Proceedings of the 35th Chinese Control Conference
July 27-29, 2016, Chengdu, China

A matlab toolbox for continuous state transition algorithm

Xiaojun Zhou, Chunhua Yang, Weihua Gui
School of Information Science and Engineering, Cental South University, Changsha 410083, P. R. China
E-mail: michael.x.zhou@csu.edu.cn

Abstract: State transition algorithm (STA) has been emerging as a novel stochastic method for global optimization in recent few years. To make better understanding of continuous STA, a matlab toolbox for continuous STA has been developed. Firstly, the basic principles of continuous STA are briefly described. Then, a matlab implementation of the standard continuous STA is explained, with several instances given to show how to use to the matlab toolbox to minimize an optimization problem with bound constraints. In the same while, a link is provided to download the matlab toolbox via available resources.

Key Words: State transition algorithm, matlab toolbox, global optimization, continuous optimization

1 Introduction

State transition algorithm (STA) [1–7] has been emerging as a novel stochastic method for global optimization in recent few years, and it has found applications in nonlinear system identification, image segmentation, task assignment, optimized controller design, water distribution networks, energy conservation optimization, optimized controller design, signal processing, etc [8–15]. In state transition algorithm, a solution to an optimization problem is considered as a state, and an update of a solution can be regarded as a state transition. Unlike other population-based stochastic optimization techniques, such as genetic algorithm, particle swarm optimization, differential evolution, etc, the basic state transition algorithm is an individual-based optimization method. Based on an incumbent best solution, a neighborhood with special property will be formed automatically when using certain state transformation operator. A variety of state transformation operators, for example, rotation, translation, expansion, and axesion in continuous STA, or swap, shift, symmetry and substitute in discrete STA, are designed purposely for both global and local search. On the basis of the neighborhood, then, a sampling technique is used to generate a candidate set, and the next best solution is updated by using a selection technique based on previous best solution and the candidate set. This process is repeated until some terminal conditions are satisfied.

To make better understanding of continuous STA, a matlab toolbox for continuous STA for global optimization problem with bound constraints has been developed. In this paper, we will describe the standard continuous STA and its corresponding matlab toolbox in detail. The remainder of this paper is organized as follows. In Section 2, the basic principles of continuous STA are given. Section 3 illustrates how to use the matlab toolbox for continuous STA. Finally, conclusion is drawn in Section 4.

2 The basic principles of continuous STA

In this paper, we focus on the continuous STA for the following global optimization problem

$$\min_{\boldsymbol{x}\in\Omega} f(\boldsymbol{x}) \tag{1}$$

This work is supported by National Natural Science Foundation (NNSF) of China (61503416, 61533020, 61590921) and the Key Exploration Project (7131253).

where $\boldsymbol{x} = [x_1, \cdots, x_n] \in \mathbb{R}^n$, $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n | l_i \leq x_i \leq u_i, i = 1, \cdots, n\} \subseteq \mathbb{R}^n$ is a closed and compact set, which is usually composed of lower and upper bounds of $\boldsymbol{x}$.

By referring to state space representation, on the basis of current state $\boldsymbol{x}_k$, the unified form of generation of a new state $\boldsymbol{x}_{k+1}$ in state transition algorithm can be described as follows:

$$\begin{cases} \boldsymbol{x}_{k+1} = A_k \boldsymbol{x}_k + B_k \boldsymbol{u}_k \\ y_{k+1} = f(\boldsymbol{x}_{k+1}) \end{cases}, \tag{2}$$

where $\boldsymbol{x}_k = [x_1, x_2, \cdots, x_n]^T$ stands for a state, corresponding to a solution of an optimization problem; $\boldsymbol{u}_k$ is a function of $\boldsymbol{x}_k$ and historical states; A_k and B_k are state transition matrices, which are usually some state transformation operators; $f(\cdot)$ is the objective function or fitness function, and y_{k+1} is the function value at $\boldsymbol{x}_{k+1}$.

