Soft Computing
https://doi.org/10.1007/s00500-019-04165-2

METHODOLOGIES AND APPLICATION

A unified algorithm based on HTS and self-adapting PSO for the construction of octagonal and rectilinear SMT

Genggeng Liu[1,2,3] · Zhisheng Chen[1] · Zhen Zhuang[1] · Wenzhong Guo[1,2,3] · Guolong Chen[1]

Abstract

The Steiner minimal tree (SMT) problem is an NP-hard problem, which is the best connection model for a multi-terminal net in global routing problem. This paper presents a unified algorithm for octagonal and rectilinear SMT construction based on hybrid transformation strategy (HTS) and self-adapting particle swarm optimization. Firstly, an effective HTS is proposed to enlarge the search space and improve the convergence speed. Secondly, the proposed HTS in the evolutionary process may produce an ineffective solution, and consequently the crossover and mutation operators of genetic algorithm (GA) based on union-find sets is proposed. Thirdly, a self-adapting strategy that can adjust the acceleration coefficients is proposed to further improve the convergence and the quality of the proposed algorithm. Finally, the hybrid transformation can be applied to GA and the proposed algorithm can be applied to rectilinear architecture. To our best knowledge, the proposed algorithm is the first unified algorithm to solve the SMT construction under both octagonal and rectilinear architecture. The experimental results show that the proposed algorithm can efficiently provide a better solution for SMT problem both in octagonal and rectilinear architectures than others. Moreover, the algorithm can obtain several topologies of SMT, which is beneficial for optimizing congestion in VLSI global routing stage.

Keywords Hybrid transformation strategy · Particle swarm optimization · Self-adapting strategy · Unified algorithm · Octagonal steiner minimal tree · Rectilinear steiner minimal tree

1 Introduction

Global routing is a very important step in very large scale integration (VLSI) physical design. The Steiner minimal tree (SMT) is the best connection model for the multi-terminal net in VLSI global routing, and thus, SMT construction is also very important (Xu and Hong 2009). The SMT problem is to find a minimum cost routing tree that connects a given pin set by introducing additional points (Steiner points).

The research on global routing is mainly focused on the construction of the routing tree and the design of the global routing algorithm in rectilinear architecture, and some good research has been done (Han et al. 2014; Scheifele 2016, Siddiqi and Sait 2017, Held et al. 2017). In particular, there are some competitions held on ISPD and then NCTU-R (Dai et al. 2009), NTHU-Route2.0 (Chang et al. 2010), NCTU-GR (Dai et al. 2012), NCTU-GR2.0 (Liu et al. 2013) and other global routers have emerged.

So far, most routing algorithms are proposed for rectilinear architecture. However, the routing model based on rectilinear architecture requires that the connections between the pins

Communicated by V. Loia.

✉ Guolong Chen
fzucgl@163.com

Genggeng Liu
liu_genggeng@126.com

Zhisheng Chen
fzu_laughing@qq.com

Zhen Zhuang
zhuang_zhen@126.com

Wenzhong Guo
fzugwz@163.com

1 College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350116, China

2 Fujian Provincial Key Laboratory of Network Computing and Intelligent Information Processing, Fuzhou University, Fuzhou 350116, China

3 Key Laboratory of Spatial Data Mining and Information Sharing, Ministry of Education, Fuzhou 350116, China

Published online: 22 June 2019

can only be horizontal and vertical, resulting in limited ability to optimize the interconnect length. As a result, more and more researchers are interested in the non-Manhattan architecture which can make full use of routing resources and achieve shorter interconnect length.

In order to better study the routing problem based on non-Manhattan architecture, the first work is to study the construction of SMT in non-Manhattan architecture. The branch and bound method was proposed to construct a hexagonal SMT, but is only suitable for small-scale problems (Thurber and Xue 1999). Coulston (2003) used the accurate algorithm and a series of pruning strategies to construct the octagonal SMT. Compared with rectilinear Steiner minimal tree (RSMT), it can achieve 1.16% shorter wirelength with more time. In view of the high time complexity of exact algorithms, some heuristic strategies are adopted to construct the non-Manhattan architecture Steiner trees. An algorithm with the time complexity $O(|V|+|E|)$ was proposed to construct the octagonal Steiner tree (Chiang and Chiang 2002), but the constructed tree must be isomorphic. A heuristic algorithm with the time complexity $O(n^3 \log n)$ was proposed to solve the hexagonal Steiner tree problem (Samanta et al. 2006), which is based on greedy algorithm. Two algorithms based on octagonal spanning graph were proposed for the construction of octagonal Steiner minimal tree (OSMT) (Zhu et al. 2005). The first one used an edge replacement technology, and the other one used the triangle contraction method. Both methods are based on greedy algorithms. A modified algorithm is proposed to construct a time-driven octagonal Steiner tree. The delay of the longest path is effectively reduced by adjusting the longest path of the hexagonal routing (Yan 2008). Based on Hanan grid and Elmore delay model, Samanta et al. (2011) constructed the timing driven Steiner tree and verified that the Elmore delay model has higher calculation accuracy than the linear delay model. Based on graph theory, k-IDEA algorithm was used to generate multiple sets of routing solutions (Koh and Madden 2000). The shorter wirelength is obtained, but excessive vias are generated.

Constructing the non-Manhattan SMT is an NP-hard problem (Garey and Johnson 1977). The above researches on non-Manhattan SMT are all based on exact algorithm and traditional heuristic algorithm. However, the time complexity of exact algorithm increases exponentially with the extension of problem scale. Most of the traditional heuristic algorithms are based on greedy strategy and easy to fall into the local optimum. The above two methods did not make full use of the geometric properties of non-Manhattan architecture and cannot guarantee the quality of SMT. These methods provided less suitable method for the topology optimization, and therefore, it is not satisfactory in terms of time efficiency, wirelength and so on.

The SMT problem is an NP-hard problem, so some evolutionary algorithms which have shown good application prospect in solving NP-hard problems were used to solve RSMT (Arora and Mose 2009; Liu et al. 2011, Abhinandan et al. 2013, Bhattacharya et al. 2014; Manna et al. 2015; Kundu et al. 2016, 2017) and OSMT problem (Arora and Mose 2009; Liu et al. 2012). As a swarm-based evolutionary method, particle swarm optimization (PSO) was proposed by Eberhar and Kennedy (1995). PSO algorithm has many advantages such as global search and stable (Xue et al. 2013; Agrawal and Silakari 2014; Rada-Vilela et al. 2013; Wang et al. 2017). In recent years, PSO algorithm has been widely and successfully applied in many fields and proved to be a powerful optimization tool by means of continuous simulation and theoretical analysis (Ruiz-Cruz et al. 2013; Liu et al. 2015a, b, c; Xu et al. 2016; Huang et al. 2017).

Therefore, an efficient unified algorithm based on hybrid transformation strategy (HTS) and PSO, namely HTS-PSO, is proposed to solve the construction of OSMT and RSMT problem. This paper makes the following contributions.

- To our best knowledge, HTS-PSO is the first unified algorithm to solve the SMT construction under both octagonal and rectilinear architecture. And it can achieve the best solution. With less runtime,our algorithm reduces 8.65%, 9.93%, 0.83%, and 1.22% than recently published methods (Kundu et al. 2017, 2016; Manna et al. 2015; Bhattacharya et al. 2014) on wirelength, respectively. Moreover, the algorithm can obtain several topologies of SMTs, which is beneficial for optimizing congestion in VLSI global routing stage.
- An efficient HTS is proposed to enlarge the search space and accelerate the convergence of the proposed algorithm. Meanwhile, HTS can be applied to genetic algorithm (GA) and the proposed algorithm can also be applied to rectilinear architecture. Therefore, the proposed HTS has high versatility.
- For the acceleration coefficients, an effective self-adapting adjustment strategy is adopted to achieve fast convergence rate. Meanwhile, the crossover and mutation operators of genetic algorithm based on union-find sets is proposed to o avoid invalid solution. Moreover, when the proposed algorithm is used to solve large-scale problems, its optimization ability is becoming stronger. Therefore, the proposed algorithm is more advantageous and suitable for solving VLSI routing problems.

The rest of the paper is organized as follows. In Sect. 2, the preliminaries are presented. Section 3 introduces the hybrid transformation strategies. In Sect. 4, the corresponding implementation details and the global convergence proof of HT-PSO are described. To prove the good performance of the proposed HTS-PSO, several comparisons are presented in Sect. 5. Section 6 concludes this paper.

Table 1 The input information for the pins of net

Number	1	2	3	4	5	6	7	8
X-coordinate	33	2	42	47	34	38	37	20
Y-coordinate	33	9	35	2	1	2	5	4

2 Preliminaries

2.1 Problem Formulation

Definition 1 In the λ-geometry, the routing direction is $i\pi/\lambda$, where i is an arbitrary integer and λ is an integer. Different routing directions are obtained by different values of i and λ.

(1) Rectilinear architecture: The value of λ is set to 2, i.e., the routing direction is $i\pi/2$, which includes 0° and 90°, namely horizontal and vertical orientations. Rectilinear architecture belongs to Manhattan architecture.
(2) Octagonal architecture: The value of λ is set to 4, i.e., the routing direction is $i\pi/4$, which includes 0°, 45°, 90° and 135°. Octagonal architecture belongs to non-Manhattan architecture.

The SMT problem based on octagonal architecture is more complicated and challenging than the one based on rectilinear architecture. Therefore, we first study the efficient algorithm for the OSMT problem and then extend it to the RSMT problem.

OSMT problem: Given a set of n pins $P = \{P_1, P_2, P_3, \ldots, P_m\}$ and each P_i represented by a coordinate of (x_i, y_i), construct an OSMT to connect the pins in P through some Steiner points, where the direction of routing path can be 45° and 135°, in addition to the traditional horizontal and vertical directions. For example, there is a routing net with 8 pins and the input information of the pins is given in Table 1. The layout distribution of the given pins is shown in Fig. 1. The coordinate pairs of pin 1 is (33,33), as shown in Column 2 of Table 1.

2.2 Definitions

Definition 2 (*Pseudo-Steiner point*) For convenience, we assume that the connection point except the pins is called the pseudo-Steiner point (PSP). In Fig. 2, the point S is PSP, and PSP contains the Steiner point.

Definition 3 (*Choice 0*): In Fig. 2a, let $A = (x_1, y_1)$ and $B = (x_2, y_2)$ be the two endpoints of line segment L, where $x_1 < x_2$. The Choice 0 of L is shown in Fig. 2b, which leads rectilinear edge from A to S firstly, and then leads octagonal edge from S to B.

