Fast Micro-Differential Evolution for Topological Active Net Optimization

Yuan-Long Li, *Student Member, IEEE*, Zhi-Hui Zhan, *Member, IEEE*, Yue-Jiao Gong, *Member, IEEE*, Jun Zhang, *Senior Member, IEEE*, Yun Li, *Member, IEEE*, and Qing Li, *Senior Member, IEEE*

***Abstract*—This paper studies the optimization problem of topological active net (TAN), which is often seen in image segmentation and shape modeling. A TAN is a topological structure containing many nodes, whose positions must be optimized while a predefined topology needs to be maintained. TAN optimization is often time-consuming and even constructing a single solution is hard to do. Such a problem is usually approached by a "best improvement local search" (BILS) algorithm based on deterministic search (DS), which is inefficient because it spends too much efforts in nonpromising probing. In this paper, we propose the use of micro-differential evolution (DE) to replace DS in BILS for improved directional guidance. The resultant algorithm is termed deBILS. Its micro-population efficiently utilizes historical information for potentially promising search directions and hence improves efficiency in probing. Results show that deBILS can probe promising neighborhoods for each node of a TAN. Experimental tests verify that deBILS offers substantially higher search speed and solution quality not only than ordinary BILS, but also the genetic algorithm and scatter search algorithm.**

***Index Terms*—Differential evolution (DE), grid deformation, structure optimization, topological active net (TAN), topological optimization.**

I. Introduction

OFTEN found in image segmentation and shape modeling, a topological active net (TAN) [1] is an example of complex structure which often needs to be optimized via a time-consuming optimization procedure. A TAN usually comprises a 2-D elastic mesh of interrelated nodes, whose construction requires dynamic structural optimization, as nodal positions must be optimized while maintaining their predefined topology [2]. As illustrated in Fig. 1, a TAN uses internal nodes to find the inner structure of a target object and uses external nodes to reveal boundary information by adjusting the nodal positions. An energy function defined on the position of the nodes is minimized when the TAN fits the target object best. Such an optimization problem can be challenging in real time and usually complex due to the large number of correlated nodes [3].

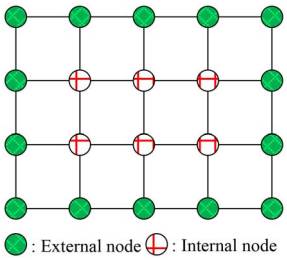

Fig. 1. Example of a TAN.

Several TAN optimization methods have been developed so far [3], [5], [8]. The "best improvement local search" (BILS) algorithm [3] is a well-known deterministic search (DS) algorithm for TAN optimization. It is an exhaustive process, during which each node searches its square neighborhood and moves to a new position that reduces the energy most. The BILS iterates this process throughout the net until no movement can reduce the energy any further. Although, BILS is a generic framework for searching around the neighbors by defining various neighborhood structures, it appears that no BILS variant has made use of historical information to adjust the search. That is, no matter how large the window, BILS is deterministic and exhaustive that no historical information is used to guide the search or to improve the efficiency.

On the other hand, evolutionary algorithms (EAs) such as genetic algorithm (GA) [4], [5], differential evolution (DE) [6], [7], and scatter search (SS) [8] have also been used for the TAN problem recently. An EA is a population-based coarse search algorithm and hence can make use of global information of the TAN. However, due to the accuracy requirement of the TAN, EAs need to refine their local search.

Manuscript received March 10, 2015; accepted May 13, 2015. This work was supported in part by the National Natural Science Foundations of China (NSFC) under Grant 61402545, in part by the NSFC Key Program under Grant 61332002, in part by the NSFC for Distinguished Young Scholars under Grant 61125205, in part by the Natural Science Foundations of Guangdong Province for Distinguished Young Scholars under Grant 2014A030306038, in part by the Project for Pearl River New Star in Science and Technology, Guangzhou, China, in part by the Fundamental Research Funds for the Central Universities under Grant 15lgzd08, in part by the National High-Technology Research and Development Program (863 Program) of China under Grant 2013AA01A212, and in part by the Strategic Research Grant from the City University of Hong Kong under Grant 7004218. This paper was recommended by Associate Editor K.-C. Tan. *(Corresponding author: Zhi-Hui Zhan.)*

Y.-L. Li, Z.-H. Zhan, Y.-J. Gong, and J. Zhang are with the School of Advanced Computing, Sun Yat-sen University Guangzhou 510275, China, also with the Key Laboratory of Machine Intelligence and Advanced Computing, Sun Yat-sen University, Guangzhou 510275, China, also with the Engineering Research Center of Supercomputing Engineering Software, Sun Yat-sen University, Guangzhou 510275, China, and also with the Key Laboratory of Software Technology, Sun Yat-sen University, Guangzhou 510275, China (e-mail: zhanzhh@mail.sysu.edu.cn).

Y. Li is with the University of Glasgow, Glasgow G12 8QQ, U.K.

Q. Li is with the City University of Hong Kong, Hong Kong.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2015.2437282

To learn from historical search information, we develop an efficient EA-based approach to TAN optimization in this paper. Different from BILS that searches the neighborhood exhaustively with DS strategy, we propose to use the random search (RS) strategy in EAs to replace DS so as to improve the efficiency and speed of BILS. In particular, we propose the use of DE because it is a very efficient RS algorithm. With DE, a new BILS algorithm named deBILS for TAN optimization problem is developed in Section III, following an outline of DE and the TAN problem in Section II. In Section IV, experiments are conducted to test and verify the proposed algorithm. The conclusion is drawn in Section V.

II. Topological Active Net and Differential Evolution

A. Problem Formulation of TAN Optimization

Here we focus on the extended TAN (ETAN) proposed in [9], which is a new model extending TAN in the definition of the energy function used in [8]. The ETAN also consists of two kinds of nodes, the internal and the external ones. In the definition of the net energy, there are also two kinds of energies, the internal and the external ones. The first energy is used to control the structure of the net while the second energy is used to fit the net to the target object.

For a 2-D ETAN V, assume its nodes are placed in a normalized grid $G = [0, 1]^2$. For a node $(t, s) \in G$, its position is $v(t, s) = (x(t, s), y(t, s))$, where x and y are coordinates and t and s are indices of the node. The energy of the net is then defined as

$$E(v(t,s)) = \int_0^1 \int_0^1 [E_{\text{int}}(v(t,s)) + E_{\text{ext}}(v(t,s))]dtds \quad (1)$$

where E_{int} is the internal energy and E_{ext} is the external energy.

The internal energy of a node, which controls the net structure, depends on the first and second order of derivatives of the node

$$E_{\text{int}}(v(t,s)) = \alpha\Big[|v_t(t,s)|^2 + |v_s(t,s)|^2\Big] + \beta\Big[|v_{tt}(t,s)|^2 + |v_{ts}(t,s)|^2 + |v_{ss}(t,s)|^2\Big] \quad (2)$$

where v_t and v_s are the first order derivatives, v_{tt}, v_{ts}, and v_{ss} are the second order derivatives, and α and β are weight parameters. The computation of the derivatives is estimated by its nearby nodes [8].

The external energy measures how suitable a node is placed, as an internal node should be placed inside the target object while the external node should be placed on the boundary of the object. Hence, the external energy of node (t, s) is defined differently

$$E_{\text{ext}}(v(t,s)) = \omega f[I(v(t,s)] + \frac{\rho}{|\chi(t,s)|}\sum_{p\in\chi(t,s)}\frac{1}{||v(t,s)-v(p)||}f[I(v(p)] \quad (3)$$

where $\chi(t, s)$ is the neighbor nodes of (t, s) and f is a function defined by

$$f[I(v(p))] = \begin{cases} \gamma\overline{I(v(p))_n} + \\ \sigma\overline{I(\text{Link}(v(t,s),p))} & \text{for internal nodes} \\ I_{\max} - \overline{I(v(p))_n} \\ +\ \xi(G_{\max} - G(v(p))) & \text{for external nodes} \\ +\ \delta DG_{\text{evfc}}(v(p)) \end{cases} \quad (4)$$

where $I(v(p))_n$ and $G(v(p))$ in a computer vision application, for example, are the intensity values of node p in an original image and its gradient image. Here, $\overline{I(v(p))_n}$ denotes the average intensity value of the original image in an n^2-sized window, $\overline{I(\text{Link}(v(t,s),p))}$ denotes the mean intensity value along the path from p to $v(t, s)$, which are proposed in this paper as it leads to better results in optimizing TANs for nonconvex objects. $I_{\max}$ and $G_{\max}$ denote the maximum intensity values of the original and the gradient images, respectively, DG_{evfc} denotes the distance of the node to its nearest edge in the gradient image, which is computed through an extended vector field convolution (EVFC) [10]. Detailed computation of DG_{evfc} can be found in [8].

The objects to detect in computer vision are assumed to be dark on a bright background here, such that the lower value of f indicates a better net fitting.