2.1 State transition operators

Using state space transformation for reference, four special state transition operators are designed to generate continuous solutions for an optimization problem.

(1) Rotation transformation

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha \frac{1}{n\|\boldsymbol{x}_k\|_2} R_r \boldsymbol{x}_k, \tag{3}$$

where α is a positive constant, called the rotation factor; $R_r \in \mathbb{R}^{n\times n}$, is a random matrix with its entries being uniformly distributed random variables defined on the interval [-1, 1], and $\|\cdot\|_2$ is the 2-norm of a vector. This rotation transformation has the function of searching in a hypersphere with the maximal radius α.

(2) Translation transformation

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \beta R_t \frac{\boldsymbol{x}_k - \boldsymbol{x}_{k-1}}{\|\boldsymbol{x}_k - \boldsymbol{x}_{k-1}\|_2}, \tag{4}$$

where β is a positive constant, called the translation factor; $R_t \in \mathbb{R}$ is a uniformly distributed random variable defined on the interval [0,1]. The translation transformation has the function of searching along a line from x_{k-1} to x_k at the starting point x_k with the maximum length β.

(3) Expansion transformation

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \gamma R_e \boldsymbol{x}_k, \tag{5}$$

where γ is a positive constant, called the expansion factor; $R_e \in \mathbb{R}^{n\times n}$ is a random diagonal matrix with its entries

obeying the Gaussian distribution. The expansion transformation has the function of expanding the entries in $\boldsymbol{x}_k$ to the range of [-∞, +∞], searching in the whole space.
(4) Axesion transformation

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \delta R_a \boldsymbol{x}_k \tag{6}$$

where δ is a positive constant, called the axesion factor; $R_a \in \mathbb{R}^{n\times n}$ is a random diagonal matrix with its entries obeying the Gaussian distribution and only one random position having nonzero value. The axesion transformation is aiming to search along the axes, strengthening single dimensional search.

2.2 Regular neighborhood and sampling

For a given solution, a candidate solution is generated by using one of the aforementioned state transition operators. Since the state transition matrix in each state transformation is random, the generated candidate solution is not unique. Based on the same given point, it is not difficult to imagine that a regular neighborhood will be automatically formed when using certain state transition operator. In theory, the number of candidate solutions in the neighborhood is infinity; as a result, it is impractical to enumerate all possible candidate solutions.

Since the entries in state transition matrix obey certain stochastic distribution, for any given solution, the new candidate becomes a random vector and its corresponding solution (the value of a random vector) can be regarded as a sample. Considering that any two random state transition matrices in each state transformation are independent, several times of state transformation (called the degree of search enforcement, *SE* for short) based on the same given solution are performed for certain state transition operator, consisting of *SE* samples. It is not difficult to find that all of the *SE* samples are independent, and they are representatives of the neighborhood. Taking the rotation transformation for example, a total number of *SE* samples are generated in pseudocode as follows

for $i \leftarrow 1$, *SE* **do**
  $\mathrm{State}(:, i) \leftarrow \mathrm{Best} + \alpha \frac{1}{n\|\mathrm{Best}\|_2} R_r \mathrm{Best}$
end for

where $Best$ is the incumbent best solution, and *SE* samples are stored in the matrix State.

2.3 An update strategy

As mentioned above, based on the incumbent best solution, a total number of *SE* candidate solutions are generated. A new best solution is selected from the candidate set by virtue of the fitness function, denoted as *newBest*. Then, an update strategy based on greedy criterion is used to update the incumbent best as shown below

$$Best = \begin{cases} \mathrm{newBest}, & \text{if } f(\mathrm{newBest}) < f(\mathrm{Best}) \\ \mathrm{Best}, & \text{otherwise} \end{cases} \tag{7}$$