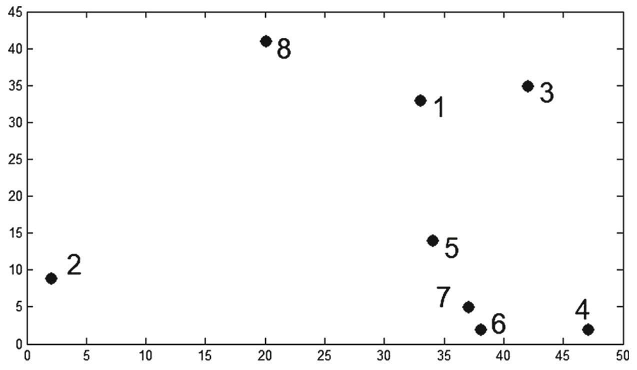

Fig. 1 The layout distribution of the given pins in Table 1

Definition 4 (*Choice 1*): The Choice 1 of L is shown in Fig. 2b, which leads octagonal edge from A to S firstly, and then leads rectilinear edge from S to B.

Definition 5 (*Choice 2*): The Choice 2 of L is shown in Fig. 2c, which leads vertical edge from A to S firstly, and then leads horizontal edge from S to B.

Definition 6 (*Choice 3*): The Choice 3 of L is shown in Fig. 2d, which leads horizontal edge from A to S firstly, and then leads vertical edge from S to B.

2.3 Basic PSO Algorithm

PSO is a swarm intelligence method, which considers that a swarm contain p particles in a D-dimensional continuous solution space. Each ith particle has its own position and velocity. Assuming that the search space is D-dimensional, the position of the ith particle is denoted as a D-dimensional vector: $X_i = (X_{i1}, X_{i2}, \ldots, X_{iD})$ and the best particle in the swarm is denoted as P_g. The best previous position of the ith particle is recorded and represented as $P_i = (P_{i1}, P_{i2}, \ldots, P_{iD})$, while the velocity for the ith particle can be defined by another D-dimensional vector: $V_i = (V_{i1}, V_{i2}, \ldots, V_{iD})$. According to these definitions, the particle position and velocity can be calculated according to the following equations.

$$V_i^{t+1} = w \times V_i^t + c_1 r_1 (P_i^t - X_i^t) + c_2 r_2 (P_g^t - X_i^t) \qquad (1)$$

$$X_i^{t+1} = X_i^t + V_i^{t+1} \qquad (2)$$

where w is an inertia weight. c_1 and c_2 are acceleration coefficients. r_1 and r_2 are both random numbers on the interval [0, 1).

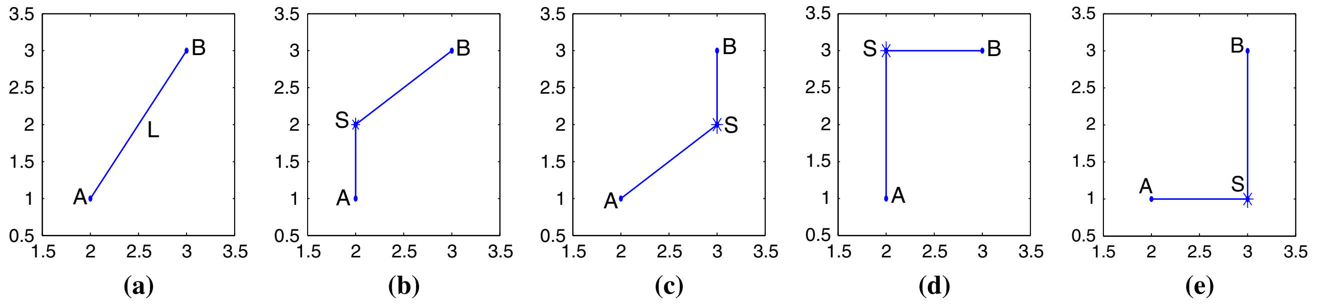

Fig. 2 Four choices of PSP for the given line segment. **a** Line segment *L*, **b** Choice 0, **c** Choice 1, **d** Choice 2, **e** Choice 3

3 Hybrid transformation strategy

3.1 PSP transformation

For the OSMT construction (Liu et al. 2012), it used PSP transformation strategy and each PSP selection method contains four choices, namely Choice 0, Choice 1, Choice 2 and Choice 3, respectively. Based on PSP transformation strategy, the proposed algorithm takes the wirelength as the optimization target and achieves short wirelength than RSMT, and the average wirelength can be reduced by 9.68% in Table 2.

Property 1 Only with PSP transformation strategy based on minimum spanning tree (MST), the optimization ability of OSMT construction algorithm is limited.

If the algorithm uses only PSP transformation strategy, then it is a method based on MST to construct SMT. According to Ma et al. (2000), for the rectilinear architecture, the wirelength of SMT based on MST construction may be 1.5 times than exact solution and has a certain distance from the optimal solution to the accurate RSMT. As shown in (3), cost(MST) represents the length of the SMT based on MST and cost(RSMT) represents the exact solution of RSMT. For the octagonal routing problem with more routing direction, the maximum distance will be greater.

$$\mathrm{cost(MST)}/\mathrm{cost(RSMT)} < 3/2 \tag{3}$$

Therefore, for constructing SMT, only PSP transformation strategy is executed, the solution space does not necessarily contain the optimal solution and even 0.5 times farther from the optimal solution. Therefore, we need to redesign the transformation strategy. Then another basic transformation strategy, namely edge transformation (‘E transformation’ represents ‘edge transformation’ in this paper), will be introduced in next subsection.

3.2 E transformation

Property 2 The introduction of E transformation enlarges the search space of the algorithm, and it has the opportunity to obtain better routing solution than PSP transformation. Even in some cases, it can help the algorithm to find the optimal solution.

Table 2 Comparison between PSP transformation and RSMT

Circuit	Pin#	RSMT	PSP	Imp(%)/ RSMT
1	8	17,928	16,918	5.64
2	9	20,478	18,041	11.90
3	10	21,969	19,696	10.35
4	20	35,675	32,207	9.72
5	50	53,518	48,020	10.27
6	70	62,633	56,433	9.90
7	100	75,584	68,855	8.90
8	410	158,206	141,894	10.31
9	500	170,924	154,825	9.42
10	1000	246,731	221,090	10.39
Average				9.68

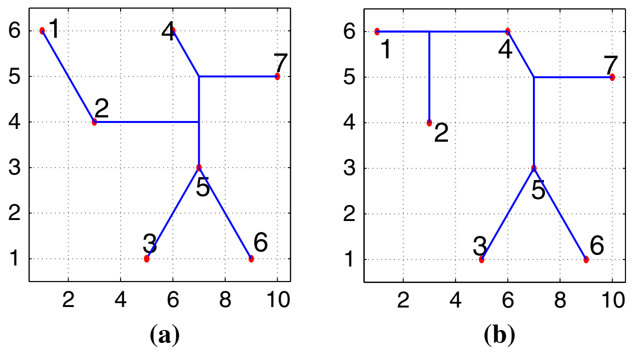

Fig. 3 The necessity of E transformation in the evolutionary process. **a** Tree topology A, **b** Tree topology B

Property 3 With the introduction of E transformation, the search space of our proposed algorithm contains the exact solution for rectilinear routing.

The update operation of PSO for constructing OSMT (Liu et al. 2012) only considers the PSP transformation. However, in the process of constructing RSMT or OSMT, the

Table 3 Comparison among E transformation, PSP transformation and RSMT

TEST	RSMT	PSP	E	Imp/RSMT(%)	Imp/PSP (%)
1	17,928	16,918	16,921	5.62	– 0.02
2	20,478	18,041	18,023	11.99	0.10
3	21,969	19,696	19,397	11.71	1.52
4	35,675	32,207	32,163	9.85	0.14
5	53,518	48,020	48,027	10.26	– 0.02
6	62,633	56,433	56,551	9.71	– 0.21
7	75,584	68,855	68,991	8.72	– 0.20
8	158,206	141,894	141,443	10.60	0.32
9	170,924	154,825	155,820	8.84	– 0.64
10	246,731	221,090	221,324	10.30	– 0.11
Average				9.76	0.09

topology of the routing tree is variable. For example, in Fig. 3, in the evolutionary process of constructing OSMT, there are two different topologies. The edge (4,2) exists in the routing tree shown in Fig. 3b, but does not exist in the routing tree in Fig. 3a. In Fig. 3, the transformation of the two edges is not possible only through PSP transformation strategy. Therefore, in the update operation of the RSMT or OSMT construction algorithm, E transformation strategy is needed in addition to PSP transformation strategy. Moreover, E transformation strategy can further enhance the wirelength optimization ability. In Table 3, E transformation can reduce the wirelength by 0.09% compared with PSP transformation. Meanwhile, E transformation is introduced into the update operation and it can ensure that the search space of the proposed algorithm (Liu et al. 2011) contains the optimal solution. And compared with the RSMT construction algorithm (Liu et al. 2011), the average wirelength reduction of 9.76% is obtained.

3.3 Combinations of HTS

Considering the respective advantages of PSP transformation and E transformation, we assume that these two transformations can be combined with different combinations to find the optimal HTS. In Table 4, we list all the combinations of HTS, including PSP:PSP, E:E, (PSP+E):(PSP+E), PSP:(PSP+E), (PSP+E):E, E:(PSP+E), (PSP+E):PSP, E:PSP, PSP:E. We divide the iterative process of the algorithm into the first half and the second half. In Table 4, for combinations CM1, CM2 and CM3, the first half and the second half are using the same transformation strategy. CM1 is PSP transformation, CM2 is E transformation and CM3 is the basic hybrid strategy that will be described later.

3.3.1 Combination of PSP and E transformation

Property 4 The hybrid strategy with PSP and E transformation can further enlarge the search space. However, it may have smaller convergence rate and even in some cases it cannot converge to better solution than PSP or E transformation.

Table 4 The different combinations of HTS

Number	First half	Second half	Abbreviated way
1	PSP	PSP	CM1
2	E	E	CM2
3	PSP+E	PSP +E	CM3
4	PSP	PSP+E	CM4
5	PSP +E	E	CM5
6	E	PSP+E	CM6
7	PSP+E	PSP	CM7
8	E	PSP	CM8
9	PSP	E	CM9

The direct hybrid idea is that both PSP and E transformations are combined during the whole iterative process. We apply this method to the previous algorithm, and the results are obtained in Table 5. From Table 5, it can be seen that part of the circuits, HTS based on PSP and E transformation (it is represented by 'PSP+E', known as 'CM3') has the larger search space than PSP transformation. However, due to the large search space of CM3, in some circuits, its optimization ability is weaker than PSP or E transformation. In Table 5, for Circuit 1,5,6,7,8,9,10, CM3 is worse than PSP transformation, and for Circuit 5,7,8,10, CM3 is worse than E transformation. Finally, the SMT algorithm based on CM3 is compared with the RSMT algorithm. In Table 6, CM3 can achieve the 9.67% average improvements over RSMT algorithm on wirelength.