B. Differential Evolution as Efficient EA

The EA is an RS framework with a population and evolutionary operations. It is a collection of nonspecialized black-box optimization methods like GA and DE. As we incorporate a DE procedure in the proposed algorithm, we briefly introduce DE here as a typical EA.

The DE algorithm has three basic operations for population reproduction: 1) crossover; 2) mutation; and 3) selection for reproduction. The mutation operation of DE is very special in that it uses a linear combination of a base vector and one (or more) differential vector(s) to generate the mutated vector. For example, for every individual p_i in P, its mutated vector q_i is generated by

$$q_i = p_{r_1} + F \cdot \left(p_{r_2} - p_{r_3}\right) \quad (5)$$

where r_1, r_2, and r_3 are three different randomly selected individuals, and are also different from i. In this mutation scheme, the difference between individuals r_2 and r_3 is used as the mutation step while factor F controls the step scale. After a mutated population Q is created, it will then go through the crossover process with the parent population P.

The crossover operation recombines every pair of individuals of (q_i, p_i) in order to generate a new individual u_i

$$u_{ij} = \begin{cases} q_{ij}, & \text{if rand}(0,1) \le \text{CR} || j == j\text{rand} \\ p_{ij}, & \text{otherwise} \end{cases}, \text{for } j = 1, \ldots, D \quad (6)$$

where rand(0,1) is a random number uniformly distributed in the interval [0, 1], D is the number of dimensions, and CR is the crossover rate which controls how many dimensions of the newly generated individual come from the mutated vector q_i while $j$rand is a randomly selected index in order to make

sure that at least 1-D of the mutated vector will enter into the newly generated individual. The crossover process creates a temporary population U which will be evaluated and then enters into the selection procedure. The selection procedure of DE uses a pairwise comparison of U and P. As shown in (7), individual u_i and p_i are compared and the better one will enter into the next generation

$$p_i^{\text{new}} = \begin{cases} u_i, \text{ if fitness}(u_i) \text{ is better than fitness}(p_i) \\ p_i, \qquad\qquad\qquad \text{otherwise.} \end{cases} \tag{7}$$

Like other EAs, various studies on the DE framework have been carried out to improve the efficiency and search quality [11]. Parameter adaptation methods have been widely used in DE [12]–[14], [16], [34], [36], which have proved to be important for DE to perform well on different kinds of problems. In [32], a bare-bone DE is proposed to solve the parameter setting problem of DE in a different way. Using different mutation strategies to improve the performance of DE are also widely studied [13]–[15], [19]. Different ways have been proposed to improve the mutation operations [17], [18], [20], [22], [26], [28], [35]. The selection mechanism was studied in [25], while the evolution path strategy is used to guide the evolution in [33] and co-evolutionary is used in [37]. A theoretical study on the convergence characteristics of DE can be found in [23]. Besides, DE based frameworks are also used in constrained optimization [24] and dynamic optimization [21], [29], [31].

III. deBILS Algorithm

Different from DS in BILS, the deBILS uses RS like in an EA to make use of historical information to guide the search more efficiently. That is, deBILS does not probe all the neighbor points of a node when try to find a better point for this node. The deBILS can randomly but intelligently probe some nearby points without necessarily searching all the points in the square window. Although this may face the risk that the best neighbor point is missed, the risk can be reduced if deBILS does the RS for each node by using multiple individuals instead of only one individual. That is, if a micro-DE population is used to perform the RS in deBILS, the search diversity and interactions among different individuals can be maintained to find promising new point.

The micro-DE is adopted as the RS in deBILS mainly because that the mutation mechanism in DE is simple and is proven to be powerful by many benchmark studies [12]–[18] and applications [27], [30]. At every iteration in the deBILS algorithm, the DE mechanism is used to search for a better new position for each node. Learning from historical information, deBILS can thus find certain good quality positions from fewer trials than the BILS does for a TAN.

Moreover, after the position of each node has been modified by the micro-DE, deBILS can further utilize the common DE search mechanism to optimize the positions of all the nodes in the TAN as a whole like other EA-based TAN optimizers. Therefore, deBILS makes use of global information of all the node to generate more promising TANs efficiently.

(*x*,*y*,*type*) of Node 1	(*x*,*y*,*type*) of Node 2	(*x*,*y*,*type*) of Node ...	(*x*,*y*,*type*) of Node $R\times C$-1	(*x*,*y*,*type*) of Node $R\times C$

Fig. 2. Encoding of an individual in deBILS.

A. Framework of deBILS

The deBILS algorithm retains a framework similar to traditional BILS in which all the nodes adjust their position iteratively one-by-one. Each individual in deBILS encodes a TAN with all the nodes' position and type, as shown in Fig. 2. The type indicates whether the node is an internal or an external node and $R \times C$ is the total number of nodes in a TAN of R rows and C columns. At the beginning, deBILS initializes a micro-population of individuals to present a number of randomly generated TANs. During the following evolution process, deBILS uses two procedures to optimize the positions of the nodes in every generation. The first procedure is a "node-by-node optimization" procedure and the second is an "individual-by-individual optimization" procedure.

In the node-by-node optimization procedure, deBILS optimizes the position of the nodes one-by-one. When adjusting the position of each node, deBILS does not perform a thorough DS as in traditional BILS, but uses an RS mechanism like the DE mutation to find the next position. With the help of historical information, trials likely to contain potentially promising directions to let deBILS offer potentially higher efficacy and thus efficiency in generating promising positions. This makes the runtime of deBILS in generating a high-quality TAN individual much shorter than BILS does.

In the individual-by-individual optimization procedure, deBILS optimizes the positions of all the nodes simultaneously. Each individual represents a whole TAN. During the process, deBILS optimizes the individuals one-by-one through the common DE evolutionary mechanism.

A generic flowchart of the deBILS algorithm is shown in Fig. 3. The T_1, T_2, T_3 are the three individuals which are initialized to a same basic solution net_0. The updated is a variable to denote if any progress is made in a generation. Details of the node-by-node optimization procedure and the individual-by-individual optimization procedure are described in the following sections.

B. Node-by-Node Optimization

The node-by-node search pattern in BILS is enhanced with a heuristically guided RS mechanism. Moreover, multiple individuals are used in the search of each node in deBILS to reduce the risk of missing the best neighborhood point. In each generation, the search loop is going from node-to-node, as shown in Fig. 4. It should be noticed that the outer loop of Fig. 4 is the node-by-node process. For each node, there is an inner loop controlled by each individual, as lines 3)–6) in Fig. 4. That is, deBILS does not let each node probe the entire window exhaustively but uses several individuals to probe certain positions according to historical information. As such, targeting TAN optimization, deBILS is a problem specific design that embeds EA operations into the BILS framework.

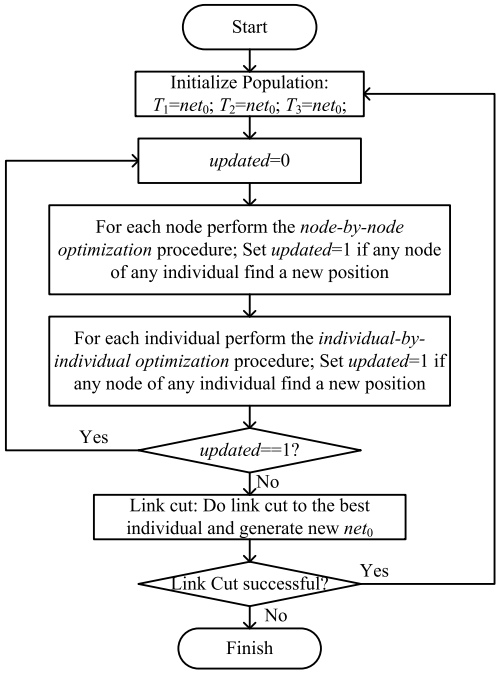

Fig. 3. Flow chart of deBILS. T_1–T_3 are the three individuals in the population which are initialized to a same basic solution net_0 while updated is used to record whether a new better solution is found or not.

<The following pseudo code corresponding to the "node-by-node" optimization in Fig. 3>

1)**For** each node in the TAN
2) **Read** information left by neighbor nodes from all individuals and generate candidate search direction pool V_{pool};
3) **For** each individual
4) **Do the search** for this node of this indivdiual;
5) **Update** heuristic information.
6) **End**
7)**End**

Fig. 4. Node-by-node optimization procedure.

1) Reading and Updating Historical Information: As shown in Fig. 4, node-by-node search is set to be the outer control loop as in the BILS algorithm, which makes it easy to utilize useful historical information left by the neighboring nodes. Each node searches for the next better position by its individuals (that is, the individuals in the inner loop of this node). Herein, a successful step vector (initially zero) of node i last made by individual j is denoted as $V_{\text{step}}(\text{node}_i, \text{ind}_j)$, which stands for a step toward an improved direction. Otherwise individual j of node i will not make such a move. Therefore, when each node searches for its next better position, the successful step vectors of its neighborhood nodes can be used as historical search direction information to guide the search. The reason is that nearby nodes are highly likely to move along similar directions (like in Fig. 5) and the same nodes of other individuals can also make similar steps. Hence, it is useful to learn from other individuals.