2.4 Algorithm procedure of the basic continuous STA

With the state transformation operators, sampling technique and update strategy, the basic state transition algorithm can be described by the following pseudocode

repeat
  if $\alpha < \alpha_{\min}$ **then**
    $\alpha \leftarrow \alpha_{\max}$
  end if
  Best $\leftarrow$ expansion(funfcn,Best,SE,β,γ)
  Best $\leftarrow$ rotation(funfcn,Best,SE,α,β)
  Best $\leftarrow$ axesion(funfcn,Best,SE,β,δ)
  $\alpha \leftarrow \frac{\alpha}{fc}$
until the specified termination criterion is met

As for detailed explanations, rotation$(\cdot)$ in above pseudocode is given for illustration purposes as follows

oldBest $\leftarrow$ Best
fBest $\leftarrow$ feval(funfcn,oldBest)
State $\leftarrow$ op_rotate(Best,SE,α)
[newBest,fnewBest] $\leftarrow$ fitness(funfcn,State)
if fnewBest $<$ fBest **then**
  fBest $\leftarrow$ fnewBest
  Best $\leftarrow$ newBest
  State $\leftarrow$ op_translate(oldBest,newBest,SE,β)
  [newBest,fnewBest] $\leftarrow$ fitness(funfcn,State)
  if fnewBest $<$ fBest **then**
    fBest $\leftarrow$ fnewBest
    Best $\leftarrow$ newBest
  end if
end if

As shown in the above pseudocodes, the rotation factor α is decreasing periodically from a maximum value $\alpha_{\max}$ to a minimum value $\alpha_{\min}$ in an exponential way with base *fc*, which is called lessening coefficient. op_rotate$(\cdot)$ and op_translate$(\cdot)$ represent the implementations of proposed sampling technique for rotation and translation operators, respectively, and fitness$(\cdot)$ represents the implementation of selecting the new best solution from *SE* samples. It should be noted that the translation operator is only executed when a solution better than the incumbent best solution can be found in the *SE* samples from rotation, expansion or axesion transformation. In the basic continuous STA, the parameter settings are given as follows: $\alpha_{\max} = 1, \alpha_{\min} = 1e$-4, $\beta = 1, \gamma = 1, \delta = 1$, $SE = 30, fc = 2$.

When using the fitness$(\cdot)$ function, solutions in *State* are projected into Ω by using the following formula

$$x_i = \begin{cases} u_i, & \text{if } x_i > u_i \\ l_i, & \text{if } x_i < l_i \\ x_i, & \text{otherwise} \end{cases} \tag{8}$$

where u_i and l_i are the upper and lower bounds of x_i respectively.

3 A matlab implementation of the continuous STA

3.1 Installation

The STA toolbox was developed under matlab 7.11.0 (R2010b), which can be download via the following link `http://www.mathworks.com/matlabcentral/fileexchange/52498-state-transition-algorithm`.

After unzipping the file "basic_STA.zip", you will get the following file list:

There exist a file folder named "sta" (it contains the core files of the STA toolbox), and several .m files as well as a text document.