3.3.2 The hybrid transformation strategy CM8

Property 5 E transformation strategy can enlarge the search space, and PSP transformation can accelerate convergence in the later iteration.

Table 5 Comparison among CM3, PSP and E transformation

Circuit	PSP	E	CM3	Imp/PSP (%)	Imp/E (%)
1	16,918	16,921	16,922	– 0.02	0.00
2	18,041	18,023	18,023	0.10	0.00
3	19,696	19,397	19,397	1.52	0.00
4	32,207	32,163	32,112	0.29	0.16
5	48,020	48,027	48,141	– 0.25	– 0.24
6	56,433	56,551	56,498	– 0.11	0.09
7	68,855	68,991	69,334	– 0.70	– 0.50
8	141,894	141,443	142,282	– 0.27	– 0.59
9	154,825	155,820	155,392	– 0.37	0.27
10	221,090	221,324	221,612	– 0.24	– 0.13
Average				– 0.01	– 0.09

Table 6 Comparison between CM3 and RSMT

Circuit	RSMT	CM3	Imp/RSMT
1	17,928	16,922	5.61
2	20,478	18,023	11.99
3	21,969	19,397	11.71
4	35,675	32,112	9.99
5	53,518	48,141	10.05
6	62,633	56,498	9.80
7	75,584	69,334	8.27
8	158,206	142,282	10.07
9	170,924	155,392	9.09
10	246,731	221,612	10.18
Average			9.67

E:PSP hybrid transformation strategy, known as CM8 which refers to E transformation, is adopted in the early stage of the algorithm, and the latter only PSP transformation is adopted. In Table 7, the hybrid transformation strategy CM8 can achieve the 0.25% average improvements over PSP transformation on wirelength. In Table 7, CM8 can achieve the 9.91% average improvements over the RSMT construction algorithm on wirelength.

3.3.3 The hybrid transformation strategy CM7

Property 6 PSP+E can further enlarge the search space in the early iteration and then find a better topology. Meanwhile, based on a better topology, the PSP can speed up the convergence to obtain a better solution in the later iteration.

CM7 is that E and PSP transformation are combined in the early stage of the algorithm, and later only PSP transformation is adopted. CM7 can achieve the 0.37% average improvements over PSP transformation on wirelength in Table 8. So the optimization effect of CM7 with PSP+E is better than CM8 with E. In Table 8, CM7 can achieve the 10.01% average improvements over the RSMT construction algorithm on wirelength.

For CM7, in the early stage of iterative algorithm, to a certain extent, PSP+E can expand the more search space than CM8. And later using PSP transformation helps the algorithm to converge to a better solution in the richer search space.

3.3.4 Other hybrid transformation strategies

In Table 9, we list all combinations of HTS, from CM1 to CM9 and then test the wirelength optimization ability of these hybrid transformation strategies. Finally, we choose the best hybrid transformation strategy CM7 as an efficient HTS. We combine the best HTS with PSO, and then, an efficient algorithm based on the best HTS and self-adapting PSO, namely HTS-PSO, is proposed to solve the OSMT problem more efficiently.

4 Details of HTS-PSO algorithm

As (1) and (2) mentioned in Sect. 2.3, it is obvious that the basic PSO cannot obtain a discrete solution for the OSMT problem as a result of its continuous nature. It has guided many discrete PSO algorithms to solve discrete problems (Liu et al. 2010; Costas et al. 2012; Liu et al. 2015a, b, c). In our previous works, the importance of VLSI physical design and the advantages of PSO have made many research in partitioning, floorplanning, and routing (Chen et al. 2010; Guo et al. 2014; Liu et al. 2015a, b, c; Huang et al. 2017). For the OSMT problem, an effective algorithm based on HTS and self-adapting PSO are presented in this paper.

Table 7 Comparison among CM8, PSP and E transformation

Circuit	RSMT	PSP	CM8	Imp/RSMT (%)	Imp/PSP (%)
1	17,928	16,918	16,923	5.61	– 0.03
2	20,478	18,041	18,023	11.99	0.10
3	21,969	19,696	19,397	11.71	1.52
4	35,675	32,207	32,099	10.02	0.33
5	53,518	48,020	48,090	10.14	– 0.15
6	62,633	56,433	56,352	10.03	0.14
7	75,584	68,855	68,915	8.82	– 0.09
8	158,206	141,894	141,497	10.56	0.28
9	170,924	154,825	154,410	9.66	0.27
10	246,731	221,090	220,725	10.54	0.17
Average				9.91	0.25

Table 8 Comparison among CM7, PSP and E transformation

Circuit	RSMT	PSP	CM7	Imp/RSMT (%)	Imp/PSP (%)
1	17,928	16,918	16,921	5.62	– 0.02
2	20,478	18,041	18,023	11.99	0.10
3	21,969	19,696	19,397	11.71	1.52
4	35,675	32,207	32,073	10.10	0.41
5	53,518	48,020	48,058	10.20	– 0.08
6	62,633	56,433	56,326	10.07	0.19
7	75,584	68,855	68,790	8.99	0.09
8	158,206	141,894	141,124	10.80	0.54
9	170,924	154,825	154,001	9.90	0.53
10	246,731	221,090	220,266	10.73	0.37
Average				10.01	0.37

4.1 Encoding strategy

The encoding strategy of a spanning tree usually includes two ways: Prufer number encoding (Gottlieb et al. 2001) and edge-vertex encoding.

Definition 7 (*Puffer number*) The Puffer number encoding gives a one-one mapping between the spanning tree of the N nodes and the string of numbers of $n - 2$. It is customarily used to mark these nodes with numbers. The resulting string is called the Puffer number.

Property 7 An edge-vertex encoding strategy is more suitable for evolutionary algorithms than Prufer number. Furthermore, an efficient decoding of Prufer number has the time cost that is $O(n \log n)$, while edge-vertex encoding strategy does not have to spend time decoding it.

The length of the Puffer number encoding is 1/3 of the length of the edge-vertex encoding. However, since one digit of the Puffer number encoding is changed, the topology of the entire spanning tree is greatly changed. And thus the optimal topological information of the particle cannot be well retained. Therefore, edge-vertex encoding is more suitable for the PSO algorithm and retains the optimal topological information of particles during the iterative process. Meanwhile, changing one digit of the edge-vertex encoding does not have a huge impact on the topology. Therefore, the proposed HTS-PSO algorithm adopts the edge-vertex encoding to encode a particle, which is more suitable for PSO, GA and other evolutionary algorithms.

Table 9 Comparison between various combinations of HTS and RSMT

Number	First half	Second half	Imp/RSMT (%)
CM1	PSP	PSP	9.28
CM2	E	E	9.76
CM3	PSP+E	PSP+E	9.67
CM4	PSP	PSP+E	9.54
CM5	PSP+E	E	9.40
CM6	E	PSP+E	9.36
CM7	PSP+E	PSP	10.01
CM8	E	PSP	9.91
CM9	PSP	E	9.84

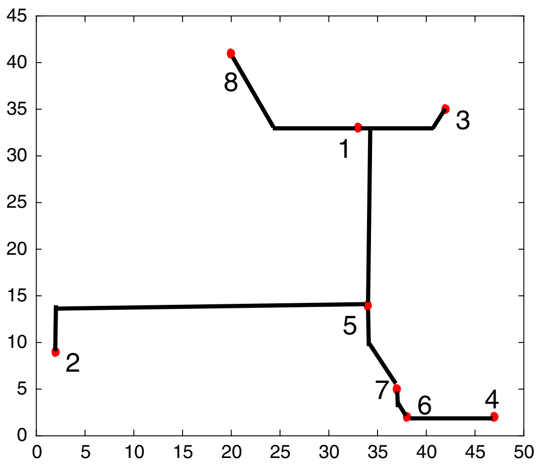

Fig. 4 One routing tree with eight pins

One candidate Steiner tree is represented as lists of spanning tree edges and the choice of pseudo-Steiner point which specifies the transformation from the spanning tree edge to the octagonal edge. Each choice of pseudo-Steiner point, includes four types in Definitions 3–6 and the value is 0, 1, 2 or 3 which denotes Choice 0, Choice 1, Choice 2 or Choice 3, respectively.

If a net has n pins, a spanning tree would have n-1 edges and one extra digit which is the fitness of particle. For each three digits, the first two digits represent one edge that include two vertices and the last digit indicates the choice of pseudo-Steiner point. Thus the length of one particle is $3\times(n-1)+1$. For example, one routing tree ($n = 8$) shown in Fig. 4 can be expressed as one particle whose encoding can be represented as the following numeric string.

7 6 *0* 6 4 *1* 7 5 *1* 5 1 2 1 3 *0* 1 8 *1* 5 2 2 **74.63**

where the length of the particle is $3 \times (8-1)+1 = 22$, the bold number 74.63 is the fitness of the particle and each italic number represents the choice of PSP for each edge. The coordinates of all pins are derived from Table 1. From Fig. 4, we can calculate the total length of the tree is 133.04 and then the fitness is 74.63 which is calculated according to (5). The first substring (7,6,*0*) represents one edge of the spanning tree in Fig. 4 which composed of Pin 7 and Pin 6 with Choice 0.

4.2 The Fitness Function

Definition 8 The length of the Octilinear Steiner tree is the sum of the lengths of all the edge segments in the tree, which is calculated as follows.

$$L(T_x) = \sum_{e_i \in T_x} l(e_i) \tag{4}$$

where $l(e_i)$ represents the length of each segment e_i in the tree T_x.

When calculating the sum of the length of each edge segment in octagonal Steiner tree, all the edge segments are divided into the following four types: horizontal edge segments, vertical edge segments, 45° edge segments, and 135° edge segments. First of all, the algorithm divides all the edge segment into the above four types. Then rotates the 45° edge clockwise direction to be the horizontal edge, and simultaneously rotates the 135° edge clockwise direction to be the vertical edge. The horizontal edge is arranged from the bottom to top and then from left to right according to the coordinate of the left pin. Simultaneously arrange the vertical edge from left to right and then from bottom to top according to the coordinate of the lower pin. Finally, the length of octagonal Steiner tree is the sum of the total length of these edges excluding overlap segments.