Specifically, when make the search for each node i [step 1) in Fig. 4], the candidate mutation vector pool named V_{pool}

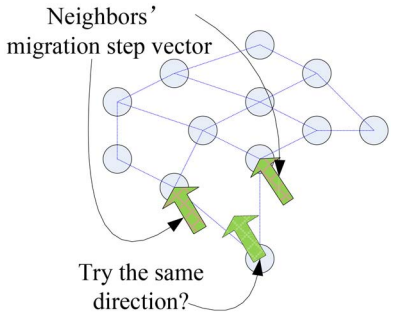

Fig. 5. Neighborhood historical information.

is generated first [step 2) in Fig. 4]. Specifically, at step 2), if V_{step} of all the individuals in the neighbor nodes are not zero, V_{pool} is initialized as $V_{\text{pool}} = \{V_{\text{step}}(\text{node}_i, \text{ind}_j)\}$, where $i \in \{1, \ldots, \text{size of current neighborhood}\}$ and $j \in \{1, \ldots, \text{popsize}\}$. For a node at row t and column s, its neighbor node set is defined as the set of nodes with (row, column) index of $\{(t-1, s-1), (t-1, s+1), (t+1, s-1), (t+1, s+1)\}$. At step 5), V_{step} and V_{pool} are updated dynamically if a new better position is found at step 4).

2) Search Steps: Following the historical information based mutation vector pool V_{pool} generated at step 2), node optimization then goes through each individual as in step 4). The search of each individual for each node is based on two kinds of mutation vector generation methods, noted as methods 1 and 2.

Method 1 uses a vector randomly selected from V_{pool} if it is not empty. The step vector v_{m1} of the current node is generated based on a randomly selected vector from V_{pool} and is scaled by a random factor

$$v_{m1} = (3 \cdot \text{rand} + 0.5) V_{\text{pool}}(r) \tag{8}$$

where $V_{\text{pool}}(r)$ is a random vector selected from V_{pool} and $3 \cdot \text{rand} + 0.5$ is the random scale factor, with rand being a random number within the interval [0, 1]. Increasing the scale factor too much will lead the algorithm to wastefully venturing outside the original search window while decreasing the factor too much will miss out a potentially promising direction. The scale factor is hence set empirically with a good tradeoff between searching inside and outside the window, which has worked well as seen in the experiments.

Method 2 generates a mutation vector, i.e., the step vector v_{m2}, from a square window of size $2n + 1$ centered at the current node's position, as

$$v_{m2} = \text{WindowStep(random)} \tag{9}$$

where WindowStep(random) indicates the vector from the node's current location to a randomly selected position from the square window (the same as the search window of BILS).

Here, two kinds of mutation vectors are used in the following fashion: v_{m1} is used, if V_{pool} is not empty and the times of using method 1 is smaller than a maximum try number heu_try; otherwise, v_{m2} is used before the total try number achieved a maximum try number total_try, as shown by Fig. 6, which is the detail of step 4) in Fig. 4.

Due to that both v_{m1} and v_{m2} are generated randomly, a searched mask is initialized before the search in order to

```
<The following pseudo code corresponding to step 4 in Fig.4>
trycount=0;
k=0;
While trycount<total_try
   if k<heu_try
      vm1=genrate vm1 with method 1.
      while vm1 is illegal (already searched or out of the image range)  (*)
         vm1=genrate vm1 with method 1.
      end
      k=k+1;
      trycount=trycount+1;
   else
      vm2=generate vm2 with method 2.
      while vm2 is illegal &&
            (WindowStep have more than one unsearched vector (**))
         vm2=generate vm2 with method 2.
      end
      trycount=trycount+1;
   end
end
```

Fig. 6. Search step generation procedure.

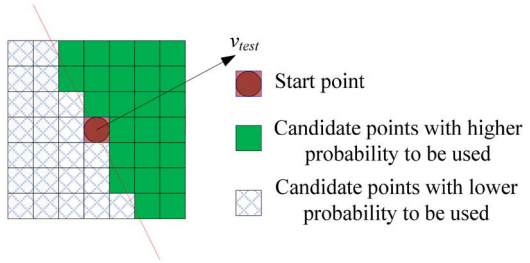

Fig. 7. Heuristic search window cut.

avoid searching the same point. If a new step vector might lead the node to a point that has been marked or out of the range of the image, the step vector will be regenerated (the step marked * in Fig. 6). The search will stop if all the vectors in WindowsStep have been tried (the step marked ** in Fig. 6).

The usage of the pool V_{pool} in method 1 is very helpful in finding good search directions of the node quickly while method 2 maintains the search diversity.

A simple heuristic is also used in mutation method 2 to focus on more promising areas with the help of V_{pool} vectors as shown in Fig. 7. As the search window in (9) is a square window centered at the current node's position, better move steps should be at a similar direction to vectors from V_{pool}. Hence, almost half of the window is helpless. We can use a vector v_{test} randomly chosen from V_{pool} as a guidance vector. If the v_{m2} generated by mutation method 2 has a large angle ($>\pi/2$) with v_{test}, we will regenerate v_{m2} with a high probability, e.g., 0.9 used in our experiments. The v_{m2} regeneration condition (**) in Fig. 6 for method 2 is then added by "angle(v_{m2}, v_{test}) $> \pi/2$ and rand < 0.9." Note that a probability of 0.9 is used here because v_{test} based useful vector judgment is not always right. Such a pruning method cannot be used in a DS algorithm because it requires that the guided half of the opposite v_{test} direction must be strictly proved useless.

Following the step vector is generated, a net topology test is carried out in order to make sure that the move step will not destroy the net topology. The area test method of [8] is used here. If the step vector (v_{m1} or v_{m2}) passes the test, the local energy of the node after moving with the step vector is computed. If it is smaller than the original energy, the position will be recorded. Like the BILS algorithm, after all trials have been exhausted, the best trial will be used to update the position of this individual for this node. The step vector which generates the new position will be used to update V_{step} and will be added to search candidate pool V_{pool} immediately. V_{step} and V_{pool} are thus dynamically updated during the search process so that the algorithm can use the most recent progress information in the search as the historical information. Such a mechanism makes the communication among individuals in the population more efficient than a normal DE mutation by taking the TAN as a whole.

C. Individual-by-Individual Optimization

After all the nodes have been optimized by the above process, a normal DE mutation will be performed to optimize each TAN as a whole in the following way.

As each individual represents a whole TAN, each individual stores the positions of all the nodes of the whole TAN. The global energy of each individual (a whole TAN) in the population is computed and the individual with the best energy is denoted by i_{best}.

For each individual $T(i)$ in the population, the common DE mutation is applied so as to generate a new TAN $T'(i)$

$$T'(i) = T(i_{\text{best}}) + (0.5 + 0.3 \cdot \text{rand})[T(r_1) - T(r_2)] \quad (10)$$

where $T(r_1)$ and $T(r_2)$ are two randomly selected individuals (TANs) from the current population.

Then, we compute the energy of $T'(i)$ and compare it with the energy of $T(i)$. If the former is better, it will replace $T(i)$ and i_{best} will also be updated if $T(i)$ is better than the original i_{best} individual.

Mutation is then performed for each individual of the population. If a better solution is found during this process, it means that the mutation as (10) is helpful. In this case, the mutation for the entire TAN, as (10), will be carried out one more time for all individuals. The loop terminates until it fails in generating better individuals. The TAN as a whole mutation is a fast TAN generation method which is useful in the early stage of the search process.

The generation of a new TAN T' by the normal DE mutation in the above optimization process results in some conflicting nodes that occupy the same point. In this case, we find a nearest legal point (which can retain the topology of the TAN and is not occupied by any other nodes) for each of the conflicting nodes to make sure that they are not placed at the same point.

D. Notes

1) Comparison of BILS and deBILS on Their Search Methodology: A fundamental difference between BILS and deBILS is that BILS is a DS algorithm while deBILS is a random heuristic search algorithm based on micro-DE. Historical information is useful when one chooses not to carry out a thorough search like BILS, hence making the search much faster, although risk exists that the best step may not be found.

Incorporating historical information in RS, deBILS can also search points outside the basic window, whilst BILS only searches inside. If the historical information is carefully used, it is highly possible that deBILS can find with a high probability a step of a high quality. Then with the help of multiple individuals, the probability will be further improved toward better or similar steps, which will enhance robustness of the random algorithm. Interaction among individuals can provide learning from better ones for a promising search direction. Hence, deBILS runs faster and can yield better results compared with BILS because of the micro-evolutionary framework.