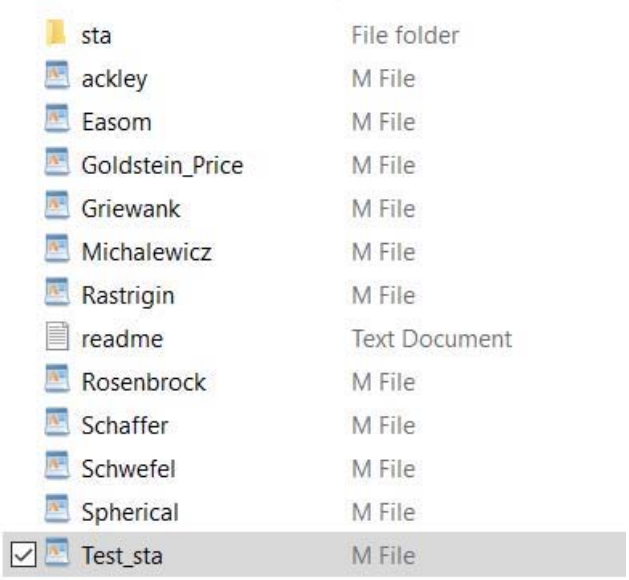

Fig. 1: The unzipped basic_STA file list

3.2 The main file

By running the main file "Test_sta.m", which is shown as below

```
clear all
clc
currentFolder = pwd;
addpath(genpath(currentFolder))
% parameter setting
warning('off')
SE =  30; % degree of search enforcement
Dim = 10;% dimension
Range = repmat([-5.12;5.12],1,Dim);
Iterations = 1e3;
tic
[Best,fBest,history] = STA(@Rastrigin,SE,
Dim,Range,Iterations);
toc
Best
fBest
semilogy(history)
xlabel('Iterations'),ylabel('Fitness(log)')
```

you can get the following similar results

If you want to optimize another benchmark function (the file list as shown in Fig. 1), for example, the Griewank function with 15 dimension, you just need to do the following changes

```
Dim = 15;% dimension
Range = repmat([-600;600],1,Dim);%range
[Best,fBest,history] = STA(@Griewank,SE,
Dim,Range,Iterations);
```

After that, by running the main file, you can get the following similar results as shown in Fig. 3.

```
Elapsed time is 0.686311 seconds.

Best =

  1.0e-008 *

  Columns 1 through 5

   -0.6730    0.0080    0.0613   -0.2359   -0.0114

  Columns 6 through 10

    0.1098    0.1003    0.5083    0.0490   -0.5944


fBest =

     0
```

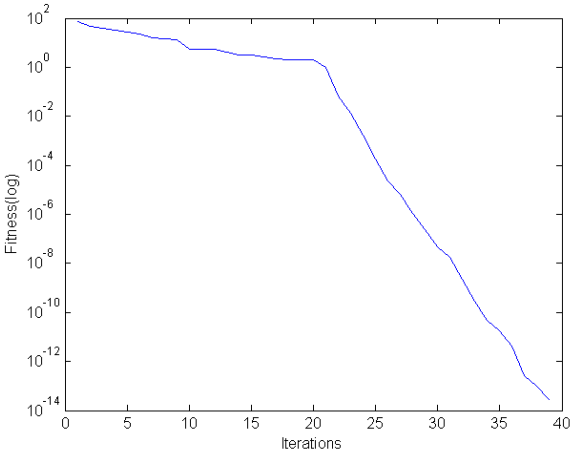

Fig. 2: The results for the Rastrigin function (10D)

If you want to optimize a user-defined function, for example, the following minimization problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^3} \ (x_1-1)^2+(x_2-2x_1)^2+(x_3-3x_2)^2$$
$$\text{s.t.} \ \ -3\le x_1\le 3, -2\le x_2\le 2, -1\le x_3\le 1$$

Firstly, you have to create a m-file (for example with a name 'myfun') as follows

```
function y = myfun(x)
x1=x(:,1);
x2=x(:,2);
x3=x(:,3);
y=(x1-1).^2+(x2 - 2*x1).^2+(x3-3*x2).^2;
```

Then, you need to do the following changes of the main file "Test_sta.m"

```
Dim = 3;% dimension
Range = [-3 -2 -1;3 2 1];%range
Iterations = 1e1;
tic
[Best,fBest,history] = STA(@myfun,SE,
Dim,Range,Iterations);
```

```
Elapsed time is 0.941310 seconds.

Best =

  1.0e-007 *

  Columns 1 through 5

    0.0309    0.0005    0.0000   -0.0290   -0.0193

  Columns 6 through 10

   -0.0002   -0.0203   -0.0105    0.0240   -0.0092

  Columns 11 through 15

   -0.0371   -0.0004    0.3285   -0.1284    0.0172


fBest =

     0
```