The smaller the fitness value, the better the particle is represented. Thus the particle fitness function is designed as follows.

$$fitness = \frac{1}{L(T_x)+1} \times 10{,}000 \tag{5}$$

4.3 Update formula of the particle

The OSMT problem is the discrete optimization problem. Therefore, the update formula of basic PSO for particles can be no longer suitable for solving the OSMT problem. For this reason, the crossover and mutation operators combined with union-find partition are designed to construct the discrete update formula of particles for HTS-PSO algorithm. The redesigned particle update formula is defined as follows.

$$X_i^t = N_3(N_2(N_1(X_i^{t-1}, w), c_1), c_2) \tag{6}$$

where w is an inertia weight, c_1 and c_2 are acceleration constants. N_1 denotes the mutation operation and N_2, N_3 denote the crossover operations.

The introduction of E transformation further optimizes the wirelength of the Steiner tree. However, the introduction of E transformation may lead to the appearance of loop and generate the invalid solution in the iterative process. For this reason, the union-find sets which keeps track of the connected components is integrated into the following update operators.

The velocity of particles can be written as follows.

$$W_i^t = N_1(X_i^{t-1}, w) = \begin{cases} M(X_i^{t-1}), & r_1 < w \\ X_i^{t-1}, & \text{others} \end{cases} \tag{7}$$

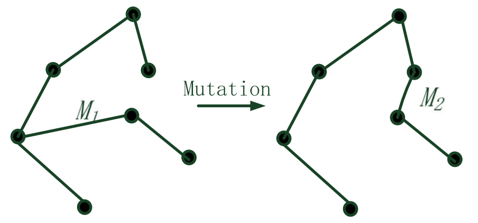

Fig. 5 Mutation operation with the Union-Find partition

Algorithm 1 Mutation operator(p)

```
Input: Particle $p$
Output: New particle
  $Initialize\ each\ pin's\ partition\ to\ singletons$
  $r = random(1, n-1)$; //$n$ is the number of pins
  for each edge $e_i$ of $p$ do
    if $e_i \neq e_r$ then
      $Union\_partition(u, v)$;//$u$ and $v$ is endpoint of $e_i$, $u$ and $v$ are merged into the same set
    end if
  end for
  while $true$ do
    $p_1 = random(1, n-1)$;// generate a random number between (1,$n$-1)
    $p_2 = random(1, n-1)$;
    if $Find\_set(p_1) \neq Find\_set(p_2)$ then
      $Union\_partition(p_1, p_2)$;//$p_1$ and $p_2$ are not in the same set
      $generate\_edge(p_1, p_2)$;
      $break$;
    end if
  end while
```

where w denotes the mutation probability. r_1 is a random number on the interval [0, 1).

Mutation operator randomly deletes an edge from the spanning tree and replaces it with another randomly generated edge. In order to make sure that the spanning tree is connected, we use a union-find set to record all the rest of points as two sets after one edge is deleted and randomly select a point from the two point sets, respectively and then connect the two points to form a new spanning tree. The principle is shown in Fig. 5, where edge M_1 represents the edge to be removed, and edge M_2 represents the new edge. The pseudo code of mutation operator is shown in Algorithm 1.

The cognitive personal experience of particles can be written as follows.

$$S_i^t = N_2(W_i^t, c_1) = \begin{cases} C_p(W_i^t), & r_2 < c_1 \\ W_i^t, & \text{others} \end{cases} \tag{8}$$

where c_1 denotes the crossover probability between the particles and the personal optimal solution. r_2 is a random number on the interval [0, 1).

The cooperative global experience of particles can be written as follows.

$$X_i^t = N_3(S_i^t, c_2) = \begin{cases} C_g(S_i^t), & r_3 < c_2 \\ S_i^t, & \text{others} \end{cases} \tag{9}$$

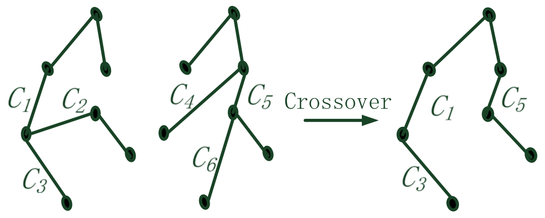

Fig. 6 Crossover operation with the Union-Find partition

Algorithm 2 Crossover operator(p, q)

```
Input: Particle $p$ and $q$
Output: New particle
  $Initialize\ each\ pin's\ partition\ to\ singletons$
  $Sort\_edge(p, u)$;//sort edge of $p$ according to the serial number of the first endpoint $u$
  $Sort\_edge(p, v)$;//sort edge of $p$ according to the serial number of the second endpoint $v$
  $Sort\_edge(q, u)$;
  $Sort\_edge(q, v)$;
  $set1 = Selecct\_same\_edge(p, q)$;
  $set2 = Select\_different\_edge(p, q)$;
  $Union\_partition(u, v, set1)$;//merge each edge of set1
  $New\ particle = Generate\_edge(set1)$;
  while $New\ particle\ is\ not\ a\ complete\ tree$ do
    $L(u, v) = Random\_select_edge(set2)$;
    if $Find\_set(u) \neq Find\_set(v)$ then
      $add\ L(u, v)\ to\ New\ particle$;
      $Union\_partition(u, v, L)$;
    end if
  end while
```

where c_2 denotes the crossover probability between the particles and the global optimal solution. r_3 is a random number on the interval [0, 1).

When implementing the crossover operator in (8) and (9), we firstly sort the edges according to the serial number of pins from small to large in both the spanning tree. Secondly, we use union-find set to choose the same edge from the two sorted tree as a set and the rest of the different edges as another set and then the first set of edges directly as the new spanning tree's edges. Finally, we select one edge randomly from the second set and add to the new spanning tree, while we prevent generating circle by using union-find set. The principle is shown in Fig. 6, where $C_1, C_2, C_3, C_4, C_5, C_6$ represent the different edges between two spanning trees, and the set C_1, C_3, C_5 represent new edges of the new spanning tree. The pseudo code of crossover operator is shown in Algorithm 2.

4.4 Parameter adjustment analysis

Property 8 The setting of inertia weight affects the balance between local search ability and global search ability of particle.

From the velocity update formula, the first part provides the flight impetus of particle in search space and represents the effect of previous velocity on the flight trajectory. Thus inertia weight is a numerical value which indicates the extent of such influence.

Property 9 Larger inertia weight will make the algorithm has strong global search ability.

Property 8 and (1) show that inertia weight decides how much previous velocity will be preserved. Thus a lager inertia weight can strengthen the capability of searching the unreached area. It is conductive to enhance the global search ability of the algorithm and jump out of the local minima. A smaller inertia weight suggests that the algorithm mainly searches near the current solution. It is conductive to enhance the local search ability and accelerate convergence.

In order to ensure a stronger global search, a larger inertia weight early in the program and a smaller one in the later stages to guarantee the local search is employed (Shi and Eberhart 1998). Simulation on four kinds of different benchmark functions showed that such strategy of parameters actually improved the performance of PSO.

Property 10 Larger acceleration coefficients c_1 may cause wandering in local scope. Larger acceleration coefficients c_2 will make the algorithm prematurely converge on local optimal solution.

Acceleration coefficients c_1 and c_2 are used in communicating between particles. According to Property 10, when the fitness value of the current particle is worse than the average fitness value of the current population, the current particle should require a greater degree of movement to the optimal particle and a larger acceleration coefficients c_2 is necessary. On the contrary, when the fitness value of the current particle is better than the average fitness value of the current population, the current particle could move within its own local scope and a larger acceleration coefficients c_1 is more helpful. In this way, the algorithm with the self-adapting parameter adjustment strategy will guarantee detailed search in local scope, not have to move directly to the position of global optimal in early phases, and speed up convergence in the later stages.

Similar to Chen et al. (2010), we have tested several kinds of the initial value of the inertia weight and the final value of the inertia weight in Oliver30 TSP which is minimization problem. Each experimental setting is conducted five runs, and each average is calculated. Consequently, the initial value $w\_start = 0.95$ and the final value of $w\_end = 0.4$ are considered as the optimal combination of parameter settings.

Based on the above analysis, we adopt the idea of linear decline proposed by Shi and Eberhart (1998) to update inertia weight w which decreases linearly from 0.95 to 0.4 according to (10). Acceleration coefficients c_1 and c_2 are updated by the self-adapting parameter adjustment strategy according to (11) and (12), respectively.

$$w = w\_start - \frac{w\_start - w\_end}{\text{evaluations}} \times \text{eval} \tag{10}$$

$$c_1 = \begin{cases} 0.5 \times \left(1 + \frac{f_{\text{ave}} - f_i}{f_{\text{ave}} - f_{\min}}\right), & f_i < f_{\text{ave}} \\ 0.5 \times \frac{f_{\max} - \text{fitness}}{f_{\max} - f_{\text{ave}}}, & \text{otherwise} \end{cases} \tag{11}$$

$$c_2 = 1 - c_1 \tag{12}$$

where evaluations represent the maximum number of iterations and eval represents the current iteration. f_i represents the fitness value of the current particle i, and f_{ave}, $f_{\max}$, $f_{\min}$ represent the average, the maximum, minimum fitness value of the current population, respectively.

4.5 Procedure of HTS-PSO

The detailed procedure of HTS-PSO can be summarized as follows.

Step 1: Load circuit net-list data and sort them in ascending order according to the size of coordinates.
Step 2: Initialize the population size, maximum iterations, inertia weight, and acceleration coefficients. Then the initial population is randomly generated.
Step 3: Calculate the fitness of each particle among the initial population according to (5). And the personal optimal solution of each particle is set as itself and select the particle with minimal fitness as the global optimal solution of the initial population.
Step 4: The proposed HTS is used as the basic operation of particle update method.
Step 5: Adjust the position and velocity of each particle according to (7)–(9), including three steps: the retention of particle's previous velocity, learning the best direction of particles, and learning the optimal direction of the current population.
Step 6: And then calculate the fitness value of each particle. If its fitness value is less than the personal optimal value, it is set as the personal optimal solution. If its fitness value is less than the global optimal value of the population, it is set as the global optimal solution.
Step 7: Check the termination condition. If fulfilled, the run is terminated and the solution is obtained. Otherwise, go to Step5.