2) Link-Cut Procedure: The link-cut procedure can be used to divide the net into more than one subnets to fit more than one target objects. The link-cut procedure itself is not studied in this paper because it relates to the problem model but not the algorithm design itself. However, it can be directly added to the deBILS procedure in the same way as BILS. After all the individuals in the population stopped improving, the link-cut operation will be carried out and the best individual will go through the process. Following that, the topology of the net will be changed and the nodes are able to optimize further. This is an optional operation that can be used in both BILS and deBILS. For a detailed method of link-cut, refer to [8].

IV. Experiments

In this section, we present experimental results on the comparison between the existing BILS approach and the proposed deBILS. We also compare the performance of deBILS with other EA-based TAN optimization approaches including the GA-based approach and SS-based approach.

All experiments are carried out on a PC with an Intel Core i5-2300 CPU running MATLAB 2013a. The deBILS is first compared with BILS on ten single target test images with a TAN grid size of 15×15. To focus on the nodes' position optimization procedure in these tests, we compare the two algorithms on images with only one target object without link cutting procedure. In fact, the nodes' position optimization process is the most time-consuming part of TAN optimization. Therefore, if a proposed algorithm can achieve better efficiency on the nodes' position optimization without link cutting procedure, it also should be useful to replace the original BILS algorithm when link cutting procedure is used. To prove the above analysis more clearly, we further provide tests with a link cutting procedure on ten more test cases with multiple targets. Note that all the test images used here are binary images. This is reasonable because like in [8], the BILS (and deBILS) are used to optimize a TAN after the original image has gone through preprocessing and been transformed into a binary image.

The parameters used specifically in the deBILS algorithm are set to be the same for all test images as shown in Table I. For parameter window size n and population size, we carry out experimental studies on different settings in the following parameter studies Section IV-A. Apart from the algorithm parameters, different test images require different energy function parameter settings, as in [8]. Herein, we set these parameters as in Table II, which are optimized for BILS. Note that, as we focus on the optimization algorithm, we do not show more analysis on these parameters because they are pertinent to the TAN problem per se. Interested readers are referred to [8] for more information.

TABLE I
Parameters Settings Used in the Following Tests

deBILS independent run times	10
Population size	3
total_try	(*number of points in the window*)/4
heu_try	(*number of points in the window*)/8
Image size	436×416
TAN size	15×15
Square window size $\pm n$	5

TABLE II
Energy Function Parameters Used for Different Images

Test image	α	β	ω	ρ	ξ	δ	γ	σ
1	1	3	3	1	10	15	5	20
2	1	2	3	1	15	15	10	10
3	1	1	3	1	10	15	5	10
4	1	1	3	1	10	15	5	10
5	1	1	3	1	10	15	5	10
6	1	1	3	1	10	20	5	10
7	1	1	3	1	10	15	5	10
8	1	1	3	1	10	15	5	10
9	1	1	3	1	10	15	5	10
10	1	1	3	1	10	15	5	15

TABLE III
Search Results Comparison on TAN Energy

Image	ER		TAN energy	
	BILS	deBILS(mean±std)	BILS	deBILS(mean±std)
1	0.53%	0.53%±0.03%	1.79E+05	1.80E+05±5.47E+02
2	57.61%	**11.07%±4.65%**	5.34E+05	**2.18E+05±3.77E+04**
3	3.25%	**3.07%±0.14%**	1.62E+05	**1.60E+05±1.33E+03**
4	7.56%	**3.41%±0.77%**	2.25E+05	**1.87E+05±5.15E+03**
5	6.27%	**5.55%±0.35%**	1.78E+05	1.84E+05±4.55E+03
6	22.35%	**11.76%±8.45%**	3.80E+05	**2.88E+05±6.31E+04**
7	7.11%	**6.08%±0.64%**	2.15E+05	**1.89E+05±4.40E+03**
8	6.06%	**4.12%±0.67%**	3.09E+05	**2.55E+05±1.77E+04**
9	2.67%	2.73%±0.12%	1.74E+05	**1.73E+05±7.02E+02**
10	8.02%	**7.21%±2.96%**	2.80E+05	**2.68E+05±1.93E+04**

The stop condition for BILS and deBILS is when no node's position can be further improved. Both TAN energy and true error rate (ER) of the resulted TAN are compared, where the TAN energy is as described in (1) in Section II and the ER is defined as

$$\text{ER} = \frac{\text{No.of pixel swrongly classified by the TAN}}{\text{No.of pixel soft he ground truth target}}. \quad (11)$$

A. Parameter Studies

We study key parameters for deBILS, including the basic search window size n used both in BILS and deBILS, the population size, and the split number for total_try and heu_try, i.e., the number like 4 and 8 in Table III that split the points in the window.

The basic search window size used in BILS and deBILS is set to 5 according to experimental studies, as shown in Fig. 8. It can be seen that with a larger n, both BILS algorithm

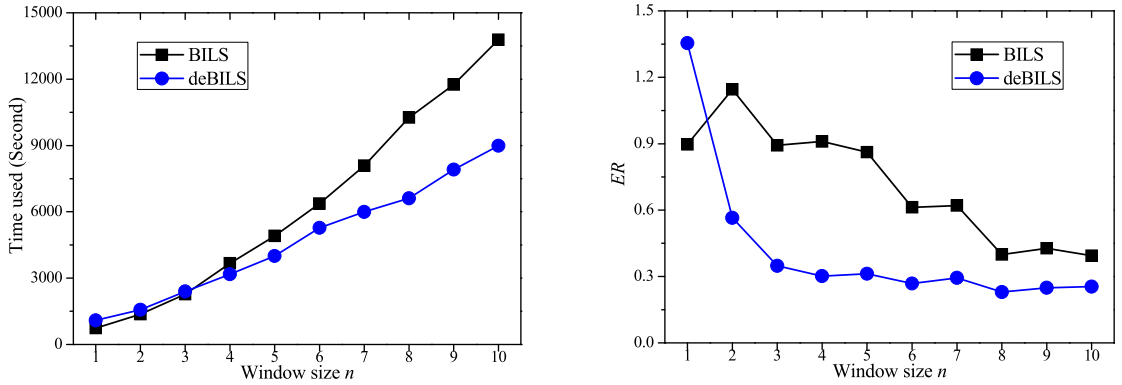

Fig. 8. Comparison of time used (second) and ER between BILS and deBILS on different search window sizes n.

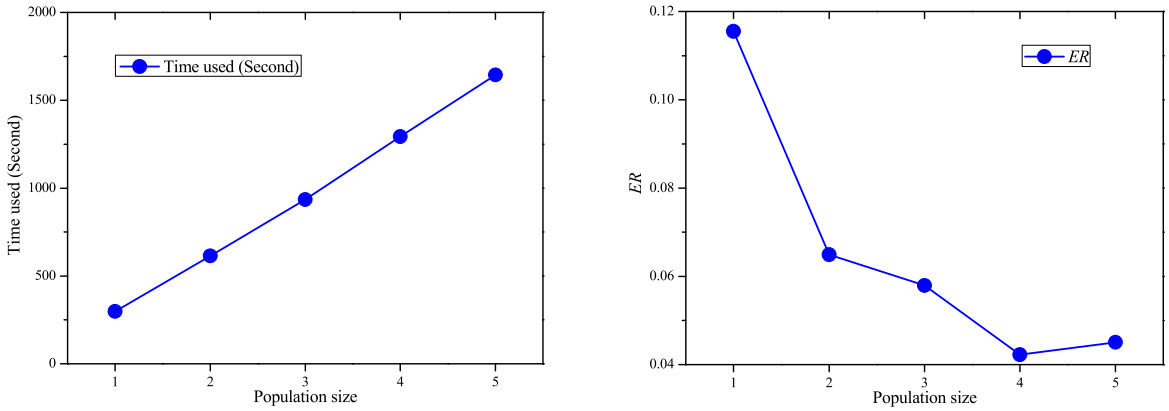

Fig. 9. Comparison of time used (second) and ER on different population sizes for deBILS.

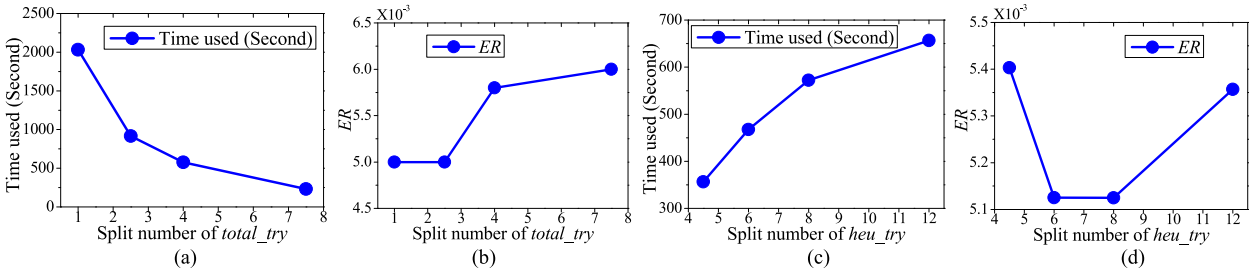

Fig. 10. Comparisons of time used (second) and ER with different settings of split numbers for total_try and heu_try. (a) CPU time with different split number of total_try. (b) ER values with different split number of total_try. (c) CPU time with different split number of heu_try. (d) ER values with different split number of heu_try.

and deBILS algorithm are able to find better quality TANs while the time consumed is increasing rapidly. On both ER and time indicators, deBILS outperforms BILS when n is larger than 2. To balance the solution quality and time cost, we set n to 5, which is also sufficient to offer much of the advantages of deBILS. Note that, Fig. 8 also reveals that the relative advantages of deBILS can increase with n increasing.