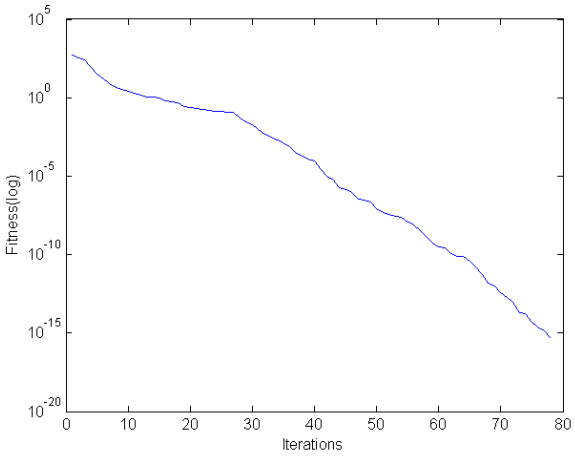

Fig. 3: The results for the Griewank function (15D)

```
toc
Best
fBest
```

After that, by running the main file, you can get the following similar results as shown in Fig. 4.

3.3 The core files of STA

The core files of STA are contained in the file folder "sta", in which, the file list can be seen in Fig. 5.

By opening the file STA.m, the main algorithm procedure of the basic continuous STA is shown as below

```
function [Best,fBest,history]=STA(funfcn,
SE,Dim,Range,Iterations)
% parameter setting
alpha_max = 1;
alpha_min = 1e-4;
```

```
Elapsed time is 0.558033 seconds.

Best =

    0.3478    0.3696    1.0000


fBest =

    0.5435
```

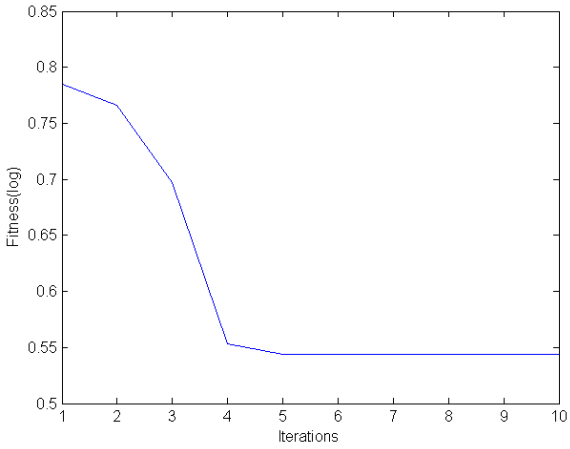

Fig. 4: The results for the user-defined function (3D)

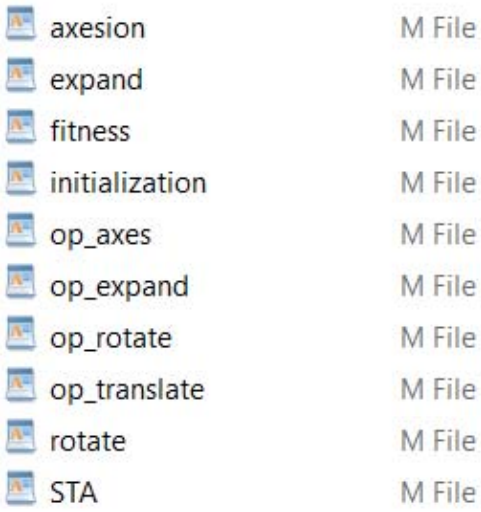

Fig. 5: The file list in the file folder "sta"

```
alpha = alpha_max;
beta = 1;
gamma = 1;
delta = 1;
fc = 2;
% initialization
State = initialization(SE,Dim,Range);
[Best,fBest] = fitness(funfcn,State);
```

```
% iterative process
for iter = 1:Iterations
    if alpha < alpha_min
        alpha = alpha_max;
    end
    [Best,fBest] = expand(funfcn,Best,
    SE,Range,beta,gamma);
    [Best,fBest] = rotate(funfcn,Best,
    SE,Range,alpha,beta);
    [Best,fBest] = axesion(funfcn,Best,
    SE,Range,beta,delta);
    history(iter) = fBest;
    alpha = alpha/fc;
end
```