4.6 Complexity of HTS-PSO

Lemma 1 *Assume that the population size is p, the number of iterations is $iters$, and the number of pins is n, and then, the complexity of the HTS-PSO algorithm is $O(iters \times p \times n\log n)$.*

Proof In the internal loop of the HTS-PSO algorithm, from step 2 to step 6, mutation operations, crossover operations, and fitness value calculation operations are included. For mutation and crossover operations, since the time complexity of the union-find set is slightly more than linear time, the sorting method whose complexity is $O(n\log n)$ dominates the algorithm complexity of these operations. In the calculation of fitness, the complexity of computing the length of routing tree is determined by the complexity of the sorting method. Therefore, the complexity of the internal loop of the algorithm is $O(n\log n)$. At the same time, the complexity of the external loop of the algorithm is mainly related to the size of the population and the number of iterations of the algorithm. Therefore, the complexity of the HTS-PSO algorithm is $O(iters \times p \times n\log n)$. □

4.7 Convergence analysis of HTS-PSO

Theorem 1 *The Markov chain of HTS-PSO is finite and homogeneous.*

Proof Follow the similar method of Lv et al. (2009) and Liu et al. (2015a, b, c), and we can prove that the Markov chain of HTS-PSO is finite and homogeneous. The definition of finite Markov chain can be referred as Rudolph (1994). □

Theorem 2 *Transition probability matrix of the Markov chain made up of HTS-PSO is positive definite.*

Proof Follow the similar method of Rudolph (1994) and Liu et al. (2015a, b, c), we can prove that transition probability matrix of the Markov chain made up of HTS-PSO is positive definite.

The limit theorem for Markov chain (Rudolph 1994) is the basis for the convergence of the algorithm. The limit theorem explains that the long-term probability of Markov chain does not depend on its initial states. □

Lemma 2 *If mutation probability, the algorithm is an ergodic irreducible Markov chain which has only one limited distribution and nothing to do with the initial distribution. Moreover, the probability at a random time and random state is greater than zero.*

Proof At the tth time, the jth state probability distribution of population $X(t)$ is:

$$P_j(t) = \sum_{j \in S} P_i(1) P_{ij}^{(t)}, \quad t = 1, 2, \ldots \tag{13}$$

According to Theorem 2, we can get the formulation as following:

$$P_j(\infty) = \lim_{t\to\infty} \left(\sum_{i \in S} P_i(1) P_{ij}^{(t)} \right) = \sum_{i \in S} P_i(1) P_{ij}^{(\infty)} > 0, \quad \forall j \in S \tag{14}$$

□

Definition 9 Suppose a stochastic variant $Z_t = \max\{f(x_k^{(t)}(i))|k = 1, 2, \ldots, N\}$ which represents individual best fitness at the tth step and ith state of the population. Then the algorithm converges to the global optimum, if and only if

$$\lim_{t\to\infty} P\{Z_t = Z^*\} = 1 \tag{15}$$

where $Z^* = \max\{f(x)|x \in S\}$ represents the global optimum.

Theorem 3 *For any i and j, the time transiting of an ergodic Markov chain from the ith state to the jth state is limited.*

Theorem 4 *HTS-PSO algorithm can converge to the global optimum.*

Proof Suppose that $i \in S$, $Z_t < Z^*$ and $P_i(t)$ is the probability of HTS-PSO algorithm at ith state and the tth step. Obviously $P\{Z_t \neq Z^*\} \geq P_i(t)$; hence, we can know that $P\{Z_t = Z^*\} \leq 1 - P_i(t)$.

According to Lemma 2, the ith state probability of the operator in HTS-PSO algorithm is $P_i(\infty) > 0$, then

$$\lim_{t\to\infty} P\{Z_t = Z^*\} \leq 1 - P_i(\infty) < 1 \tag{16}$$

Observe a new population such as $X_t^+ = \{Z_t, X_t\}, t \geq 1, x_{ti} \in S$ denoting the search space (which is a finite set or a countable set), where Z_t, the same to that in Definition 9, represents individual best fitness in current population. X_t denotes the population during the search. As it is easy to prove that the group shift process $\{X_t^+, t \geq 1\}$ is still a homogeneous and ergodic Markov chain, we can know that

$$\begin{aligned} & P_j^+(t) = \sum_{i \in S} P_i^+(1) P_{ij}^+(t) \\ & P_{ij}^+ > 0\,(\forall i \in S, \forall j \in S_0) \\ & P_{ij}^+ = 0\,(\forall i \in S, \forall j \notin S_0) \end{aligned} \tag{17}$$

So

$$\begin{aligned} & (P_{ij}^+)^t \to 0\,(t \to \infty) \\ & P_j^+(\infty) \to 0\,(j \notin S_0) \\ & \lim_{t\to\infty} P\{Z_t = Z^*\} = 1 \end{aligned} \tag{18}$$

□

5 Experiment results

In order to verify the performance and effectiveness of the algorithm and its related strategies, experimental comparisons are shown in detail from Sects 5.1–5.8. Two benchmark

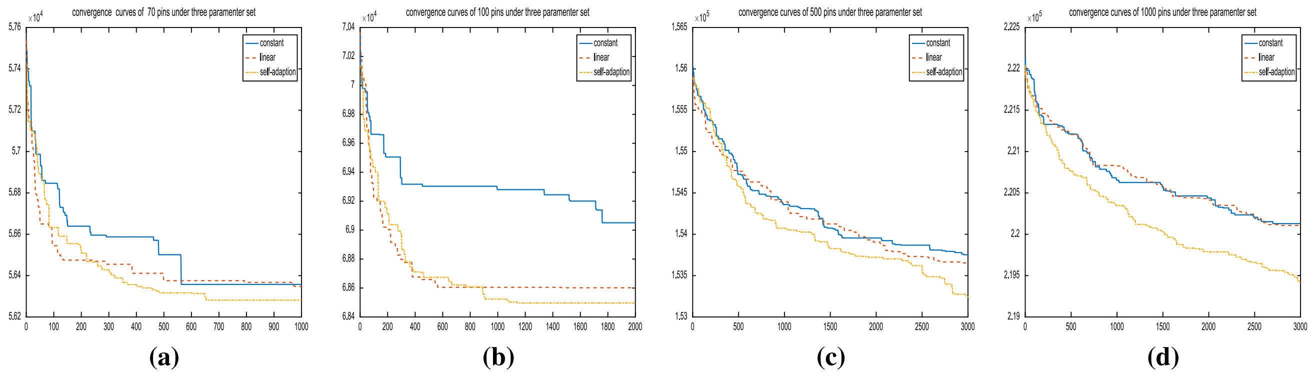

Fig. 7 The algorithm convergence comparison based on the three adjustment strategies. **a** Test 6: 70 pins, **b** Test 7: 100 pins, **c** Test 9: 500 pins, **d** Test 10: 1000 pins

circuit suites, namely GEO and ISPD, respectively, are used in this paper (Alpert 1998, Zachariasen 2003). The circuit scales of GEO and ISPD are given in Tables 2 and 18, respectively. It can be seen from Tables 2 and 18, the scale of GEO including the number of total pins from 8 to 1000, while the scale of ISPD including the number of total pins from 44,266 to 26,900 and the number of nets from 11,507 to 64,227.

5.1 Validation of the self-adapting parameter adjustment strategy

For the adjustment strategy of the acceleration coefficients in PSO, Ratnaweera et al. (2004) proposed a kind of strategy which employs a lager c_1 and a smaller c_2 in the early phases and the opposite in the later. Similar to Chen et al. (2010), $c_1 = 0.82 - 0.5$ and $c_2 = 0.4 - 0.83$ are considered as the optimal combination of parameter settings. The acceleration coefficients of our algorithm are based on the proposed self-adapting parameter adjustment strategy. The adaptive adjustment strategy of the acceleration coefficients is compared with the linear adjustment strategy (Ratnaweera 2004) and the constant strategy (Eberhar and Kennedy 1995), and then, the convergence of the three algorithms is shown in Fig. 7.

For the test circuits 6, 7, 9, 10, it can be seen that the proposed algorithm based on the self-adapting parameter adjustment has the best convergence performance, the linear parameter adjustment strategy is in the middle, and the constant strategy is the worst. Similarly, in other test circuits, the proposed self-adapting parameter strategy has the better convergence effect than others. In view of the limitation of space, the convergence comparison of other test circuits is not shown in the paper. In summary, the self-adapting parameter adjustment strategy proposed in this paper can further increase the diversity of the population and accelerate the convergence.

Table 10 The influence of HTS on wirelength optimization under different threshold values

Threshold value	Imp/RSMT (%)	Imp/OSMT (%)
0.0	9.68	0.00
0.1	10.07	0.44
0.2	10.10	0.46
0.3	10.04	0.40
0.4	10.02	0.38
0.5	10.01	0.37
0.6	9.96	0.31
0.7	9.91	0.25
0.8	9.83	0.17
0.9	9.69	0.01
1.0	9.67	– 0.01

5.2 Validation of hybrid transformation strategies

In previous sections, HTS divides the whole iterative process into the first half and the second half, and the threshold parameter for the partition is 0.5. To selection the better threshold parameter for our proposed algorithm, we first range the threshold parameter α from 0.0 to 1.0 with increasing the value by 0.1. Secondly, we implement the proposed algorithm based on the threshold parameter that have been set already. Finally, the experimental results are listed in Table 10.

From Table 10, the proposed algorithm has the best effect on the proposed HTS when the threshold value is 0.2. Based on $\alpha = 0.2$, our algorithm achieves 10.10% shorter wirelength than RSMT and 0.46% shorter wirelength than OSMT. Therefore, the threshold parameter of the proposed algorithm is set as 0.2.

5.3 Versatility validation of the proposed HTS

The efficient HTS in this paper can be applied to a variety of evolutionary algorithms, including PSO, GA, etc. Thus it has

Table 11 The comparison results between HTS based on GA and two SMT construction algorithms

Circuit	RSMT	OSMT	GA	Imp/RSMT (%)	Imp/OSMT(%)
1	17,928	16,918	16,925	5.60	– 0.04
2	20,478	18,041	18,023	11.99	0.10
3	21,969	19,696	19,397	11.71	1.52
4	35,675	32,207	32,052	10.16	0.48
5	53,518	48,020	48,112	10.10	– 0.19
6	62,633	56,433	56,180	10.30	0.45
7	75,584	68,855	69,016	8.69	– 0.23
8	158,206	141,894	141,778	10.38	0.08
9	170,924	154,825	154,490	9.61	0.22
10	246,731	221,090	221,028	10.42	0.03
Average				9.90	0.24

good versatility. The application of the proposed HTS in two typical evolutionary algorithms, including PSO and GA, is designed in this paper, and the experimental results are given in Table 10 and Table 11, respectively. The two algorithms achieve the 9.90% and 10.10% reduction on the wirelength than RSMT, respectively. Meanwhile, the two algorithms achieve 0.24% and 0.46% reduction on the wirelength than RSMT, respectively.