Fig. 9 shows typical results when different population sizes are used in deBILS. As can be anticipated and seen in the figure, the quality of the solution improves with a larger population size while the search time increases almost linearly with the population size. Yet again to strike a balance of the solution quality and time cost, we set the population size to 3. Fig. 9 also reveals that the size can be even smaller when the solution quality is good enough for a prespecified vision application.

Fig. 10 shows how the split number of total_try and heu_try in Table I affect the search performance. For the total_try, a larger split number means smaller trial number. Therefore, the time will be less, as shown in Fig. 10(a). However, large split number causes the ER increases, as can be seen from Fig. 10(b). To achieve a balance, therefore, we set the split number for total_try to 4 as shown in Table I. For heu_try, it controls how many trials use historical information to generate the mutation vector. A too small or too large value will decrease its usefulness. For example, a too small heu_try value (i.e., the split number for heu_try is too large) will render too little historical information and hence too little search efficiency as suffered by BILS. Fig. 10(c) confirms this.

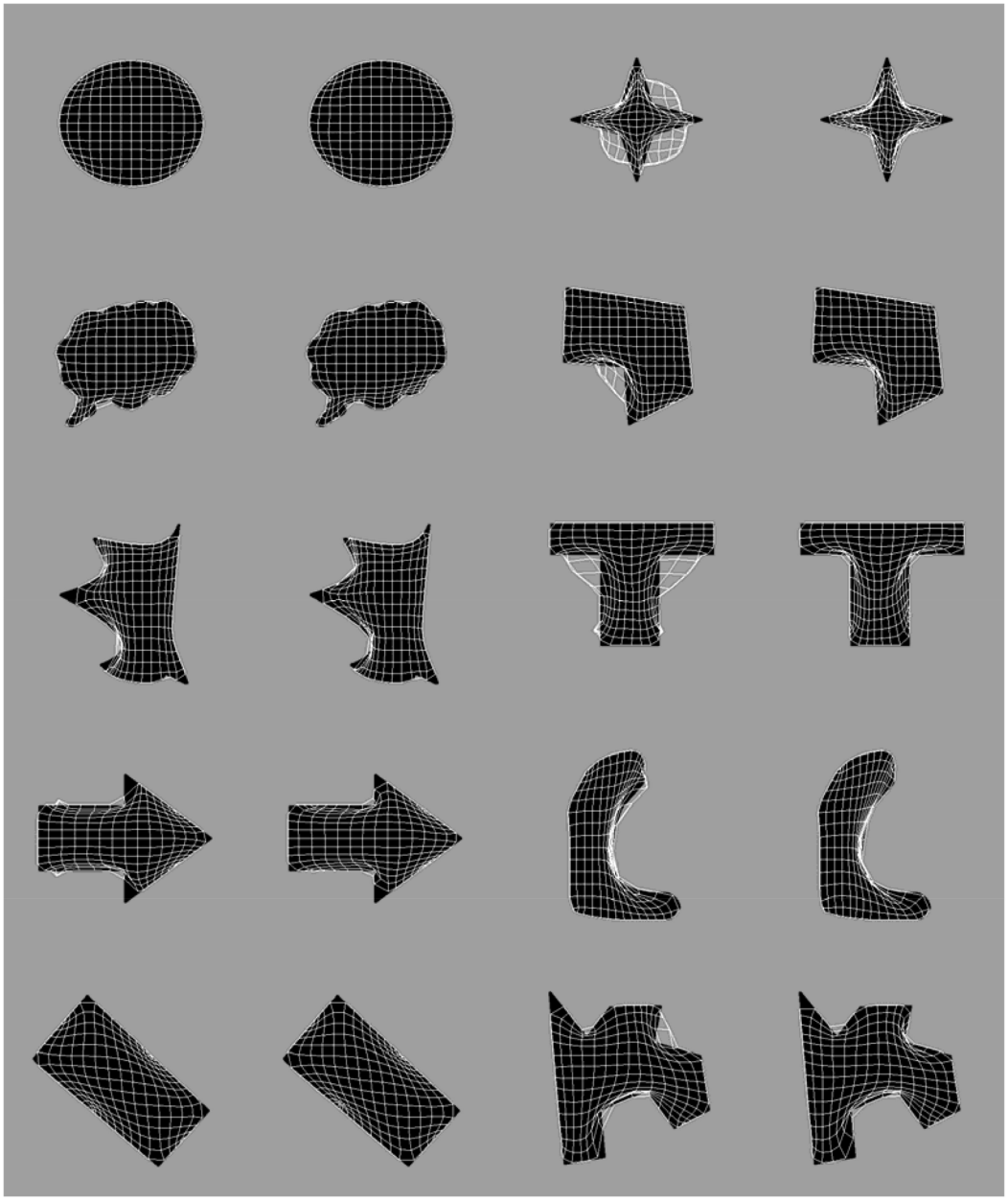

Fig. 11. TANs generated by BILS and deBILS on single target test images 1–10 (without link cutting procedure). For each pair of the same shape, the left TAN is generated by BILS and the right by deBILS.

However, a too small split number for heu_try (i.e., the tries with historical information become more) will cause the search unnecessarily random and hence will result in too large ER value, as confirmed by Fig. 10(d). Therefore, a medium value 8 is a good split number for heu_try, as given in Table I.

B. Experiments on Single Target Test Images

For a typical population size of 3, the TANs optimization results from BILS and deBILS on single target test cases are compared in Fig. 11. For each pair of the object images, the left TAN is generated by BILS and the right is generated by deBILS, which is seen to offer a better vision. Table III shows the average search results (energy and ER of the best TAN) of deBILS compared with those of BILS under the same conditions. Table IV shows the time used by BILS and deBILS to obtain the results in Table III. The obvious better results in the two tables are marked in **boldface**.

First, from the search quality comparison in Table III, we can see that only on two examples deBILS performs slightly worse than BILS, but on the other eight images deBILS outperforms BILS. Fig. 11 also shows that deBILS can generate better TANs than BILS does on nonconvex shapes, e.g., the star (image 2) and "*T*" (image 6) objects. This is due to that deBILS is capable of searching not only in a square window but also outside the square window along a potentially promising direction, as generated from the DE historical information.

TABLE IV
SEARCH TIME COMPARISON

Image	Search time (Seconds)		Saving by deBILS
	BILS	deBILS(mean±std)	
1	4.34E+02	**3.15E+02±2.14E+01**	**27.51%**
2	1.79E+03	**1.50E+03±4.59E+01**	**16.25%**
3	5.91E+02	**5.13E+02±3.00E+01**	**13.17%**
4	1.09E+03	**9.83E+02±6.68E+01**	**10.12%**
5	1.30E+03	**1.01E+03±6.63E+01**	**22.18%**
6	1.75E+03	**1.53E+03±6.85E+01**	**12.68%**
7	1.01E+03	**8.98E+02±3.32E+01**	**10.92%**
8	1.47E+03	**1.21E+03±7.99E+01**	**17.80%**
9	1.55E+03	**1.24E+03±6.52E+01**	**20.07%**
10	1.95E+03	**1.41E+03±1.18E+02**	**28.04%**

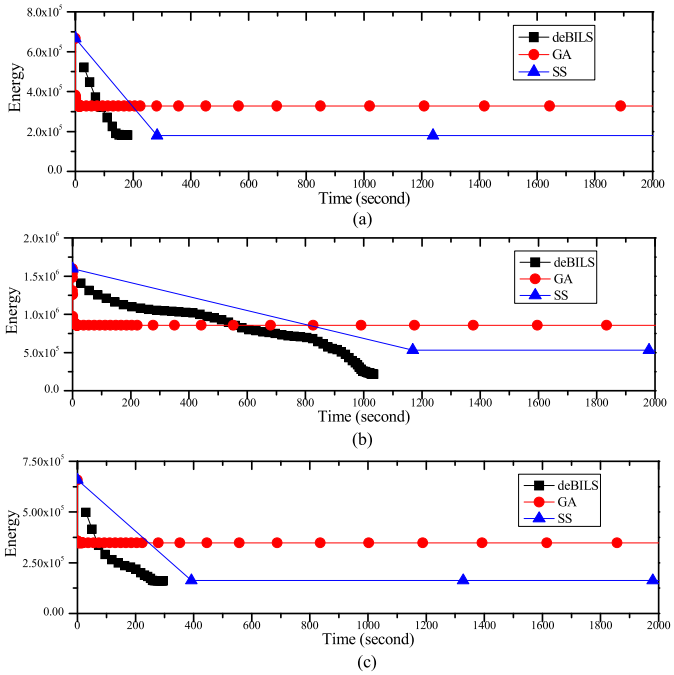

Fig. 12. Energy convergence curves against time of deBILS, GA, and SS on (a)–(c) images 1–3.