The function *initialization* is to generate *SE* uniformly random initial points within the range (lower and upper bounds), and *fitness* is to select the best candidate among them using greedy criterion. They are shown in matlab codes as below

```
function  State=initialization(SE,Dim,
Range)
Pop_Lb = Range(1,:);
Pop_Ub = Range(2,:);
State  = rand(SE,Dim).*repmat
(Pop_Ub-Pop_Lb,SE,1)+repmat(Pop_Lb,SE,1);

function [Best,fBest] = fitness(funfcn,
State) % calculate fitness
fState = feval(funfcn,State);
[fGBest, g] = min(fState);
fBest = fGBest;
Best = State(g,:);
```

The functions *op_rotate*, *op_translate*, *op_expand* and *op_axes* are the implementation of rotation, translation, expansion and axesion transformation respectively. The corresponding matlab codes are listed as below

```
function y=op_rotate(Best,SE,alpha)
%rotation transformation
n = length(Best);
y = repmat(Best',1,SE) +
alpha*(1/n/(norm(Best)+eps))*
reshape(unifrnd(-1,1,SE*n,n)*Best',n,SE);
y = y';

function y=op_translate(oldBest,newBest,
SE,beta)
%translation transfomration
n = length(oldBest);
y = repmat(newBest',1,SE) +
beta/(norm(newBest-oldBest)+eps)
*reshape(kron(rand(SE,1),
(newBest-oldBest)'),n,SE);
y = y';

function y = op_expand(Best,SE,gamma)
%expansion transformation
n = length(Best);
y = repmat(Best',1,SE) + gamma*
(normrnd(0,1,n,SE).*repmat(Best',1,SE));
y = y';

function y = op_axes(Best,SE,delta)
% axesion transformation
n = length(Best);
A = zeros(n,SE);
index = randint(1,SE,[1,n]);
A(n*(0:SE-1)+index) = 1;
y = repmat(Best',1,SE) + delta*
normrnd(0,1,n,SE).*A.*repmat(Best',1,SE);
y = y';
```

Each of these functions aims to generate *SE* random points with special property based on incumbent best solution.

The functions *expansion*, *rotation*, *axesion* are to generate random points and update incumbent best solution by using corresponding state transition operators. Taking the *expansion* function for example, it is shown as below

```
function [Best,fBest] = expand(funfcn,
oldBest,SE,Range,beta,gamma)
Pop_Lb=repmat(Range(1,:),SE,1);
Pop_Ub=repmat(Range(2,:),SE,1);
Best = oldBest;
fBest  = feval(funfcn,Best);
flag = 0;
State = op_expand(Best,SE,gamma);
changeRows = State > Pop_Ub;
State(find(changeRows))
= Pop_Ub(find(changeRows));
changeRows = State < Pop_Lb;
State(find(changeRows))
= Pop_Lb(find(changeRows));
[newBest,fGBest] = fitness(funfcn,State);
if fGBest < fBest
    fBest = fGBest;
    Best = newBest;
    flag = 1;
else
    flag = 0;
end

if flag ==1
    State =
    op_translate(oldBest,Best,SE,beta);
    changeRows = State > Pop_Ub;
    State(find(changeRows))
    = Pop_Ub(find(changeRows));
    changeRows = State < Pop_Lb;
    State(find(changeRows))
    = Pop_Lb(find(changeRows));
    [newBest,fGBest]
    = fitness(funfcn,State);
    if fGBest < fBest
        fBest = fGBest;
        Best = newBest;
    end
end
```

and it can be found that the translation transformation is executed only when a better solution is found by the expansion

transformation (flag =1).