5.4 Validation of HTS for rectilinear architecture

We further apply the proposed HTS for the rectilinear architecture, that is, the RSMT construction based on our proposed HTS, and the experimental results are shown in Table 12. In Table 12, HRSMT indicates that the proposed HTS is involved, while RSMT indicates that HTS is not involved. Experimental results have shown that the proposed HTS designed for the rectilinear architecture can achieve 2.46% of the average wirelength optimization than the method without HTS.

In summary, the proposed HTS can bring better wirelength optimization ability for both the rectilinear architecture and octagonal architecture, and it provides a good application prospect for designing the unified algorithm and solving the routing problem based on different architectures. Finally, from Table 12, not only can the proposed HTS further enhance the wirelength optimization of rectilinear architecture, but also has an ability to obtain accurate solutions.

5.5 Validation of multiple topologies

For Circuit 3, four different topologies can be obtained by running the proposed algorithm several times, as shown in Fig. 8. In fact, there are more than 4 kinds of topologies available with the same wirelength for Circuit 3, and only four topologies are shown in Fig. 8 due to limited space. The experimental results show that the proposed algorithm can obtain many different topologies of RSMT and OSMT under the same or near optimal wirelength. The ability to obtain multiple Steiner trees with different topologies can help to provide different topology options for congestion optimization in the global routing stage.

Table 12 The effectiveness of the RSMT construction algorithm based on HTS

Circuit	RSMT	HRMST	Imp/RSMT (%)
1	17,928	17,693	1.31
2	20,478	19,797	3.33
3	21,969	21,226	3.38
4	35,675	35,072	1.69
5	53,518	52,025	2.79
6	62,633	61,129	2.40
7	75,584	74,416	1.55
8	158,206	153,672	2.87
9	170,924	166,592	2.53
10	246,731	239,824	2.80
Average			2.46

5.6 Statistical results of several population-based techniques

To verify the superiority of the proposed method in terms of robustness, statistical results of the proposed method compared with two well-established population-based techniques are shown in Tables 13–16. These two compared population-based techniques are a differential evolution(DE) technique proposed by Manna et al. (2015), and an artificial bee colony optimization technique (ABC) proposed by Bhattacharya et al. (2014). The experimental results of Table 13 and Table 14 are based on the benchmark circuits GEO, while the ones of Table 15 and Table 16 are based on the benchmark circuits ISPD.

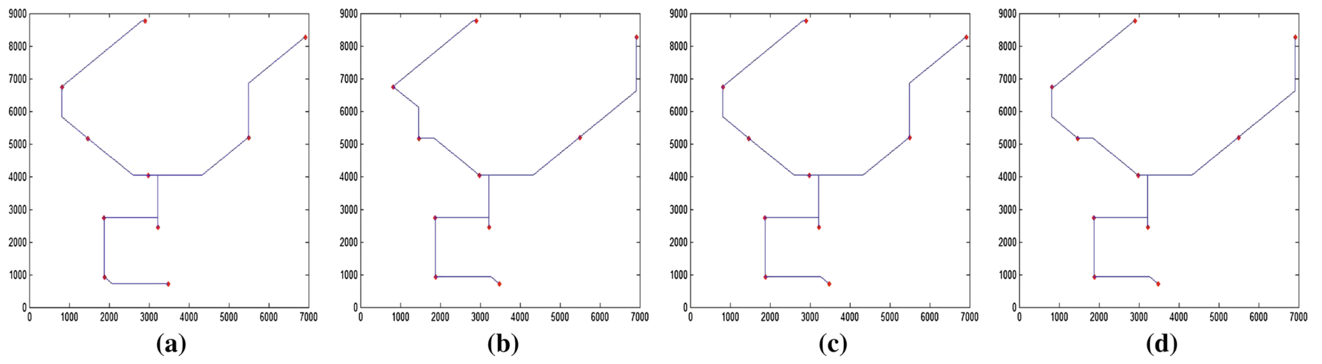

Fig. 8 Four different topologies with the same wirelength for Circuit 3

Table 13 Statistical results of several population-based techniques in the circuits of GEO (1)

Circuit	Best value						Mean value					
	Absolute values			Normalized values			Absolute values			Normalized values		
	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours
1	16,918	17,506	16,900	1.001	1.036	1.000	16,918	17,540	16,921	1.000	1.037	1.000
2	18,041	18,041	18,023	1.001	1.001	1.000	18,041	18,041	18,023	1.001	1.001	1.000
3	19,696	19,703	19,397	1.015	1.016	1.000	19,696	19,734	19,397	1.015	1.017	1.000
4	32,250	32,810	32,039	1.007	1.024	1.000	32,405	32,849	32,063	1.011	1.025	1.000
5	48,196	49,138	47,883	1.007	1.026	1.000	48,603	49,184	48,027	1.012	1.024	1.000
6	56,984	57,547	56,086	1.016	1.026	1.000	57,447	57,596	56,350	1.019	1.022	1.000
7	70,414	70,368	68,427	1.029	1.028	1.000	70,548	70,416	68,625	1.028	1.026	1.000
8	143,242	142,996	140,520	1.019	1.018	1.000	143,681	143,183	140,898	1.020	1.016	1.000
9	156,352	155,742	153,377	1.019	1.015	1.000	156,533	155,929	153,708	1.018	1.014	1.000
10	222,450	221,818	219,637	1.013	1.010	1.000	222,626	222,028	219,954	1.012	1.009	1.000
Average				1.013	1.020	1.000				1.014	1.019	1.000

Table 14 Statistical results of several population-based techniques in the circuits of GEO (2)

Circuit	SD						Runtime (in s)					
	Absolute values			Normalized values			Absolute values			Normalized values		
	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours
1	0.0	100.8	15.7	0.000	6.404	1.000	0.12	0.37	0.12	1.000	3.033	1.000
2	0.0	0.0	0.0	/	/	1.000	0.14	0.41	0.13	1.075	3.060	1.000
3	0.0	93.6	0.0	/	/	1.000	0.16	0.46	0.16	1.058	2.968	1.000
4	71.2	99.9	31.0	2.300	3.227	1.000	0.39	1.07	0.31	1.272	3.511	1.000
5	259.3	70.4	93.9	2.762	0.750	1.000	1.29	4.17	0.89	1.447	4.688	1.000
6	231.4	41.0	144.8	1.598	0.283	1.000	2.01	6.52	1.41	1.432	4.639	1.000
7	55.1	25.9	149.5	0.369	0.173	1.000	3.10	11.32	2.20	1.410	5.147	1.000
8	168.5	102.9	187.6	0.898	0.549	1.000	15.78	74.23	13.31	1.185	5.576	1.000
9	142.7	113.9	180.5	0.790	0.631	1.000	19.62	95.52	17.08	1.149	5.593	1.000
10	100.3	103.2	184.1	0.545	0.560	1.000	42.55	276.33	40.00	1.064	6.909	1.000
Average				1.158	1.572	1.000				1.209	4.513	1.000

Table 15 Statistical results of several population-based techniques in the circuits of ISPD (1)

Circuit	Best value						Mean value					
	Absolute values			Normalized values			Absolute values			Normalized values		
	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours
ibm01	56,730	56,722	56,139	1.011	1.010	1.000	56,734	56,726	56,145	1.010	1.010	1.000
ibm02	156,480	157,287	155,122	1.009	1.014	1.000	156,495	157,299	155,135	1.009	1.014	1.000
ibm03	135,217	135,661	134,147	1.008	1.011	1.000	135,234	135,671	134,165	1.008	1.011	1.000
ibm04	151,006	151,539	149,871	1.008	1.011	1.000	151,024	151,549	149,879	1.008	1.011	1.000
ibm06	258,951	260,638	257,013	1.008	1.014	1.000	258,982	260,666	257,030	1.008	1.014	1.000
ibm07	338,482	340,042	335,859	1.008	1.012	1.000	338,512	340,060	335,874	1.008	1.012	1.000
ibm08	376,283	378,120	372,903	1.009	1.014	1.000	376,312	378,132	372,938	1.009	1.014	1.000
ibm09	385,513	387,186	382,695	1.007	1.012	1.000	385,549	387,209	382,722	1.007	1.012	1.000
ibm10	537,828	539,909	533,192	1.009	1.013	1.000	537,857	539,940	533,218	1.009	1.013	1.000
Average				1.008	1.012	1.000				1.008	1.012	1.000

Table 16 Statistical results of several population-based techniques in the circuits of ISPD (2)

Circuit	SD						Runtime (in min)					
	Absolute values			Normalized values			Absolute values			Normalized values		
	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours	ABC	DE	Ours
ibm01	3.1	2.0	3.0	1.037	0.671	1.000	0.98	3.35	0.75	1.310	4.454	1.000
ibm02	7.0	6.8	6.8	1.038	1.000	1.000	1.88	6.27	1.41	1.327	4.433	1.000
ibm03	12.9	6.4	12.3	1.051	0.524	1.000	1.63	5.40	1.13	1.443	4.791	1.000
ibm04	7.2	5.2	6.0	1.201	0.870	1.000	1.89	6.41	1.34	1.410	4.786	1.000
ibm06	23.6	11.4	11.9	1.985	0.957	1.000	2.71	8.77	1.95	1.386	4.491	1.000
ibm07	20.3	12.6	8.3	2.434	1.507	1.000	3.53	11.46	2.62	1.347	4.373	1.000
ibm08	22.4	7.9	24.5	0.918	0.323	1.000	4.73	15.65	3.49	1.356	4.485	1.000
ibm09	26.0	11.7	15.1	1.719	0.775	1.000	4.07	13.16	2.99	1.364	4.406	1.000
ibm10	18.8	19.4	13.7	1.375	1.418	1.000	6.11	19.86	4.53	1.349	4.381	1.000
Average				1.418	0.894	1.000				1.366	4.511	1.000

Each algorithm runs 20 times. The population size of the three population-based techniques is set to 100, while the maximum iterations is set to 1000. In Table 13, on the best value, our algorithms outperform ABC and DE by 1.3% and 2.0%, respectively, while on the mean value, 1.4% and 1.9%, respectively. In Table 14, on the standard deviation, our algorithms outperform ABC and DE by 15.8% and 57.2%, respectively, while on the runtime, it speeds up 1.209X and 4.513X, respectively. Similar to the results of the benchmark circuit GEO, the results of the benchmark circuit ISPD in Tables 15 and 16 show our algorithm achieve the better values than ABC and DE on the best value, the mean value, the standard deviation, and the runtime, respectively (except for the standard deviation compared with DE).

