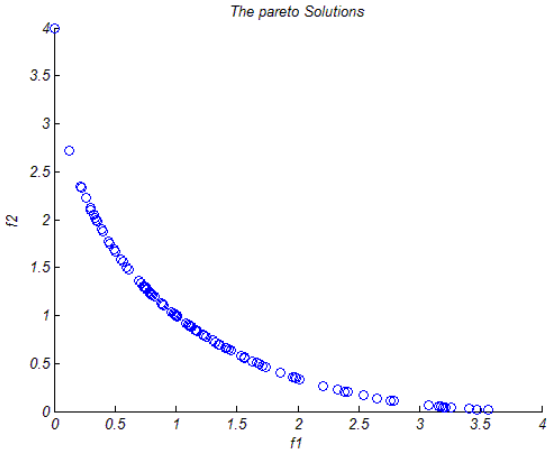

FIGURE 19. The pareto solution for S.T.

These results show that all the algorithms have worked efficiently and as it can be observed by the results, S.T. algorithm has been able to find the pareto solution as well as the other algorithms. Regarding the diversity of the represented solutions, it can be seen that S.T. has been able to derive the results with better diversity than the other methods, considering the results shown in Figures 17-19, so it can be proposed as a good tool for solving multi-objective problems.

4. **Conclusion.** As the simulation results of the presented problems show, S.T. has been able to have better or the same performance identical with powerful and popular algorithms like GA and PSO in much less numbers of iterations. This property has been

demonstrated in niching, goal programming and TSP problems. The mentioned property enables S.T. to be used for data transfer in communication networks and best path's signification in metropolitan travels. The results that have been achieved in multi-objective optimization problem show that S.T. is able to derive the solutions with better diversity. It has also been shown that if the dimension of the problem is increased as in the presented benchmark optimization functions, S.T. will be able to derive the solutions in distinctively less numbers of iterations. This allows S.T. to be able to perform the optimization task even for online tasks that is a good achievement as one of the deficiencies of most of the evolutionary optimization algorithms is their long run time. It has shown that S.T. is an optimization algorithm that is efficient and fast and can be used for different optimization and engineering applications.

REFERENCES

[1] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons, 2005.
[2] Y. Shi and R. C. Eberhart, Fuzzy adaptive particle swarm pptimization, *Proc. of the Congress on Evolutionary Computation*, Seoul, South Korea, pp.101-106, 2001.
[3] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, John Wiley & Sons, 2000.
[4] O. M. Shir and T. Back, Dynamic niching in evolution strategies with covariance matrix adaptation, *Proc. of the IEEE Congress on Evolutionary Computation*, pp.2584-2591, 2005.
[5] L. Wang, A. A. Maciejewsi, H. J. Siegel, V. P. Roychowdhury and B. D. Eldridge, A study of five parallel approaches to a genetic algorithm for the traveling salesman problem, *Intelligent Automation and Soft Computing*, vol.11, no.4, pp.217-234, 2005.
[6] M. Barth, *Approximation of the Traveling Salesman Problem Utilizing a Genetic Algorithm in a Parallel System*, 2009.
[7] M. Bhattacharyya and A. K. Bandyopadhyay, Comparative study of some solution algorithms for traveling salesman problem using genetic algorithms, *Cybernetics and Systems*, vol.40, no.6, pp.490-507, 2009.
[8] X. Shi, Y. Liang, H. Lee, C. Lu and Q. Wang, Particle swarm optimization-based algorithms for TSP and generalized TSP, *Information Processing Letters*, vol.103, pp.169-176, 2007.
[9] S.-F. Lin and Y.-C. Cheng, Two-strategy reinforcement evolutionary algorithm using data-mining based crossover strategy with TSK-type fuzzy controllers, *International Journal of Innovative Computing, Information and Control*, vol.6, no.9, pp.3863-3885, 2010.
[10] Y. Guo, X. Cao and J. Zhang, Constraint handling based multiobjective evolutionary algorithm for aircraft landing scheduling, *International Journal of Innovative Computing, Information and Control*, vol.5, no.8, pp.2229-2238, 2009.
[11] G.-R. Yu and L.-W. Huang, Design of LMI-based fuzzy controller for robot arm using quantum evolutionary algorithms, *ICIC Express Letters*, vol.4, no.3(A), pp.719-724, 2010.
[12] C. Liu, An evolutionary algorithm for solving dynamic nonlinear constrained optimization, *ICIC Express Letters*, vol.4, no.3(B), pp.1039-1044, 2010.
[13] X. Wang, Y. Cheng and W. Sun, A proposal of adaptive PID controller, *China University Mining & Technology*, vol.17, no.1, pp.40-44, 2007.
[14] M. A. Montes de Oca, T. Stutzle, M. Birattari and M. Dorigo, Frankenstein's PSO: A composite particle swarm optimization algorithm, *IEEE Transactions on Evolutionary Computation*, vol.13, no.5, pp.1120-1132, 2009.
[15] N. Noman and H. Iba, Accelerating differential evolution using an adaptive local search, *IEEE Transactions on Evolutionary Computation*, vol.12, no.1, pp.107-125, 2008.