The matlab codes of *rotation* function and *axesion* function are very similar to that of *expansion* function. The only difference is that the subfunction *op_expand* is replaced by *op_rotate* function or *op_axes* function. For more details of the *rotation* and *axesion* functions, please refer to [16].

4 Conclusion

In this paper, a matlab toolbox for the standard continuous STA is described, and several instances are given to show how to use the STA toolbox. The core functions in the STA toolbox are explained as well. An available link to download the STA toolbox is also provided for reference.

References

[1] X.J. Zhou, C.H. Yang and W.H. Gui, Initial version of state transition algorithm, *in the 2nd International Conference on Digital Manufacturing and Automation (ICDMA)*, 2011:644–647.

[2] X.J. Zhou, C.H. Yang and W.H. Gui, A new transformation into state transition algorithm for finding the global minimum, *in the 2nd International Conference on Intelligent Control and Information Processing (ICICIP)*, 2011: 674-678.

[3] X.J. Zhou, C.H. Yang and W.H. Gui,State transition algorithm, *Journal of Industrial and Management Optimization*, 8(4): 1039–1056, 2012.

[4] X.J. Zhou, D.Y. Gao and C.H. Yang, A Comparative study of state transition algorithm with harmony search and artificial bee colony, *Advances in Intelligent Systems and Computing*, 212: 651-659, 2013.

[5] C.H. Yang, X.L. Tang, X.J. Zhou and W.H. Gui, A discrete state transition algorithm for traveling salesman problem, *Control Theory & Applications*, 30(8): 1040–1046, 2013.

[6] J. Han, T.X. Dong, X.J. Zhou, C.H. Yang and W.H., Gui, State transition algorithm for constrained optimization problems, *in the 33rd Chinese Control Conference (CCC)*, 2014: 7543–7548.

[7] X.J. Zhou, D.Y. Gao, C.H. Yang and W.H. Gui, Discrete state transition algorithm for unconstrained integer optimization problems, *Neurocomputing*, 173: 864–874, 2016.

[8] J. Han, X.J. Zhou, C.H. Yang and W.H., Gui, A multi-threshold image segmentation approach using state transition algorithm, *in the 34th Chinese Control Conference (CCC)*, 2015: 2662–2666.

[9] T.X. Dong, X.J. Zhou, C.H. Yang and W.H. Gui, A discrete state transition algorithm for the task assignment problem, *in the 34th Chinese Control Conference (CCC)*, 2015:2692–2697.

[10] X.L. Tang, C.H. Yang, X.J. Zhou and W.H. Gui, A discrete state transition algorithm for generalized traveling salesman problem, *Advances in Global Optimization*, 137–145, 2015.

[11] X.J. Zhou, S. Hanoun, D.Y. Gao and S. Nahavandi, A multiobjective state transition algorithm for single machine scheduling, *Advances in Global Optimization*, 79–88, 2015.

[12] X.J. Zhou, C.H. Yang and W.H. Gui, Nonlinear system identification and control using state transition algorithm, *Applied Mathematics and Computation*, 226:169–179, 2014.

[13] X.J. Zhou, D.Y. Gao and A.R. Simpson, Optimal design of water distribution networks by a discrete state transition algorithm, *Engineering Optimization*, 48(4): 603–628, 2016.

[14] Y.L. Wang, H.M. He, X.J. Zhou, C.H. Yang, Y.F. Xie, Optimization of both operating costs and energy efficiency in the alumina evaporation process by a multi-objective state transition algorithm, The Canadian Journal of Chemical Engineering, 94: 53–65, 2016.

[15] G.W. Wang, C.H. Yang, H.Q. Zhu, Y.G. Li, X.W. Peng, W.H. Gui, State-transition-algorithm-based resolution for overlapping linear sweep voltammetric peaks with high signal ratio, Chemometrics and Intelligent Laboratory Systems, 151:61–70, 2016.

[16] http://www.mathworks.com/matlabcentral/fileexchange/52498-state-transition-algorithm.