5.7 Compared with recent published methods

To verify the superiority of the proposed method, four recent published methods (Kundu et al. 2017, namely KNN, Kundu et al. 2016, namely SAT, Manna et al. 2015; Bhattacharya et al. 2014) are used to compared with our proposed algorithm, and the results are shown in Tables 17 and 18, respectively. Our algorithm, ABC, and DE are all population-based techniques, and thus, the wirelength and runtime are both the mean values of 20 runs. In Table 17, our algorithm reduces 9.93% and 8.65% than SAT and KNN on wirelength, respectively, while it speeds up 1306.61X and 8.13X, respectively. In Table 18, our algorithm reduces 0.83% and 1.22% than ABC and DE on wirelength, respectively, while it speeds up 1.37X and 4.51X, respectively.

5.8 Compared with three typical evolutionary algorithms

To further verify the superiority of the proposed PSO algorithm, three typical evolutionary algorithms (Moscato 1989, namely MA, Numaoka 1996, namely BEA, Holland 1992 namely GA) are redesigned for our problem and used to com-

Table 17 Compared with SAT and KNN

Circuit	# Nets	# Points	Wirelength					Runtime (in min)				
			SAT	KNN	Ours	Imp/SAT	Imp/KNN	SAT	KNN	Ours	Imp/SAT (%)	Imp/KNN (%)
ibm01	11,507	44,266	61,005	61,071	56,145	7.97	8.07	152.63	3.77	0.75	203.03	5.01
ibm02	18,429	78,171	172,518	167,359	155,135	10.08	7.30	1144.17	9.75	1.41	808.73	6.89
ibm03	21,621	75,710	150,138	147,982	134,165	10.64	9.34	1817.97	10.75	1.13	1612.51	9.54
ibm04	26,163	89,591	164,998	164,838	149,879	9.16	9.07	1993.45	11.55	1.34	1488.82	8.63
ibm06	33,354	124,299	289,705	280,998	257,030	11.28	8.53	3952.85	20.85	1.95	2023.80	10.67
ibm07	44,394	164,369	368,015	368,780	335,874	8.73	8.92	4385.32	23.95	2.62	1673.66	9.14
ibm08	47,944	198,180	431,879	413,201	372,938	13.65	9.74	5844.37	27.82	3.49	1674.72	7.97
ibm09	50,393	187,872	418,382	417,543	382,722	8.52	8.34	3333.63	23.08	2.99	1116.22	7.73
ibm10	64,227	269,000	588,079	583,102	533,218	9.33	8.55	5250.10	34.50	4.53	1157.98	7.61
Average						9.93	8.65				1306.61	8.13

Table 18 Compared with ABC and DE

Circuit	Wirelength					Runtime (in min)				
	ABC	DE	Ours	Imp/ABC	Imp/DE	ABC	DE	Ours	Imp/ABC (%)	Imp/DE (%)
ibm01	56,734	56,726	56,145	1.04	1.02	0.98	3.35	0.75	1.31	4.45
ibm02	156,495	157,299	155,135	0.87	1.38	1.88	6.27	1.41	1.33	4.43
ibm03	135,234	135,671	134,165	0.79	1.11	1.63	5.40	1.13	1.44	4.79
ibm04	151,024	151,549	149,879	0.76	1.10	1.89	6.41	1.34	1.41	4.79
ibm06	258,982	260,666	257,030	0.75	1.39	2.71	8.77	1.95	1.39	4.49
ibm07	338,512	340,060	335,874	0.78	1.23	3.53	11.46	2.62	1.35	4.37
ibm08	376,312	378,132	372,938	0.90	1.37	4.73	15.65	3.49	1.36	4.48
ibm09	385,549	387,209	382,722	0.73	1.16	4.07	13.16	2.99	1.36	4.41
ibm10	537,857	539,940	533,218	0.86	1.24	6.11	19.86	4.53	1.35	4.38
Average				0.83	1.22				1.37	4.51

Table 19 Compared with MA and BEA (1)

Circuit	Best value						Mean value					
	Absolute values			Normalized values			Absolute values			Normalized values		
	MA	BEA	Ours	MA	BEA	Ours	MA	BEA	Ours	MA	BEA	Ours
1	16,918	17,488	16,900	1.001	1.035	1.000	17,197	17,678	16,900	1.018	1.046	1.000
2	18,040	18,041	18,023	1.001	1.001	1.000	18,083	18,276	18,023	1.003	1.014	1.000
3	19,695	19,703	19,397	1.015	1.016	1.000	19,701	19,880	19,397	1.016	1.025	1.000
4	32,511	32,836	32,090	1.013	1.023	1.000	32,684	33,093	32,154	1.016	1.029	1.000
5	48,580	49,138	47,902	1.014	1.026	1.000	48,914	49,322	48,040	1.018	1.027	1.000
6	57,213	57,612	56,444	1.014	1.021	1.000	57,392	57,695	56,526	1.015	1.021	1.000
7	70,044	70,486	68,905	1.017	1.023	1.000	70,230	70,594	69,128	1.016	1.021	1.000
8	143,136	143,535	141,512	1.011	1.014	1.000	143,465	143,667	141,883	1.011	1.013	1.000
9	155,882	156,172	154,514	1.009	1.011	1.000	156,270	156,481	154,782	1.010	1.011	1.000
10	222,032	222,324	220,818	1.005	1.007	1.000	222,434	222,543	221,052	1.006	1.007	1.000
Average				1.010	1.018	1.000				1.013	1.021	1.000

Table 20 Compared with MA and BEA (2)

Circuit	SD						Runtime (in s)					
	Absolute values			Normalized values			Absolute values			Normalized values		
	MA	BEA	Ours	MA	BEA	Ours	MA	BEA	Ours	MA	BEA	Ours
1	193.5	163.6	0.0	/	/	1.000	0.20	0.21	0.04	4.900	5.150	1.000
2	58.2	154.2	0.0	/	/	1.000	0.24	0.21	0.05	5.333	4.600	1.000
3	3.5	159.6	0.0	/	/	1.000	0.28	0.20	0.06	5.018	3.691	1.000
4	94.5	129.4	44.5	2.123	2.907	1.000	0.68	0.25	0.10	6.657	2.431	1.000
5	172.9	107.9	89.5	1.933	1.206	1.000	2.22	0.47	0.29	7.769	1.636	1.000
6	103.9	47.1	76.1	1.366	0.619	1.000	3.55	0.55	0.52	6.876	1.060	1.000
7	97.8	51.8	147.2	0.664	0.352	1.000	5.84	0.76	0.70	8.406	1.092	1.000
8	134.3	93.2	188.0	0.714	0.496	1.000	37.28	2.85	4.18	8.923	0.682	1.000
9	170.8	158.1	157.5	1.085	1.004	1.000	47.56	3.81	5.45	8.729	0.700	1.000
10	146.0	102.5	111.5	1.309	0.920	1.000	110.87	8.00	13.00	8.530	0.615	1.000
Average				1.313	1.072	1.000				7.114	2.166	1.000

Table 21 Compared with GA (1)

Circuit	Best value				Mean value			
	Absolute values		Normalized values		Absolute values		Normalized values	
	GA	Ours	GA	Ours	GA	Ours	GA	Ours
1	17,255	16,900	1.021	1.000	17,722	16,900	1.049	1.000
2	18,383	18,023	1.020	1.000	18,377	18,023	1.020	1.000
3	19,707	19,397	1.016	1.000	19,938	19,397	1.028	1.000
4	33,149	32,090	1.033	1.000	33,166	32,154	1.031	1.000
5	49,147	47,902	1.026	1.000	49,362	48,040	1.028	1.000
6	57,573	56,444	1.020	1.000	57,714	56,526	1.021	1.000
7	70,490	68,905	1.023	1.000	70,557	69,128	1.021	1.000
8	143,352	141,512	1.013	1.000	143,725	141,883	1.013	1.000
9	156,059	154,514	1.010	1.000	156,443	154,782	1.011	1.000
10	222,364	220,818	1.007	1.000	222,580	221,052	1.007	1.000
Average			1.019	1.000			1.023	1.000

Table 22 Compared with GA (2)

Circuit	SD				Runtime (in s)			
	Absolute values		Normalized values		Absolute values		Normalized values	
	GA	Ours	GA	Ours	GA	Ours	GA	Ours
1	196.5	0.0	/	1.000	0.06	0.04	1.375	1.000
2	0.0	0.0	/	1.000	0.06	0.05	1.333	1.000
3	153.5	0.0	/	1.000	0.07	0.06	1.182	1.000
4	11.0	44.5	0.248	1.000	0.13	0.10	1.275	1.000
5	98.8	89.5	1.104	1.000	0.33	0.29	1.140	1.000
6	76.1	76.1	1.000	1.000	0.48	0.52	0.922	1.000
7	43.2	147.2	0.293	1.000	0.72	0.70	1.035	1.000
8	157.2	188.0	0.836	1.000	3.54	4.18	0.846	1.000
9	178.5	157.5	1.134	1.000	4.43	5.45	0.814	1.000
10	134.7	111.5	1.208	1.000	10.22	13.00	0.786	1.000
Average			0.832	1.000			1.071	1.000

pare with the proposed PSO algorithm. The results are shown in Tables 19, 20, 21, and 22, respectively. Our algorithm, MA, BEA, and GA are all typical evolutionary algorithms, and thus, the wirelength and runtime are both the mean values of 20 runs. The population size of the four algorithms is set to 50, while the maximum iterations is set to 500. In Table 19, on the best value, our algorithm outperforms MA and BAE by 1.0% and 1.8%, respectively, while on the mean value, 1.3% and 2.1%, respectively. In Table 20, on the standard deviation, our algorithm outperforms MA and BAE by 31.3% and 7.2%, respectively, while on the runtime, it speeds up 7.114X and 2.166X, respectively. In Table 21, on the best value, our algorithm outperforms GA by 1.9%, while on the mean value, 2.3%. In Table 22, on the standard deviation, GA outperforms our algorithm by 16.8%, while on the runtime, it speeds up 1.071X.

6 Conclusions

In this paper, we design an efficient hybrid transformation strategy, which has achieved good wirelength optimization compared with other transformation strategy and then select the best threshold parameter for our proposed algorithm. Meanwhile, in order to enhance the diversity of PSO algorithm, a self-adapting adjustment strategy is proposed to update the acceleration coefficients. Further, an efficient hybrid transformation strategy with PSO is proposed to construct the OSMT. Compared with the recently published methods , it can obtain the best wirelength with the least runtime. The proposed algorithm can obtain different topologies of OSMT, which is conducive to the congestion optimization in VLSI global routing stage.