Second, as shown in Table IV, the search time used by deBILS is much less than BILS, offering a 28% improvement. This reduction can be further improved to about 70% for simple cases like images 1 and 4. The comparisons also show that the deBILS is more flexible than BILS because we can choose different population sizes depending on whether we need a higher quality TAN and/or faster optimization. deBILS is seen to be able to offer both in most single target cases.

C. Comparisons Between deBILS and TAN EAs

In order to verify that deBILS indeed achieve a balance on search quality and time, we also compare the results of deBILS with those of two EA-based TAN optimization algorithms: 1) a GA without BILS [5] and 2) a SS with BILS refinement [8]. The comparisons are based on the above single target test images. For the GA, a large population with 500 individuals was used, as suggested by Ibanez *et al.* [5].

TABLE V
ER COMPARISON OF deBILS, GA, AND SS

Image	deBILS(mean±std)	GA(mean±std)	SS(mean±std)
1	**0.53%±0.03%**	36.54%±3.00%	0.66%±0.03%
2	**11.07%±4.65%**	71.45%±0.35%	25.17%±1.45%
3	**3.07%±0.14%**	43.32%±0.36%	3.44%±0.27%
4	**3.41%±0.77%**	47.44%±4.48%	3.80%±0.59%
5	**5.55%±0.35%**	63.67%±0.32%	6.05%±0.01%
6	**11.76%±8.45%**	47.04%±0.34%	48.80%±31.06%
7	**6.08%±0.64%**	41.04%±3.64%	7.11%±0.00%
8	**4.12%±0.67%**	82.12%±0.67%	32.68%±28.20%
9	**2.73%±0.12%**	56.32%±0.09%	2.79%±0.00%
10	**7.21%±2.96%**	43.55%±0.09%	7.94%±0.47%

TABLE VI
ENERGY COMPARISON OF deBILS, GA, AND SS

Image	deBILS(mean±std)	GA(mean±std)	SS(mean±std)
1	**1.80E+05±5.47E+02**	3.29E+05±5.79E+03	1.99E+05±1.12E+04
2	**2.18E+05±3.77E+04**	8.56E+05±9.11E+03	3.39E+05±1.48E+05
3	1.60E+05±1.33E+03	3.49E+05±1.22E+02	**1.59E+05±3.76E+03**
4	1.87E+05±5.15E+03	5.27E+05±3.05E+03	**1.75E+05±1.27E+04**
5	1.84E+05±4.55E+03	5.10E+05±3.20E+02	**1.72E+05±1.20E+04**
6	2.88E+05±6.31E+04	5.55E+05±1.68E+04	**2.74E+05±4.57E+04**
7	**1.89E+05±4.40E+03**	3.14E+05±7.70E+03	2.15E+05±0.00E+00
8	2.55E+05±1.77E+04	1.02E+06±5.12E+03	**2.29E+05±9.10E+04**
9	**1.73E+05±7.02E+02**	4.74E+05±1.19E+03	1.74E+05±0.00E+00
10	**2.68E+05±1.93E+04**	5.61E+05±5.14E+03	3.43E+05±8.67E+04

TABLE VII
TIME COMPARISON OF deBILS, GA, AND SS

Image	deBILS(mean±std)	GA(mean±std)	SS(mean±std)
1	**3.15E+02±2.14E+01**	3.62E+03±4.96E-01	1.47E+04±2.34E+02
2	**1.50E+03±4.59E+01**	3.61E+03±1.68E+00	1.02E+04±6.78E+02
3	**5.13E+02±3.00E+01**	3.62E+03±6.13E+00	1.17E+04±3.46E+01
4	**9.83E+02±6.68E+01**	3.62E+03±6.24E+00	1.31E+04±1.26E+03
5	**1.01E+03±6.63E+01**	3.62E+03±3.44E+00	1.12E+04±1.06E+02
6	**1.53E+03±6.85E+01**	3.62E+03±2.86E-01	1.20E+04±1.54E+02
7	**8.98E+02±3.32E+01**	3.62E+03±1.07E+00	1.25E+04±6.53E+02
8	**1.21E+03±7.99E+01**	3.62E+03±2.27E+00	1.00E+04±8.54E+01
9	**1.24E+03±6.52E+01**	3.63E+03±6.57E+00	1.42E+04±3.60E+02
10	**1.41E+03±1.18E+02**	3.60E+03±1.07E+00	1.26E+04±8.38E+01

TABLE VIII
ENERGY FUNCTION PARAMETERS USED FOR MULTIPLE IMAGES

Test image	α	β	ω	ρ	ξ	δ	γ	σ
C1	1	3	3	1	10	15	5	0
C2	1	3	3	1	10	15	15	0
C3	1	3	3	1	10	15	5	0
C4	1	3	3	1	10	15	5	0
C5	1	1	3	1	1	5	10	0
C6	1	3	3	1	10	15	5	0
C7	1	1	3	1	1	5	10	0
C8	1	3	3	1	10	15	15	0
C9	1	1	3	1	1	5	10	0
C10	1	1	3	1	1	5	10	0

In our experiments, the GA was terminated after it consumed more 1 h for a single image. For the SS, the population was set to 8 to be time efficient as suggested by Bova *et al.* [8]. The SS procedure is terminated after two generations. Tables V–VII show the final results of the corresponding ERs, final energies, and absolute time consumed, respectively, for each algorithm. All results are based on ten independent runs. The results of the GA and SS that are significantly different from deBILS

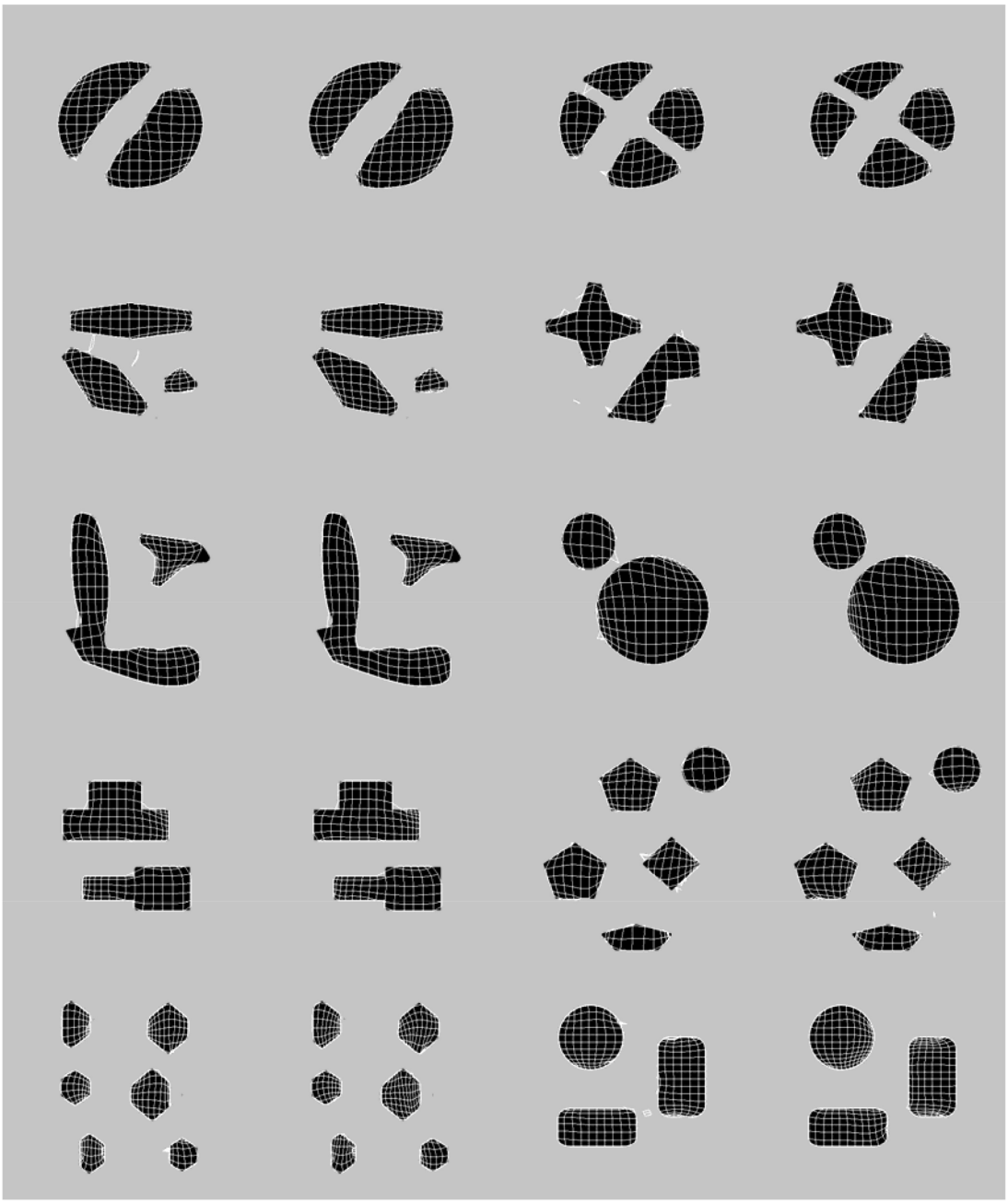

Fig. 13. TANs generated by BILS and deBILS on multiple target test images C1–C10 (first row: C1, C2, and so forth). For each pair of the same image, the left TAN is generated by BILS and the right by deBILS.

are highlighted in shade (with a Wilcoxon rank-sum test at significance level of 0.05) and the better results are marked in **boldface**.

From the results in Tables V and VI, we can see that deBILS is far better than GA and comparable with SS. Without BILS refinement, a standalone GA (even with some application-specific operators and a very long runtime) cannot optimize a TAN to a satisfactory degree, because the complex nature of a TAN makes it extremely inefficient to generate a good candidate by the coarse evolutionary operators. For the SS, we see two generations can achieve good enough results, compared with deBILS, especially the energy optimization results shown in Table VI. However, Table VII shows that SS consumed much more time than deBILS did even though SS ran only two generations. This indicates that the RS strategy used in deBILS does accelerate the search speed and enhance the global search ability.

Fig. 12 shows the curve of energy against CPU time for deBILS, GA, and SS on some selected tested images. The curves show the energy of the best TAN found by each algorithm during the running time. The figures clearly show that deBILS can achieve good quality in very short CPU time and terminates much earlier than both GA and SS. SS often consumes much more CPU time but can obtain better final

TABLE IX
SEARCH RESULTS COMPARISON WITH LINK CUTTING

Test Image	BILS			deBILS		
	Time	ER	Energy	Time(s) (reduced by)	ER	Energy
C1	1.02E+03	2.85%	1.73E+05	**9.25E+02 (9.28%)**	3.02%±0.15%	**1.66E+05±6.27E+03**
C2	1.40E+03	5.27%	2.09E+05	**1.16E+03 (16.69%)**	**4.91%±0.56%**	**2.07E+05±9.68E+03**
C3	1.08E+03	5.67%	2.14E+05	**9.11E+02 (15.96%)**	**4.59%±0.19%**	**1.55E+05±2.20E+04**
C4	2.61E+03	6.97%	2.55E+05	**1.92E+03 (26.31%)**	**6.69%±0.56%**	**1.99E+05±1.52E+04**
C5	2.35E+03	9.94%	1.09E+05	**1.42E+03 (39.54%)**	10.00%±0.28%	1.09E+05±8.05E+02
C6	1.86E+03	2.56%	1.90E+05	**1.39E+03 (25.11%)**	**1.88%±0.29%**	**1.83E+05±1.20E+04**
C7	1.55E+03	8.83%	8.66E+04	**1.04E+03 (33.16%)**	9.18%±0.15%	8.84E+04±2.70E+03
C8	5.37E+03	7.11%	3.57E+05	**3.73E+03 (30.57%)**	**6.20%±0.74%**	**2.93E+05±1.77E+04**
C9	3.38E+03	13.98%	9.23E+04	**2.60E+03 (23.04%)**	**13.91%±0.43%**	9.61E+04±8.83E+03
C10	2.68E+03	7.48%	1.20E+05	**2.06E+03 (23.25%)**	**7.02%±0.08%**	**9.81E+04±4.94E+03**

results than GA. Overall, compared with GA and SS, deBILS achieves a good balance on speed and quality.

D. Experiments on Multiple Targets Test Images

Here, we compare the TAN optimization efficiency of BILS and deBILS on test images with multiple targets. The link cutting method used in [8] is adopted in these multitarget tests. Both BILS and deBILS apply the same link cutting process and the resultant TANs are shown in Fig. 13. The search results contain some fragments that can be repaired with certain TAN reparation procedures as advocated in [8]. However, we omit the reparation procedure here, so as to focus on comparisons between the direct search results. Parameter settings in Table I are also used here. Table VIII shows the energy function parameter settings used in these tests. However, these parameters are slightly changed from the previous ones in Table II so as to let BILS work better on multiple targets. Table IX shows the optimization results of both algorithms on the resultant TAN energy and time costs in the search process. Results in Fig. 13 show that deBILS is able to generate better TANs than BILS does, with less miss-placed nodes and fragments. Results in Table IX show that the time consumed to find a good TAN by deBILS is reduced by up to nearly 40%.

V. CONCLUSION

In this paper, a new micro-DE-based optimization algorithm has been developed and applied to TAN optimization. If a standalone EA is used for TAN optimization, the solutions must be well refined after they are generated by evolutionary reproduction. This has been the main reason that a DS algorithm is used in existing TAN algorithms as the refinement method. For these problems, the existing DS methods are inefficient due to the lack of learning from historical information, as it is hard to use such information in a deterministic manner. We have therefore proposed the use of the micro-DE with a small population and incorporating historical information in a random and flexible manner for improved performance in speed and robustness. Interactions among individuals are seen useful in learning from historical information and hence help speed up optimization and enhance quality. This is especially significant when a region contains a large number of unpromising search points.

Test results have shown that the micro-DE-based deBILS approach developed in this paper is able to offer improved performance in both recognition speed and vision quality in most single and multiple target cases. For the future study, we believe that the deBILS framework can also be useful for other complex structural optimization problems in machine intelligence for time-consuming applications, especially where normal EAs are inefficient at generating good candidates or where it is hard for DS to learn from useful historical information.

REFERENCES

[1] M. Bro-Nielsen, "Active nets and cubes," Informat. Math. Modell., Tech. Univ. Denmark, Lyngby, Denmark, Tech. Rep. 13, 1994.
[2] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *J. Int. Comput. Vis.*, vol. 1, no. 4, pp. 321–331, 1987.
[3] F. M. Ansia, M. Penedo, C. Marino, and A. Mosquera, "A new approach to active nets," *Pattern Recognit. Image Anal.*, vol. 2, pp. 76–77, May 1999.
[4] J. H. Holland, *Adaptation Natural and Artificial Syst*. Ann Arbor, MI, USA: Univ. Michigan Press, 1975.
[5] O. Ibanez, N. Barreira, J. Santos, and M. G. Penedo, "Genetic approaches for topological active nets optimization," *Pattern Recognit*., vol. 42, no. 5, pp. 907–917, 2009.
[6] R. Storn and K. V. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, 1997.
[7] J. Novo, J. Santos, and M. Penedo, "Topological active models optimization with differential evolution," *Expert Syst. Appl.*, vol. 39, no. 15, pp. 12165–12176, 2012.
[8] N. Bova, O. Ibánez, and O. Cordón, "Image segmentation using extended topological active nets optimized by scatter search," *IEEE Comput. Intell. Mag.*, vol. 8, no. 1, pp. 16–32, Feb. 2013.
[9] N. Bova, O. Cordón, and O. Ibánez, "Extended topological active nets," Dept. Comput. Sci. Artif. Intel., Eur. Centre Soft Comput., Mieres, Spain, Tech. Rep. AFE 2012-01, 2012.
[10] T. Radulescu and V. Buzuloiu, "Extended vector field convolution snake for highly non-convex shapes segmentation," in *Proc. 9th Int. Conf. Inf. Technol. Appl. Biomed.*, Larnaca, Cyprus, 2009, pp. 1–4.
[11] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Trans. Evol. Comput*., vol. 15, no. 1, pp. 4–31, Feb. 2011.
[12] J. Q. Zhang and A. C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 945–958, Jun. 2009.
[13] 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.
[14] Y. Wang, Z. X. Cai, and Q. F. Zhang, "Differential evolution with composite trial vector generation strategies and control parameters," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 55–66, Feb. 2011.