Then the proposed hybrid transformation strategy can be applied in PSO and genetic algorithm, and it shows that it has very good versatility for evolutionary algorithms. Finally, the proposed hybrid transformation strategy is well extended to the rectilinear architecture, so that the hybrid transformation strategy can be applied effectively in different routing architecture. It overcomes the difficulty of designing different algorithms for different routing architectures in previous work and provides help and guidance for designing a unified algorithm to solve routing problems under different routing architectures. The final experimental results show the feasibility and effectiveness of the related strategies and algorithms in this paper.

Acknowledgements The authors thank the reviewers for their valuable comments/suggestions which helped to improve the quality of this paper significantly. This work was supported in part by the National Natural Science Foundation of China under Grants No. 61877010 and No.11501114, the National Basic Research Program of China No. 2011CB808000, and the Fujian Natural Science Funds under Grant No. 2019J01243, the project of Technology Innovation Platform of Fujian Province (Grant No. 2014H2005 and No. 2009J1007), and the Fujian Collaborative Innovation Center for Big Data Applications in Governments.

Compliance with ethical standards

Conflict of interest All the authors declare that they have no conflict of interest.

Ethical approval This article does not contain any studies with human participants performed by any of the authors.

References

Agrawal S, Silakari S (2014) FRPSO: Fletcher-Reeves based particle swarm optimization for multimodal function optimization. Soft Comput 18(11):2227–2243

Alpert CJ (1998) The ISPD98 circuit benchmark suite. In: Proceedings of the 1998 international symposium on Physical design, pp 80–85

Arora T, Mose ME (2009) Ant colony optimization for power efficient routing in manhattan and non-manhattan VLSI architectures. In: Proceedings of the Swarm intelligence symposium, pp 137–144

Bhattacharya P, Khan A, Sarkar, SK (2014) A global routing optimization scheme based on ABC algorithm. In: Proceedings of the 2nd international conference on advanced computing, networking and informatics, pp 189–197

Chang YJ, Lee YT, Gao JR, Wu PC, Wang TC (2010) NTHU-Route. 20: a robust global router for modern designs. IEEE Trans Comput Aid Des 29(12):1931–1944

Chen G, Guo W, Chen Y (2010) A PSO-based intelligent decision algorithm for VLSI floorplanning. Soft Comput 14(12):1329–1337

Chiang C, Chiang CS (2002) Octilinear steiner tree construction. In: Proceedings of the 45th midwest symposium on circuits and systems, pp 603–606

Costas V, Konstantinos E, Isaac E (2012) Particle swarm optimization with deliberate loss of information. Soft Comput 16(8):1373–1392

Coulston CS (2003) Constructing exact octagonal steiner minimal trees. In: Proceedings of the 13th ACM Great Lakes symposium on VLSI, pp 1–6

Dai KR, Liu WH, Li YL (2009) Efficient simulated evolution based rerouting and congestion-relaxed layer assignment on 3-D global routing. In: Proceedings of Asia and South Pacific Design Automation Conference, pp 570V575

Dai KR, Liu WH, Li YL (2012) NCTU-GR: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing. IEEE Trans VLSI Syst 20(3):459–472

Eberhar RC, Kennedy J (1995) A new optimizer using particles swarm theory. In: Proceedings of the 6th international symposium on micro machine and human science, pp 39–43

Garey M, Johnson D (1977) The rectilinear steiner tree problem is NP-complete. SIAM J Appl Math 32:826–834

Gottlieb J, Julstrom BA, Raidl GR, Rothlauf F (2001) Prufer Numbers: a poor representation of spanning trees for evolutionary search. In: Proceedings of the genetic and evolutionary computation conference, pp 343–350

Guo WZ, Liu GG, Chen GL, Peng SJ (2014) A hybrid multi- objective PSO algorithm with local search strategy for VLSI partitioning. Front Comput Sci 8(2):203–216

Han Y, Ancajas DM, Chakraborty K, Roy S (2014) Exploring high-throughput computing paradigm for global routing. IEEE Trans VLSI Syst 22(1):155–167

Held S, Muller D, Rotter D, Scheifele R, Traub V, Vygen J (2017) Global routing with timing constraints. IEEE Trans Comput Aid Des 1–17

Holland JH (1992) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press, Cambridge

Huang X, Guo W, Liu G, Chen G (2016) FH-OAOS: a fast four-step heuristic for obstacle-avoiding octilinear steiner tree construction. ACM Trans Des Automat Elec 21(3):1–31

Huang X, Guo W, Liu G, Chen G (2017) MLXR: multi- layer obstacle-avoiding X-architecture steiner tree construction for VLSI routing. Sci China Inform Sci 60(1):1–3

Khan A, Laha S, Sarkar SK (2013) A novel particle swarm optimization approach for VLSI routing. In: Proceedings of advance computing conference, pp 258–262

Koh CK, Madden PH (2000) Manhattan or non-manhattan? a study of alternative VLSI routing architectures. In: Proceedings of Great Lake symposium on VLSI, pp 47–52

Kundu S, Roy S, Mukherjee S (2016) SAT based rectilinear steiner tree construction. In: Proceedings of Appl. Theor Comput Commun Tech, pp 623–627

Kundu S, Roy S, Mukherjee S (2017) K-nearest neighbour (KNN) approach using SAT based technique for rectilinear steiner tree construction. In: Proceedings of embedded computer systems design, pp 1–5

Liu G, Chen G, Guo W, Chen Z (2011) DPSO-based rectilinear steiner minimal tree construction considering bend reduction. In: Proceedings of the 7th international conference on natural computation, pp 1161–1165

Liu G, Chen G, Guo W (2012) DPSO based octagonal steiner tree algorithm for VLSI routing. In: Proceedings of the 5th international conference on advanced computational intellligence, pp 383–387

Liu G, Guo W, Li R, Niu Y, Chen G (2015) XGRouter: high-quality global router in X-architecture with particle swarm optimization. Front Comput Sci 9(4):576–594

Liu G, Guo W, Niu Y, Chen G, Huang X (2015) A PSO-based timing-driven Octilinear Steiner tree algorithm for VLSI routing considering bend reduction. Soft Comput 19(5):1153–1169

Liu G, Huang X, Guo W, Niu Y, Chen G (2015) Multilayer obstacle-avoiding X-architecture steiner minimal tree construction based on particle swarm optimization. IEEE Trans Cybern 45(5):989–1002

Liu H, Cai Z, Wang Y (2010) Hybridizing particle swarm optimization with differential evolution for constrained numerical and engineering optimization. Appl Soft Comput 10(2):629–640

Liu W, Kao W, Li Y, Chao K (2013) NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing. IEEE Trans Comput Aid Des 32(5):709–722

Lv H, Zheng J, Zhou C, Zhou, C, Li K (2009) The convergence analysis of genetic algorithm based on space mating. In: Proceeding of the 5th international conference on natural computation, pp 557–562

Ma J, Yang B, Ma SH (2000) A near-optimal approximation algorithm for Manhattan Steiner Tree. J Soft 11(2):260–264 (in Chinese)

Manna S, Chakrabarti T, Sharma U, Sarkar SK (2015) Efficient VLSI routing optimization employing discrete differential evolution technique. In: Proceedings of Recent Trends in Information Systems, pp 461–464

Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Caltech Concurr Comput Progr C3P Rep 826: 1989

Numaoka C (1996) Bacterial evolution algorithm for rapid adaptation. In: Proceedings of European workshop on modelling autonomous agents in a multi-agent world, pp 139–148

Rada-Vilela J, Zhang M, Seah W (2013) A performance study on synchronicity and neighborhood size in particle swarm optimization. Soft Comput 17(6):1019–1030

Ratnaweera A, Halgamuge SK, Watson HC (2004) Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans Evol Comput 8(3):240–255

Rudolph G (1994) Convergence analysis of canonical genetic algorithms. IEEE Trans Neural Network 5(1):96–101

Ruiz-Cruz R, Sanchez EN, Ornelas-Tellez F, Loukianov AG, Harley RG (2013) Particle swarm optimization for discrete-time inverse optimal control of a doubly fed induction generator. IEEE Trans Cybern 43(6):1698–1709

Samanta T, Ghosal P, Rahaman H, Dasgupta PS (2006) A heuristic methiod for constructing hexagonal steiner minimal trees for routing in VLSI. In: IEEE international symposium on circuits and systems, pp 1788–1791

Samanta T, Rahaman H, Dasgupta PS (2011) Near-optimal Y-routed delay trees in nanometric interconnect design. IET Comput Digit Tec 5(1):36–48

Scheifele R (2016) RC-aware global routing. In: IEEE/ACM international conference on computer-aided design, pp 1–8

Scheifele R (2017) Steiner trees with bounded RC-delay. Alogithmica 78(1):86–109

Shi YH, Eberhart RC (1998) A modified particle swarm optimizer. In: Proceedings of the IEEE international conference of evolutionary computation, pp 69–73

Siddiqi UF, Sait SM (2017) A game theory based post-processing method to enhance VLSI global routers. IEEE Access 5:1328–1339

Thurber A, Xue G (1999) Computing hexagonal steiner trees using PCx for VLS. In: Proceedings of the 6th IEEE international conference on electronic, circuits, and system, pp 381–384

Wang D, Tan D, Liu L (2017) Particle swarm optimization algorithm: an overview. Soft Comput 1–22

Xu N, Hong XL (2009) Very large scale integration physical design theory and method. Tsinghua University Press, Tsinghua (in Chinese)

Xu X, Rong H, Trovati M, Liptrott M, Bessis N (2016) CS-PSO: chaotic particle swarm optimization algorithm for solving combinatorial optimization problems. Soft Comput 1–13

Xue B, Zhang MJ, Browne MN (2013) Particle swarm optimization for feature selection in classification: a multi-objective approach. IEEE Trans Cybern 43(6):1656–1671

Yan JT (2008) Timing-driven octilinear steiner tree construction based on steiner-point reassignment and path reconstruction. ACM Trans Des Automat El 13(2):26

Zachariasen M (2003) GeoSteiner Homepage. Available: http://www.diku.dk/geosteiner

Zhu Q, Zhou H, Jing T, Hong XL, Yang Y (2005) Spanning graph-based nonrectilinear Steiner tree allgorithms. IEEE Trans Comput Aid Des 24(7):1066–1075