[15] W. Y. Gong, Z. H. Cai, C. X. Ling, and H. Li, "Enhanced differential evolution with adaptive strategies for numerical optimization," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 2, pp. 397–413, Apr. 2011.
[16] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zümer, "Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems," *IEEE Trans. Evol. Comput.,* vol. 10, no. 6, pp. 646–657, Dec. 2006.
[17] S. Das, A. Abraham, and A. Konar, "Differential evolution using a neighborhood-based mutation operator," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 526–552, Jun. 2009.
[18] M. G. Epitropakis, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Enhancing differential evolution utilizing proximity-based mutation operators," *IEEE Trans. Evol. Comput.*, vol. 15, no. 1, pp. 99–119, Feb. 2011.
[19] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "An improved self-adaptive differential evolution algorithm for optimization problems," *IEEE Trans. Ind. Informat.,* vol. 9, no. 1, pp. 89–99, Feb. 2013.
[20] S. M. Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan, "An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization," *IEEE Trans. Syst. Man, Cybern. B, Cybern.,* vol. 42, no. 2, pp. 482–500, Apr. 2012.
[21] C. Li and S. Yang, "A general framework of multipopulation methods with clustering in undetectable dynamic environments," *IEEE Trans. Evol. Comput.*, vol. 16, no. 4, pp. 556–577, Aug. 2012.
[22] B. Y. Qu, P. N. Suganthan, and J. J. Liang, "Differential evolution with neighborhood mutation for multimodal optimization," *IEEE Trans. Evol. Comput.,* vol. 16, no. 5, pp. 601–614, Oct. 2012.
[23] S. Ghosh, S. Das, A. V. Vasilakos, and K. Suresh, "On convergence of differential evolution over a class of continuous functions with unique global optimum," *IEEE Trans. Syst. Man, Cybern. B, Cybern.,* vol. 42, no. 1, pp. 107–124, Feb. 2012.
[24] Y. Wang and Z. Cai, "Combining multiobjective optimization with differential evolution to solve constrained optimization problems," *IEEE Trans. Evol. Comput.*, vol. 16, no. 1, pp. 117–134, Feb. 2012.
[25] A. Basak, S. Das, and K. C. Tan, "Multimodal optimization using a biobjective differential evolution algorithm enhanced with mean distance-based selection," *IEEE Trans. Evol. Comput.*, vol. 17, no. 5, pp. 666–685, Oct. 2013.
[26] Y. Cai and J. Wang, "Differential evolution with neighborhood and direction information for numerical optimization," *IEEE Trans. Cybern.,* vol. 43, no. 6, pp. 2202–2215, Dec. 2013.
[27] Y. Fu, M. Ding, C. Zhou, and H. Hu, "Route planning for unmanned aerial vehicle (UAV) on the sea using hybrid differential evolution and quantum-behaved particle swarm optimization," *IEEE Trans. Syst., Man, Cybern., Syst.,* vol. 43, no. 6, pp. 1451–1465, Nov. 2013.
[28] W. Y. Gong and Z. H. Cai, "Differential evolution with ranking-based mutation operators," *IEEE Trans. Cybern.,* vol. 43, no. 6, pp. 2066–2081, Dec. 2013.
[29] U. Halder, S. Das, and D. Maity, "A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments," *IEEE Trans. Cybern.,* vol. 43, no. 3, pp. 881–897, Jun. 2013.
[30] J. H. Zhong *et al.*, "A differential evolution algorithm with dual populations for solving periodic railway timetable scheduling problem," *IEEE Trans. Evol. Comput.,* vol. 17, no. 4, pp. 512–527, Aug. 2013.
[31] L. X. Tan, Z. Yue, and L. Jiyin, "An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production," *IEEE Trans. Evol. Comput.,* vol. 18, no. 2, pp. 209–225, Apr. 2014.
[32] H. Wang, S. Rahnamayan, H. Sun, and M. G. H. Omran, "Gaussian bare-bones differential evolution," *IEEE Trans. Cybern.,* vol. 43, no. 2, pp. 634–647, Apr. 2013.
[33] Y. L. Li *et al.*, "Differential evolution with an evolution path: A DEEP evolutionary algorithm," *IEEE Trans. Cybern.*, DOI: 10.1109/TCYB.2014.2360752, 2014.
[34] W. J. Yu *et al.*, "Differential evolution with two-level parameter adaptation," *IEEE Trans. Cybern.*, vol. 44, no. 7, pp. 1080–1099, Jul. 2014.
[35] Q. Fan and X. Yao, "Self-adaptive differential evolution algorithm with zoning evolution of control parameters and adaptive mutation strategies," *IEEE Trans. Cybern.*, DOI: 10.1109/TCYB.2015.2399478, 2015.
[36] Z. H. Zhan and J. Zhang, "Self-adaptive differential evolution based on PSO learning strategy," in *Proc. Gen. Evol. Comput. Conf.*, Portland, OR, USA, 2010, pp. 39–46.
[37] Z. H. Zhan and J. Zhang, "Co-evolutionary differential evolution with dynamic population size and adaptive migration strategy," in *Proc. Gen. Evol. Comput. Conf.*, Dublin, Ireland, 2011, pp. 211–212.

Yuan-Long Li (S'11) received the B.S. degree in mathematics and the Ph.D. degree from Sun Yat-sen University, Guangzhou, China, in 2009 and 2014, respectively.

He is currently a Post-Doctorate with Nanyang Technology University, Singapore. His current research interests include global optimization, evolutionary algorithm, time series analysis, deep neural network, and their applications in wireless sensor network and data center modeling.

Zhi-Hui Zhan (S'09–M'13) received the bachelor's and Ph.D. degrees from the Department of Computer Science, Sun Yat-sen University, Guangzhou, China, in 2007 and 2013, respectively.

He is currently an Associate Professor with the School of Advanced Computing, Sun Yat-sen University. His current research interests include evolutionary computation algorithms, swarm intelligence algorithms, and their applications in real-world problems, and in environments of cloud computing and big data.

Dr. Zhan was a recipient of the China Computer Federation Outstanding Dissertation in 2013, for his work in doctoral dissertation, the Natural Science Foundation for Distinguished Young Scientists of Guangdong Province, China, in 2014, and the Pearl River New Star in Science and Technology in 2015. He is listed as one of the Most Cited Chinese Researchers in Computer Science.

Yue-Jiao Gong (S'10–M'15) received the B.S. and Ph.D. degrees in computer science from Sun Yat-sen University, Guangzhou, China, in 2010 and 2014, respectively.

She is currently a Post-Doctoral Research Fellow with the Department of Computer and Information Science, University of Macau, Macau, China. Her current research interests include computational intelligence, evolutionary computation, swarm intelligence, and their applications in design and optimization of intelligent transportation systems, wireless sensor networks, and Radio Frequency Identification systems.

Jun Zhang (M'02–SM'08) received the Ph.D. degree in electrical engineering from the City University of Hong Kong, Hong Kong, in 2002.

He is currently a Changjiang Chair Professor with the School of Advanced Computing, Sun Yat-sen University, Guangzhou, China. He has published over 100 technical papers in the above areas. His current research interests include computational intelligence, cloud computing, high-performance computing, data mining, wireless sensor networks, operations research, and power electronic circuits.

Dr. Zhang was a recipient of the China National Funds for Distinguished Young Scientists from the National Natural Science Foundation of China in 2011 and the First-Grade Award in Natural Science Research from the Ministry of Education, China, in 2009. He is currently an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, and the IEEE TRANSACTIONS ON CYBERNETICS.

Yun Li (S'87–M'90) received the B.S. degree in electronics science from Sichuan University, Chengdu, China, in 1984, the M.Eng. degree in electronic engineering from the University of Electronic Science and Technology of China, Chengdu, in 1987, and the Ph.D. degree in computing and control from the University of Strathclyde, Glasgow, U.K., in 1990.

From 1989 to 1990, he was with the U.K. National Engineering Laboratory and Industrial Systems and Control Ltd, Glasgow, G12 8QQ, U.K. He joined the University of Glasgow, Glasgow, as a Lecturer, in 1991, where he is currently a Professor of Systems Engineering. He served as the Founding Director with the University of Glasgow Singapore, Singapore, from 2011 to 2013, and acted as the Founding Director of the University's International Joint Program with the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2013. He was invited to Kumamoto University, Kumamoto, Japan, as a Visiting Professor in 2002. He is currently a Visiting Professor with UESTC and Sun Yat-sen University, Guangzhou, China, researching into smart design with market informatics via the cloud to complete the value chain for Industry 4.0. He has written the popular online courseware "GA demo" for interactive evolutionary computation in 1997 and has 200 publications.

Dr. Li is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and the *SM Journal of Engineering Sciences*. He established the IEEE Control System Society and the European Network of Excellence Workgroup on Computer-Automated Design in 1998. He is a Chartered Engineer.

Qing Li (SM'07) received the B.Eng. degree from Hunan University, Changsha, China, and the M.Sc. and Ph.D. degrees from the University of Southern California, Los Angeles, CA, USA, all in computer science.

He is the Founding Director of the Multimedia Software Engineering Research Centre, and concurrently a Professor with the Department of Computer Science, City University of Hong Kong, Hong Kong. His current research interests include dynamic object modeling, multimedia and mobile information retrieval and management, distributed databases and data warehousing/mining, and workflow management and Web services.